Dissertation zur Erlangung des Doktorgrades Dr.rer.nat. der Fakultät für Ingenieurwissenschaften und Informatik der Universität Ulm

Hybrid Planning & Scheduling

Bernd Schattenberg aus Laupheim

Institut für Künstliche Intelligenz
Fakultät für Ingenieurwissenschaften und Informatik
Universität Ulm

März 2009

Amtierender Dekan: Prof. Dr. Michael Weber

Gutachter: Prof. Dr. Susanne Biundo-Stephan
Prof. Dr. Friedrich von Henke

Tag der Promotion: 28. April 2009

Preface

Planning and scheduling are well-established disciplines in the field of Artificial Intelligence. They provide flexibility, robustness, and effectiveness to complex software systems in a variety of application areas. While planning is the process of finding a course of action that achieves a goal or performs a specified task, scheduling deals with the assignment of resources and time to given activities, taking into account resource restrictions and temporal dependencies. In other words, planning focuses on reasoning about causal structures and identifying the necessary actions for achieving a specific goal; scheduling concentrates on resource consumption and production for optimizing a coherent parameter assignment of a plan. As successful these techniques clearly are, the actual demands of complex, real-world applications go far beyond the potential of these single methods, however. They require an adequate *integration* of these problem solving methods as well as a *combination* of different planning and scheduling paradigms. Particularly important are abstraction-based, hierarchical approaches because of both their expressive knowledge representation and their efficiency in finding solutions. Current state-of-the-art systems rarely address the question of method integration; isolated approaches do so only in ad hoc implementations and mostly lack a proper formal basis.

This thesis presents a formal framework for plan and schedule generation based on a well-founded conceptualization of *refinement planning*: An abstract problem specification is transformed stepwise into a concrete, executable solution. In each refinement step, plan deficiencies identify faulty or under-developed parts of the plan, which in turn triggers the generation of transformation operators that try to resolve them. All involved entities are explicitly represented and therefore transparent to the framework. This property allows for two novel aspects of our approach: First, any planning and scheduling methodology can be functionally decomposed and mapped on the deficiency announcement and plan transformation generation principle, and second, the framework allows for an explicit declaration of planning strategies. We first investigate the flexibility of the extremely modular system design by instantiating the framework in a variety of system configurations including classical partial-order causal-link (POCL) planning, hierarchical task-network (HTN) planning, and classical scheduling.

As a key feature, the presented approach provides a formally integrated treatment of action and state abstraction, thus naturally combining causality-focused reasoning with hierarchical, procedure-oriented methods. While the use of procedural knowledge allows to rely on well-known, predefined solutions to planning problems, the non-hierarchical methods provide the flexibility to come up with non-standard solutions and to complete under-specified problem instances, respectively. The resulting technique of *hybrid planning* is capable of constructing a plan's causal and hierarchical structure across multiple levels of abstraction by using plan development options of both the POCL and HTN paradigms. We also present an *integrated planning and scheduling* system that is defined in our framework. For the first time, such a system is able to combine any ensemble of planning and scheduling technologies on the operational level and to address the respective deficiencies opportunistically. The accordingly unique representation of application domains incorporates temporal phenomena and resource manipulations not only for basic actions but on the abstract action level as well. This leads to the novel technique of *hierarchical scheduling*, in which the concept of abstraction is extended to resource representation and reasoning, for example resource aggregation and approximation.

Thanks to its well-defined functional composition, the framework yields a major improvement with respect to the capabilities of planning and scheduling strategies. The explicitly represented information about a plan's deficiencies and development prospects makes it possible to utilize a new quality of knowledge sources, including relationships between deficiencies, refinements, and components in the plan to which they refer. This leads to the novel class of *flexible strategies*, which decide upon problem characteristics and the current state of the plan generation process, respectively. The most prominent representatives are our HotSpot and HotZone strategies, which take into account the structural dependencies between problematic

elements in the plan when deciding upon their resolution. They are independent of both the application domain and the concrete framework instance. Therefore, they can be deployed for POCL planning as well as for integrated planning and scheduling, for example, and any other combination of methods the overall framework allows for. In addition, these strategy components are not only easily exchangeable, they can also be combined into sequences of decision procedures in which succeeding components fine-tune the choices of the preceding ones. We present the declarations of a comprehensive strategy repertoire, ranging from classical strategy components that implement well-known search principles from the literature to an assortment of flexible strategies.

Our formal framework is not only a method for specifying a variety of planning and scheduling functionality, it also enables the derivation of software architectures for fielding the corresponding systems. We show how the formal entities of the framework can be directly mapped onto software artefacts in a knowledge-based multiagent architecture, which optimally supports concurrency – by enabling parallel computations of plan deficiencies and refinement options – as well as the paradigm of distributed knowledge management. While the former addresses the practical issue of managing multiple computational resources the latter matches perfectly the idea of different modules representing different planning and scheduling aspects.

Our implementation of the framework resulted in a complex planning environment in which any planning and scheduling system can be easily compiled from a rich collection of functional components. By systematically alternating the system configuration and its parameters, it can also be used as a testbed for the evaluation of framework components, in particular planning strategies and refinement methods. This allowed for conducting a large-scale empirical study on dozens of strategy configurations, which is the first extensive experimental effort in the domain of hybrid planning. It concludes this thesis with four important results: First, we gain insights into the performance of members of our strategy portfolio on a set of benchmark problems. We thereby learned how to graduate performance measures and how to assess such test results. Second, we became familiar with the characteristics of the examined strategies, the experiment problems, and also of the benchmark domains. Third, our findings clearly support both the necessity and feasibility of systematic experimentation in order to identify suitable strategies for a given application domain. Last, but not least, our evaluation effort proves that our environment is an effective platform for orchestrating and operating component-based planning and scheduling systems, in terms of flexibility as well as in terms of efficiency.

Acknowledgements

This dissertation thesis is the incarnation of the greatest challenge in my academic life so far, and having finished it created a corresponding amount of pride and joy. But this book, although bearing only one name on the front cover, is the result of the help and support of many people. It is therefore my pleasure to thank everyone who has directly or indirectly influenced and contributed to this work.

First and foremost I thank my supervisor Prof. Susanne Biundo for encouraging me to carry out my thesis work on this particular topic, while at the same time giving me any freedom to pursue the scientific questions I was interested in. Her trust in me and constant support during many trips along side-tracks in my research was a great contribution. I also thank Prof. Friedrich von Henke for acting as referee of my thesis – this is a particularly nice "coincidence" because it was his lecture that originally has brought me to the field of Artificial Intelligence during my course of studies now some years ago. Furthermore, I thank Prof. Helmut Partsch and Prof. Michael Weber for joining the examination board.

I owe many thanks to my colleagues and friends at the Institute of Artificial Intelligence. I thank both my past and present fellow planners Hartmut Jungholt, Roland Holzer, and Julien Bidot for the fruitful discussions, great support, and enjoyable company during our joint struggle. Although it is hard to believe, there is more to life than just planning, and therefore I am also thankful to my colleagues Holger Pfeifer, Thorsten Liebig, and Olaf Noppens for both sharing their expertise in other areas of AI as well as for THE coffee breaks.

I am also very thankful to all students who have chosen me to advise their internships or diploma theses – all of them have been an invaluable source of motivation and inspiration. Their contributions substantially helped me in my research. I thank Stefan Kühnemann for building the first hybrid planner prototype and Olaf Noppens and Alexander Altmayer for their contributions to the initial system architecture; Markus Kuhnt and Dominik Maschke for implementing the current software design and setting up the first system configurations – their enthusiasm brought the first prototype to life that exactly worked in the way that I will describe in this thesis; Bastian Seegebarth for building an efficient reasoning module for state abstraction axioms; Steffen Balzer for taming the technology zoo in the knowledge-based architecture; Heiko Fröhlich, Manuel Jauernig, and Shawn Keen for programming the experimental toolkit with which I was able to conduct the empirical evaluation so painlessly; Alexander Pöltl, Guntmar Kirck, Tobias Dittmann, Martin Schels, Steffen Bucher, and Thomas Ruland for encoding domain models and problem definitions from sometimes very unspecific specifications; Andreas Lanz, Roland Ritter, and Bastian Seegebarth for building the domain model editing tool; Andreas Weigl, Andreas Lanz, Christian Bauer, and Michele Pinto for implementing and evaluating many of my novel planning strategies; Sascha Geßler for his generous commitment to a generic system development and for his help on designing an interactive planner.

Last but most definitely not least, preparing this dissertation has only been possible by the constant support and encouragement of my family. Whenever my own trust and strength faded, they picked me up.
Unfortunately, some of them are no longer amongst us and cannot see the result of this venture. I dedicate this thesis to their memory.

Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law.

(Douglas Hofstadter, Gödel, Escher, Bach: an Eternal Golden Braid, Chapter 5: "Recursive Structures and Processes", 1979)

Contents

1 An Introduction To This Thesis

I have a plan.
What sort of plan?
A plan that cannot possibly fail.

(Blake Edwards, A Shot in the Dark. *Metro-Goldwyn-Mayer Studios Inc., 1964)*

THE cited movie scene shows police Inspector Jacques Clouseau arguing with his superior at the very moment in which he is absolutely sure that he has understood the motives of the murderer, that he can anticipate his next moves, and that he knows exactly what has to be done in order to finally imprison the villain. Although he is wrong with his statement in every possible way, his situation may serve us as an illustration of what planning is all about.

First and foremost, planning is defined as the process of developing a detailed formulation of a program of action, intended to achieve an end when executed. Thus, plans are inseparable from the notion of an aim or a *goal*; the phrase "what are your plans?" not only refers to what exactly the asked person is going to do but mostly what the asked person is aiming at. The second essential concept is the assessment of the *initial situation*, which is providing an explicit description of what the world will be like when the plan is going to be executed. Inspector Clouseau is doing so by making aware to himself and to his collaborators what the known and deduced facts of the current case are. From this point of view planning is, conceptually, finding out what to do in order to reduce and eventually eliminate any difference between an initial situation and the goals (that means, arresting the murderer).

Planning consequently involves more than just a quick guess for surviving within a short time horizon and is generally regarded as the antithesis to reactive decision making; it is definitely connected with higher cognitive capabilities of human beings. All actions in a plan are included for a specific purpose; they are aiming at satisfying goals, possibly necessitating further actions to be performed before and therefore introducing new (sub-) goals. In the film scene, the Inspector's goals are to expose a criminal's identity so he is trying to anticipate which actions (more precisely: which actions in which timely order) are going to make the murderer make an unwary move. Obviously, this also involves examining the own actions for negative interactions: For instance, observing a suspect conflicts with questioning the person at the police station. All these performances are often superficially called "thinking about the actions' outcomes", but there is a more systematic process at work.

The respective entry in the Merriam-Webster Dictionary has an interesting note at the end, which says "PLAN: always implies mental formulation". This mental formulation is a key feature of the planning process: Planning, like it is shown for the police Inspector, involves temporal projection of actions and processes, their conditions and results, and finally a reasoning process about whether the acting agent will eventually achieve what has been aimed at. This implies the development of a *model* of the world in which predictions about the world's dynamics can be made, and it implies the implementation of an adequate reasoning process that is able to infer courses of actions from that model that reach goals, which in turn are represented in the model as well.

Regarding the reasoning process, Clouseau's example indicates two methodologies for building a plan. Both rely on a situation assessment; their ways of representing goals and deriving courses of actions from them

are however of a completely different kind. The first principle is to perform a regression on the goals, which means to identify what actions have to be performed in order to achieve the goals and then in turn to identify those actions that have to be performed in order to be able to perform the actions beforehand, etc., until the initial situation matches all execution requirements. There are many technical deviations possible, for instance to treat all goals equally versus to prefer goals that seem more important or to let chains of regression cause an overlap versus a linear treatment, but the basic process is essentially always the described one.

The nature of the second principle lies in the fact that human beings have certain experiences, rules of thumb, or written guidelines that give them the ability to formulate abstract plans of actions. Clouseau knows, for example, that he has "to observe the suspect" in order to get more information on the case. He does not perform a reasoning process like "in order to get the information where the suspect lives, I have to watch which house the suspect enters. In order to watch which house he enters, I have to be very close to the suspect when he enters a house. In order to be very close to the suspect in this situation, I have to ensure to follow him to the particular house from whatever place he went to before", and so on. What humans do in such cases is to start out with an abstract plan and then to think about possible refinements of the individual plan steps. The cited film shows many examples of how the Inspector thinks a professional observation has to be conducted, that means he proposes a number of alternative plan refinements. While most of the alternatives are similar undercover operations, the observation task could also be realized as an observation distributed among a number of police men, etc. As we will see in later sections, refining abstract plans in order to obtain executable courses of actions is not only a technical variation of the plan generation schema, but indeed a matter of principle and a fundamentally different approach to planning.

Having said this, it becomes apparent that although the two kinds of plan generation are working in a different way, in general, both are performed in an interleaved way. The typical human planner uses both mechanisms and representation principles in parallel.

Another aspect of this thesis's topic is the usage of time and the involved resources. For solving the case, it is not only important to know *what* to do but also *when* and *with what means* to do it. The first paragraphs assumed the temporal projection to be an abstract qualitative process, defining actions to be executed one "after" another. This is of course a strong simplification, because, for example, the suspect could leave the country at a specific point in time and has to be observed before that. Some actions have to be carried out during day, some others during night time, etc. The allocation of resources is also a crucial information that has to be taken into account, because the number of available policemen limits the number of leads that can be followed in parallel, other resources limit the usage of certain police laboratories for the examination of pieces of evidence, and so on.

Although humans tend to perform detailed temporal and resource reasoning *after* deciding on the course of action to pursue, the way *how* a plan can be implemented always feeds back into modifying the plan again. In particular, if tight deadlines are given and if resources are really scarce, temporal and resource reasoning are entwined with plan generation.

Leaving Inspector Clouseau and the anthropomorphic view on the topic, we can state that planning (including temporal reasoning and resource scheduling) is in general a very high-level cognitive capability that involves mathematical models of the application domain and logical reasoning techniques to derive courses of actions from problem specifications. The sheer amount of information processing that is required to obtain a solution in a tolerable period of time and that is both guaranteed to work properly and be a "high-quality" solution, all that makes planning a very demanding task for humans. Since planning is mission critical in numerous contexts, supportive mechanisms become necessary, ranging from practical paper-oriented methodologies to mechanized computer-based tools.

The vision that emerges from that situation is twofold: (1) Automated planning could make use of the full bandwidth of information technology that enables to overlook multiple plan alternatives simultaneously, to keep track of thousands of facts and mutual relationships between them, and finally to take into account safety-conditions, etc. (2) A second aspect would be to enhance applications with planning technology in order to make them more intelligent and flexible: Tools for decision-making support are able to propose solutions creatively instead of passively evaluating the user input, autonomous systems gain true autonomy for they work out their courses of actions by themselves, and the like.

This is where the science of *Artificial Intelligence* enters the stage.

The everyday success of human intelligence inspired many researchers in the field of computer science to mimic the cognitive processes and to built humanoid software systems of comparable, if not equal, performance. In the mid-1950s, the research area of Artificial Intelligence (AI) became apparent and was engaged in the planning topic more or less from the very first day on. Planning has been a central issue in AI ever since and it is consequently not amazing that planning has not only been pursued on the technical side in many conferences and workshops and countless scientific projects but also promoted politically. A large number of planning-related projects is sponsored by the DARPA agency of the United States Department of Defense, which consider planning as a valuable technology for use by the military. The European Commission established PLANET, the "Network of Excellence in AI Planning": it regarded the planning topic important enough to fund this coordinating framework for the interchange and social network building in the planning related scientific communities and industrial sectors.

For some reasons, the research efforts for building the described cognitive processes have been split into three mostly disjoint communities and pursued independently for many years. First, the generation of plans has been considered to be independent from the allocation of resources over time and thus the areas of *planning* and *scheduling* diverged. While the former stayed in the focus of AI researchers, the latter was dominantly pursued in the area of applied mathematics (Operations Research). The planning community finally regarded the principles of hierarchical plan refinement to be fundamentally different from the goal regression type of planning, considered them to be mutual exclusive, and split into two sub-communities. These groups perceived their fields as competing areas with one being labeled "domain-independent" planning, seeking for the holy grail of ultimate computational performance, and "domain-dependent" planning, being the application-oriented practitioners.
The following introductory sections will outline these three areas, provide some historical background, and present their main concepts and technical achievements.

However, after many years of progress in isolation, AI researchers realized that with their application areas becoming more and more complex, the tackled problems became more and more *mixed* semi-hierarchical planning and scheduling problems. The strict separation of the methods consequently begun to hamper developments, because irrespective of the individual success of the involved techniques, combining them was problematic already on the representational side and mostly impossible on the reasoning algorithm side. The research communities recollected that the difference between the developed planning and scheduling methods lay in optimized solution techniques and that a re-integration is highly desirable.

This is where this thesis comes in.

Although recent developments try to selectively bridge the gap between the three areas, we see that there is a need for a common, uniform formal basis in order to obtain a full and seamless integration. Consequently, the adjective *hybrid* has been put intentionally in an ambiguous position of this thesis' title, so the mixture may be composed of different planning aspects, but it emphasizes as well that the resulting system is an offspring of the two concepts planning and scheduling. Our ultimate goal is a theory that unifies all branches related to planning and scheduling by one representation and one family of reasoning algorithms. This formal basis will enable a new family of search strategies that take advantage of all planning methods that are covered by the hybrid and we will show how the theoretical framework can be directly translated into an effective software platform for productive use and scientific experimentation.

The following sections review both the state-of-the-art in the areas of planning and scheduling, presenting approaches that may contribute to our task from the technical point of view, and approaches that address the described issues in an alternative way. We will not only see the diversity of the field but also its fragmentation and we will motivate the necessity to overcome the differences. This thesis wants to contribute to bridging the gap in several dimensions:

- It wants to provide a reliable and theoretically well-founded basis for planning and scheduling,
- foster research by creating a planning and scheduling technology that can be deployed in a plug-and-play fashion, and

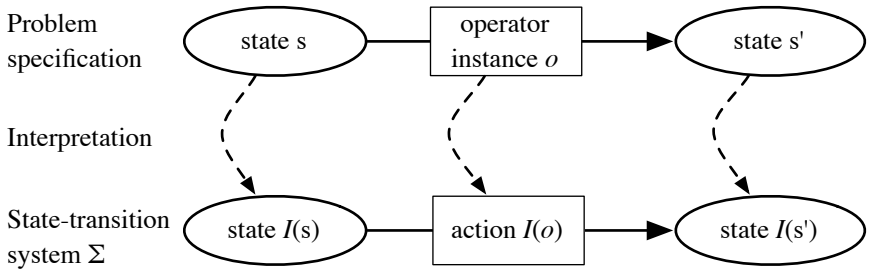


Figure 1.1: Interpretation of a planning domain as a state-transition system [118].

- make new application areas accessible by developing a software conception that allows for realizing planning functionality based on the above concepts.

These visions and aims will be made more concrete after the technical introduction sections.

1.1 AI Planning in a Nutshell

In this section, we want to give an overview over the relevant developments in the area of AI planning. The presentation is organized along two dimensions: on the one hand we want to point out technical relationships, on the other hand to show their historical progression. We analyze the mutual dependencies between the choice of representational features, in particular the *plan* data structure, and the planning algorithms. It is these dependencies, which caused not only the field's rich (positive) diversity and technical achievements but also its (negative) scattering and incompatibilities. At the same time, this section has a focus on the common representation concepts and underlying algorithmic principles, which motivate our theoretical foundations and ultimately our integration efforts.

Before we are going into the details of the different planning methods, let us have a look at their commonalities. Practically all AI planning approaches view the world as a state transition system and consequently the task of planning is to choose and organize actions for changing the state of that system (cf. the conceptual model for planning in [118]). The description of the state features and of the actions – in particular, which state changes they induce – is given by a *planning domain model*, which typically consists of a more or less formal language and of *operator* definitions. We furthermore assume that an interpretation I is given for the language elements and operators, that maps the planner's domain model entities to the state-transition model of the execution environment, thereby providing a referential frame for any course of action that is expressed in terms of the planning model. Fig. 1.1 depicts the correlation between the real-world application, represented by a state-transition system Σ, and the planning domain model, represented by a problem specification. A *plan* in this context is basically a sequence of instances of the operator definitions, but there are advanced, more sophisticated representations.

A *planning problem* is typically specified by a domain model, an initial state, and a goal state description. A plan is considered to be a *solution* to a problem if it is constructed over the given domain model, executable in the initial state, and its execution results in a state that fulfills the goal state description. Again, there will be shown various examples for enhanced problem specifications and solution criteria, but this is the most general case.

Implementations of this concept have been employed successfully in many application areas. Although there may be some researchers left who still focus on solving the re-arrangement of tons of toy bricks or the defusing of thousands of explosives in millions of lavatories, the technology has become sufficiently mature to be applied in areas that are relevant for the economy, technical innovation, and daily life of a modern society. Let us give some examples:

Crisis management is one of the most often addressed application areas, because it is the prototypical problem of planning: Find a course of action for managing the crisis, take into account numerous events, staff, and material, and do so under time pressure. For instance, planning scenarios for military evacuation operations are a subject of automated planning [261, 262] and interactive decision support [89, 195]. In the last years, support for addressing environmental crisis has gained more and more attention: Examples are emergency response to marine oil spills [2], disaster relief management in floodings [26], and forest fire management [9, 70].

Closely related to crisis management, a typical planning application in the commercial sector is transportation and logistics [90,259]. The technology is also involved in manufacturing process planning, for instance, the production of microwave modules [242], the generation of plant control programs [43], or the platform deployment in the space sector: the assembly, integration, and test of spacecraft [1,87].

Another important topic is the control of autonomous systems. The most prestigious application is certainly the New Millennium Remote Agent of NASA, where planning technology was a core component in realizing the control software for an autonomous deep space probe [142, 194]. As autonomy became increasingly relevant for commercial space missions, several planning application emerged in this sector, for example communication and antenna operation planning for supporting unpiloted interplanetary spacecraft missions [58]. There are also classical industries with the need for a high degree of autonomy, for example dock-worker robots [3], and an endless list of robotics research programs that use plan-based controllers for strategic decision-making.

With the rise of the service metaphor for web-based content providers, planning technology becomes a key component for automated composition of web-services due to increasingly complex service models and user requirements [263]. The problem of data retrieval on restricted and/or commercial knowledge sources has also been addressed by the planning of appropriate queries and integrating their results [7].

As a last application area, computer games can benefit from planning technology. Although card games are a classical problem for game-tree search techniques, planning methods have been successfully applied to the game of bridge and even provide the current world-champion in software bridge [243]. On the borderline between in-game AI and intelligent software development tools we also find planning systems that generate, for example, state automata for computer-controlled characters [210].

As a concretization of the above general conceptual view, the following sections will present the field of AI planning as four segments: First, we are going to describe classical planning techniques in order to give the background for many representational issues in planning. Second and thirdly, we introduce partial-order planning and hierarchical planning, which are the two complementary main components in our hybrid planning framework. The fourth and final segment that we investigate is the state of the art in hybrid planning. After that, we are taking a look at resource reasoning, that is to say, at scheduling and resource planning approaches. We conclude our review with a brief glance at other planning techniques that are beyond the scope of this thesis. For more information on the different trends, in particular those that we omit here, we refer the reader to textbooks and review articles that are commonly regarded to cover the field in a representative way [4, 118, 198, 281].

1.1.1 Classical Planning

The most prominent representative of AI planning systems is certainly the STRIPS system [92], and it is commonly regarded as the first specific planning system. The "STanford Research Institute Problem Solver" was built to control a mobile robot on the strategic level in order to enable it to carry out transportation tasks in an in-door environment. The STRIPS planner performs a *means-end-analysis*, that means, it performs a backward-search in the state space: Starting from a goal state specification, the algorithm chooses an unmet sub-goal (the end) for which it tries to identify the necessary action that satisfies it (the means). Choosing an action in turn induces some new goals – its executability criteria – and the algorithm is called recursively on these new goals. This backward search terminates on those actions that are directly executable in the initial state and the algorithm performs a state-progression phase: It updates the initial state by applying the effects of the executable action to it. After that, the updated initial state is passed back to the previous recursive

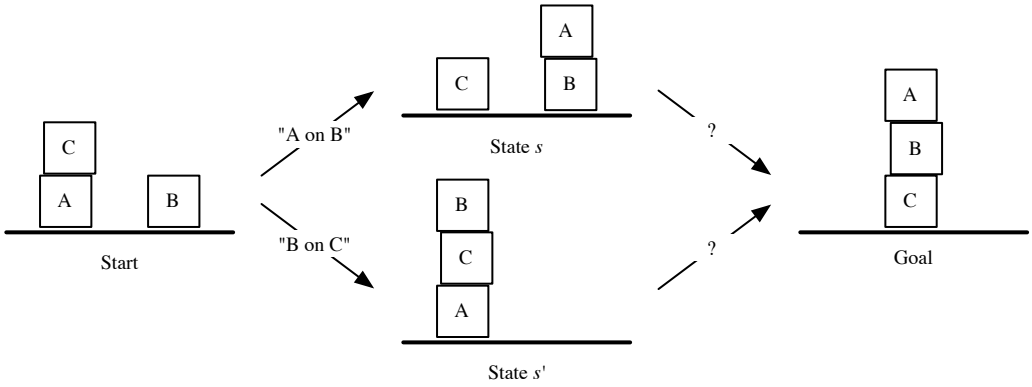


Figure 1.2: Sussman's Anomaly.

call, in which the procedure continues to search for the means to establish the current ends. The described mixture of backward-search and forward-progression is also called "island-search", that means, the planner identifies islands states for which it tries to find suitable plans to connect them. In this way, a plan is more or less constructed backwards with respect to its later execution.

An in-depth description of this well-known algorithm is beyond the scope of this presentation (cf. appropriate textbooks, for example [118, 209, 223]). In our context, we believe it is sufficient to note that STRIPS' exclusive focus on the newly introduced goals reduces the branching factor of the backward-search, however makes the algorithm incomplete in the general case. The STRIPS planning procedure namely relies on the assumption that sub-goals are independent from each other and can be pursued in an arbitrary order. The outcome of the STRIPS algorithm and any intermediate results are totally ordered sequences of actions, hence STRIPS and its relatives are also called *linear* or *total-order* planners. A great advantage of this particular modus operandi is that it becomes possible to infer the accumulated consequences and pre-requisites of actions in a computational cheap manner and for very expressive action representations.

The above mentioned incompleteness of STRIPS became visible for the famous "Sussman's Anomaly". In experiments with the HACKER planning system developed by Gerald Sussman [253], Alan Brown at MIT noticed that the planner had difficulties with a toy problem as it is depicted in Fig. 1.2. Although it appeared to be ridiculously simple, it was challenging for STRIPS-like implementations due to the interacting sub-goals: If the planner chooses solving block A to be on block B first, it consequently produces a sub-plan that builds a two-block tower with A on B. Depending on the concrete algorithm, the system is now stuck, because the complete goal cannot be achieved without undoing the reached sub-goal, or it continues with constructing a plan fragment that undoes the previously generated one. In the latter case, the obtained solution is suboptimal. Unfortunately, choosing the other sub-goal first leads to an analogous dilemma and therefore solving the Sussman Anomaly in an elegant way became a driving research task.

Since the dependency of goals was obviously not only the cause of the toy problem being hard but also proved the corresponding STRIPS assumption to be fairly unrealistic, many solutions to this phenomenon have been proposed. First patches addressed the analysis of goal interactions, for example, INTERPLAN [254]. But soon the primary weakness of the STRIPS-like approaches was identified to be the too strict commitment of the procedures to plan step orderings and the era of non-linear or partial-order planning began. We note that, ironically, the fastest planning systems today are in fact linear planners that conduct a forward state-progression. Sophisticated search techniques have been developed that use solutions for relaxed problem specifications as a heuristic function [34, 136][1] or allow to define domain-specific control knowledge [12]. Since these approaches are algorithms solely tailored for speed and not very well formally based, they are not in our focus.

[1] Note the subtle difference to introducing a domain-specific search heuristic "manually".

While the STRIPS algorithm is nowadays history and numerous successors have surpassed it in terms of computational performance and manageable application characteristics, its formalism prevailed. Although the initial STRIPS formalism's semantics had a number of weaknesses that needed clarification [169], the representation of actions and states that STRIPS introduced are still used by the community. Although our approach does not use STRIPS but instead the formally more advanced ADL action language, STRIPS is and has been the most dominating representation formalism. For example, the current standard language for the International Planning Competition, PDDL [99], is using an extension[2] of STRIPS. We are therefore going to discuss some of its aspects in more detail.

During their research on solving the robotic control problem, Fikes and Nilsson particularly considered work in the area of automated theorem proving. At that time, Green had presented an approach to domain-independent planning that is known as the *situation calculus* [124]: The world's state is encoded in first-order predicate logic such that atoms that represent fluent properties are augmented by a situation parameter. In this formalism, operators are defined as non-logical axioms that describe how the truth value of atoms changes in the situations that result from the operator application. Given an axiomatic description of the initial state, the goal, and the operators, a resolution theorem-prover is then used to produce a constructive proof that an adequate sequence of state changes exists, and from that proof the corresponding plan is extracted. Fikes and Nilsson liked the idea of using a general-purpose problem solver, however, they were also aware of the problems the situation calculus has to face: (1) Using a resolution engine for generating plans mixes two kinds of search, namely search in the space of world models (that means, constructing candidate plans) and search for a proof that the plan satisfies the goal. Finding a reasonably well performing proof strategy that is able to handle both search aspects was regarded to be extremely difficult if not impossible. (2) The formal specification of operators has to include so-called *frame axioms* that describe which facts are *not* changed by the action; reasoning about these operator invariants became known as the frame problem. Since any operator typically affects only a small subset of the world's facts but nonetheless all corresponding frame axioms have to be proven in the proof of plan existence, it is solving the frame problem that dominates the computational effort of the procedure.

The solution of Fikes and Nilsson directly addressed the frame problem: the application of operators is removed from the formal deductive system and first-order theorem proving methods are used only within a given world model to answer questions about which operators are applicable, and whether or not the goal has been satisfied. To this end, states are represented by well-formed first-order predicate logic formulae in clause form, and the operator specification is given a specific structure: the name and parameters of the operator, the precondition, and the effects. The precondition is an arbitrary formula over the operator parameters. Determining whether an action is executable is done via a theorem prover that verifies whether the precondition formula holds in that state. The operator's effects are given as so-called *add* and *delete* lists: lists of clauses that are to be added to a state and clauses that are to be deleted from it[3]. All clauses that are not mentioned explicitly in the delete list are "copied" into the successor state – a simple solution to the frame problem. It is worth noting that states are interpreted under the *closed world assumption*, that means, the state description is assumed to be complete and consequently all atoms that are not present as positive literals in the state's clause set are interpreted as false. However, this expressive planning language was not given a well-defined semantics (see discussion in [169]) and was consequently reduced to action descriptions that would contain only atoms [209].

1.1.2 Partial-Order Planning

Partial-order planning, also called non-linear planning, started out as an alternative to total-order planning approaches like STRIPS (for an overview, see [279]). As we have explained above, this research was mainly motivated by finding solutions for problems with interacting sub-goals like, for example, the Sussman Anomaly. Sacerdoti was the first to formulate, that although plans are executed one step at a time, the "plans themselves are not constrained by limitations of linearity" [225]. The NOAH system (Nets Of Action Hierarchies) introduced a new data structure to represent plans, the so-called "procedural net", that consists

[2] The extension is purely syntactic. Although syntactical elements of ADL and several new developments have been added, its semantics are still that of STRIPS.

[3] In this way, STRIPS is manipulating a state formula where the before mentioned ADL is changing the interpretation. We will revisit this particular issue later.

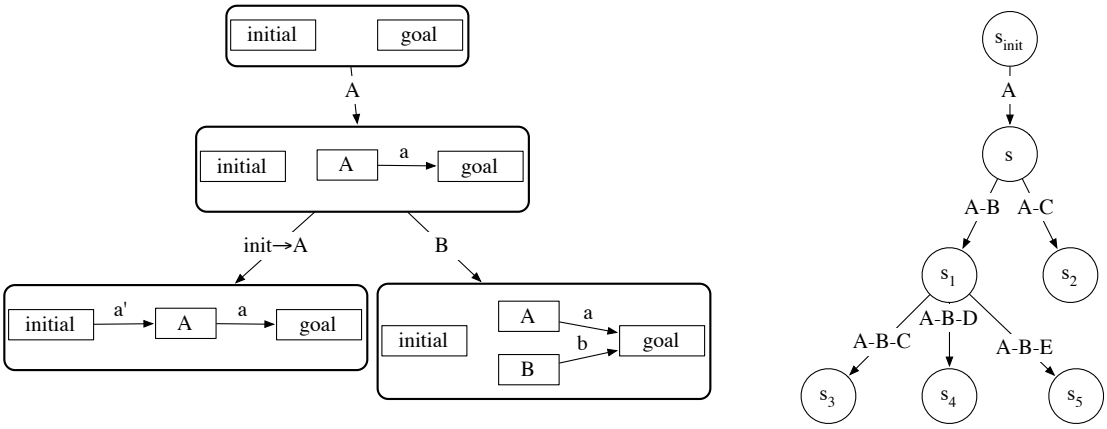


Figure 1.3: Search in the plan space (left) compared to search in the state space (right). In the plan space, the plan data structure is manipulated by adding plan steps, constraints, etc. Searching in the state space is done by extending the plan (here: forward search) and computing the corresponding result state (here: the state after executing the sequence).

of actions and a partial ordering relation on them with respect to time. Ordering steps between actions are only introduced when they become necessary due to the goal interaction analysis.

But non-linearity is not the only relevant contribution of NOAH: it also introduced the notion of *search in the space of plans*. Fig. 1.3 illustrates the approach and compares it to search in the space of states. The algorithm basically checks a current plan against three so-called *critics*, which induce a reordering of plan steps if a conflict is detected, and the like. Although the technique is quite limited, NOAH can be considered to be the ancestor of all modern planning systems that are based on search in the space of plans. We note that our proposed framework will employ a related technique, however formally grounded and considerably more flexible.

The latest noteworthy approach that is in the direct line of succession of STRIPS is the NONLIN system by Tate [255]. It completed the step towards plan-space planning as initiated by NOAH: It introduced the nowadays commonly used *partial plan* data structure – that means, plan steps, ordering constraints on the plan steps, and parameter bindings – and it applied a general backtracking schema over the plan construction operations. Many previously fixed decision points during plan generation are now considered to be (non-deterministic) choice points that (a) represent alternative ways of constructing a solution and (b) are subject to re-consideration if a particular choice turns out to lead to a dead end. NONLIN thereby also finally established the notion of least commitment in AI planning, since now all previously discovered anomalies could be solved. NONLIN was also the first system to record the commitment on causal dependencies between plan steps via *causal links*: a triple $\langle a_p, Q, a_c \rangle$ represents the information that action a_p is intended to produce proposition Q for the consumer action a_c. Causal links were in particular invented to compensate for the inability to compute complete state descriptions for partially ordered plan steps – plan-space based approaches do not keep explicit states. Causal links work in two ways: First, they document whether the precondition of a plan step is completely supported. Unsupported propositions can either be established by adding a causal link from an appropriate plan step or by adding a new plan step that carries the desired effect. Second, once the support is established, it can be used to identify negative action interactions efficiently: A causal link is *threatened* if there is a plan step that can be consistently ordered between the producer and the consumer action and that carries the the respective proposition in its negative effects.

Planners that employ the causal link technique are consequently called *Partial-Order Causal-Link* (POCL) planners.

An alternative to recording each causal support explicitly is the so-called *modal truth criterion*, a formalized criterion for determining whether a partial-order plan achieves a given precondition at a given step [54]. The criterion can not only be used for verifying plans but also for generating them, and Chapman demonstrated that with his planner TWEAK. The system uses constraints as its approach to problem solving. The basic idea is to "define" a plan by incrementally specifying partial constraints it must fit. The search space is

pruned as constraints are added, until all remaining alternatives satisfy the constraints. This idea is closely related to the technique called *refinement-based planning* in which the plan data structure is interpreted as a specification for the class of all plans that contain the same components.

The modal truth criterion states that for any plan step a_c a condition Q holds (more precisely: Q holds in the state in which a_c is to be executed) if and only if Q has been introduced by a step a_p that is executed before a_c and for any step a_x either a_c is executed before a_x or a_x does not undo Q. In addition, there may also exist another step a_w that re-establishes Q. Regarding the naming, a_x is referred to as the "Clobberer" and a_w to as the "White Knight".

The planning procedure consists of repeatedly choosing a goal and then making a plan to achieve it. It uses the modal truth criterion to do this; the criterion shows all possible ways a proposition could be made to necessarily evaluate to "true"; the procedure chooses one of them and modifies the plan accordingly. Using the modal truth criterion makes the algorithm sound and complete, however at the price of a very high computational complexity. This in fact caused Chapman and many other researchers to conjecture that plan-space planning does not scale for more expressive action languages (and TWEAK employed a simple one).

However, from today's perspective, the causal link technique solved the issue, basically because Chapman's conjecture was based on the misbelief that planning requires a necessary *and* sufficient truth condition: As it turned out, a sufficient truth condition is completely adequate. Essentially, POCL-planners only care about ensuring that a goal condition is true for a specific plan step but they do not reason about the truth values of arbitrary conditions in arbitrary states. The downside is that incrementally imposing constraints on the plan may lead to dead ends and the system has to backtrack: POCL algorithms "push the complexity of evaluating the modal truth criterion into the size of the search space" [279]. It has to be noted that search space size is a relative measure, since large parts of it become irrelevant once cleverly introduced constraints are violated. For a discussion of these aspects see [150].

The "Systematic Non-Linear Planner" (SNLP) [177] is an improved formalization of Tate's NONLIN planner[4]. In its conceptual clarity, it is one of the two archetypical algorithms of today for any POCL system that develops plans in a least commitment fashion: causal links are used to record necessary causal support, plan steps are partially ordered when actions need to precede each other, and parameter bindings are introduced on-demand in order to guarantee causal support or to resolve threats. Our framework's implementation of partial-order planning is also inspired by the SNLP planner (cf. Sec. 3.2.2).

A particularity of the SNLP approach – an intentional demarcation with respect to NONLIN and its contemporaries – is its interpretation of causal threats. Practically all other partial-order planning approaches consider a plan step a_t to be a threat to a causal link $\langle a_p, Q, a_c \rangle$ only if a_t deletes Q (and is not ordered before a_p or after a_c). SNLP employs a stronger notion of threat: Plan step a_t is a threat even if it adds Q. While this notion of a conflict appears counter-intuitive, it provides the algorithm with an interesting property: systematicity, that means, no plan is encountered twice in the plan-space. It is however widely believed that this kind of plan-space reduction is not necessarily relevant for practical implementations. We will discuss these issues in detail in the chapter that is dedicated to the formal framework, including performance considerations (Sec. 2.8.5, but also cf. [145]).

The second "standard" partial-order planning algorithm is that of the UCPOP system [214]. Penberthy and Weld wanted to build a planner that was able to handle a more expressive language than the restrictive STRIPS representation and to do so without reverting to linear plans. Concerning the domain model language, UCPOP uses ADL, a language that has been invented by Pednault in the late 80's [211]. For a detailed discussion on the ADL representation, we refer the reader to the formal framework section (cf. Sec. 2.8.1), but for now it is sufficient to note that the key features of ADL covered by UCPOP include universally quantified and conditional effects; the first two letters in UCPOP refer to this fact. The universal quantification allows effect statements to refer to the universal base of a world state, for example, when a container object is moved, *everything* inside changes its position as well. Conditional effects are intended to provide more compact operator definitions: if a given effect condition holds, additional effects are established. An equivalent way

[4] In many surveys, TWEAK is mentioned as SNLP's predecessor, but McAllester and Rosenblitt explicitly refer in their initial SNLP paper to the NONLIN procedure.

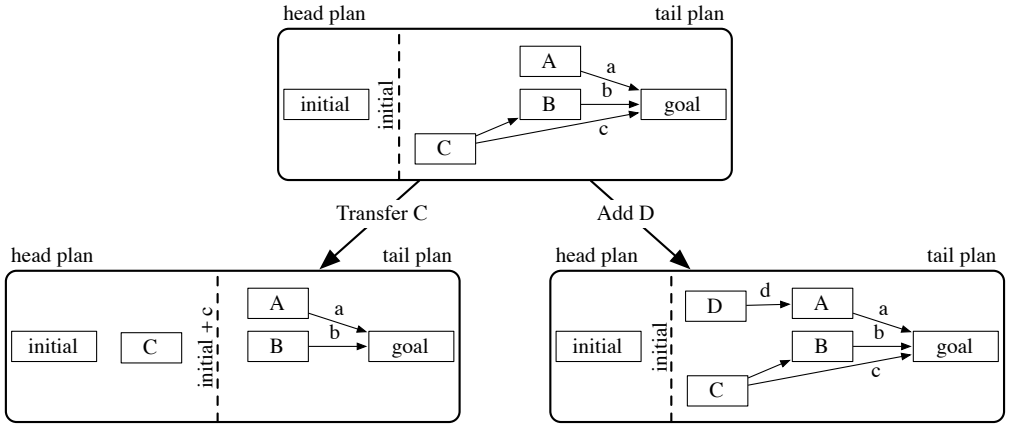


Figure 1.4: The head- and tail-plan as it is used in the PRODIGY system.

of modelling is to define a second operator with the effect condition as additional precondition and the complete effects (including the additional ones).

Besides these technical features UCPOP was the first non-linear planner of that expressivity, and moreover it was the first non-linear planner for which soundness and completeness were formally proven.

Since UCPOP and SNLP were very well documented and conceptually very clear, they inspired many planning system designers to build extensions. Successful examples for such an undertaking are the VHPOP system [297] and REPOP [207]. Based on UCPOP, these planners focus on implementing efficient planning strategies and evaluating search heuristics. The VHPOP approach in addition incorporates temporally extended actions (*durative actions* in PDDL [99]) and participated quite successfully in the International Planning Competition.

It is worth noting that the issue whether it is an advantage to use a total order on the plan steps rather than a partial one has never settled. The differences between the two techniques, with a focus on computational efficiency and the impact of respective planning strategies, is extensively discussed in [191, 192] and [16], among others.

An interesting linear/non-linear hybrid system is realized in PRODIGY [93]. It combines these contradictory principles by dividing the plan in two parts that are treated with different methods: In order to achieve the goals, the usual non-linear plan is built in a least commitment fashion, the so-called *tail-plan*. The system can decide at any time to "commit for execution" those leading steps of the tail-plan that are causally completely supported by the current initial state by queueing them into a totally ordered sequence of actions, the *head-plan*. The linear head-plan is processed by an execution simulator that updates the state at the end of the head-plan accordingly – this is at the same time an updated initial state for the tail-plan. Fig. 1.4 illustrates this procedure. The committing of a plan step is thereby an additional plan refinement operation.

The linear head-plan enables PRODIGY to support a more expressive planning language than STRIPS (disjunctive preconditions, etc.) in an efficient manner, while the tail-plan handling provides the completeness property. Apart from its technical particularities, PRODIGY became famous for its integration with various machine learning techniques that provided it with efficient planning strategies.

A complete family of planning algorithms and systems has been initiated by GRAPHPLAN [28]. Blum and Furst divided plan generation in two phases: In the first phase, a *planning graph* (Fig. 1.5) is constructed that consists of alternating layers of state facts (propositional atom) and ground actions. All actions that are applicable due to their preconditions' facts occuring in the current state layer are placed into the subsequent action layer, and all facts that are produced by the effects of the actions in an action layer will be present in the following state layer, and so forth. At the same time, *mutually exclusive* facts (for example, positive and negative atom occurrences) are propagated into the next action layer, since actions are mutually exclusive if their preconditions are. In turn, facts that are generated from mutually exclusive actions become mutually

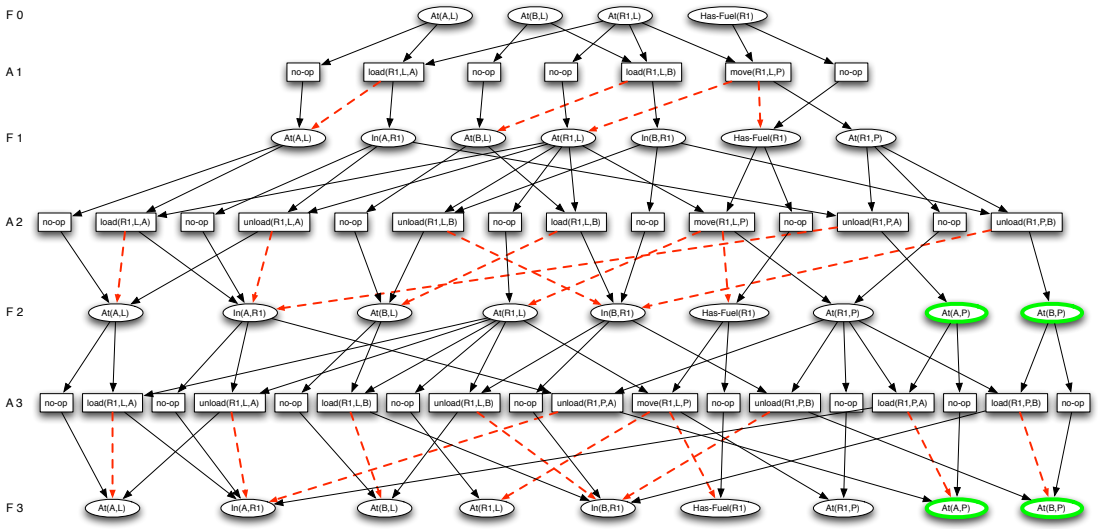


Figure 1.5: An example for the development of a GRAPHPLAN planning graph. State layer F0 denotes the initial state. We assume that according to the deployed domain model, the planning graph reaches non-mutex goal facts (in green ellipses) in the fourth state layer F3. Plan extraction is performed in a bottom up manner: the plan ends with actions from the preceding action layers that contributed to the goal facts, which activates these actions' precondition facts, and so on.

exclusive themselves, and so on. In addition, actions with contradicting preconditions and effects (cf. causal threats) are also marked mutually exclusive. The planning graph is generated in this way, until all facts of the defined goal state are present and un-marked in a fact layer. This graph represents a disjunctive set of maximum parallel plans and serves as a heuristic for the subsequent phase: the plan extraction. Beginning from the goal facts, the producing actions are determined, the facts that support these actions, their producing actions, etc. The complexity lies in this second phase, because it is a combinatorial problem to choose the appropriate path back to the initial fact layer (it is in general ambiguous). Note that a maximum parallel plan does not necessarily exist and planning graph generation has to be continued. For a more detailed treatment of the algorithms and related approaches see [280].

Much has been conjectured about where its efficiency comes from and under which conditions the system works fast or performs badly. We believe that GRAPHPLAN is a bad example for interleaving search and plan-generation, because apparently it makes the algorithm difficult to analyze, to understand, and to extend. Great efforts have been made to make GRAPHPLAN a temporal planner, to include resource reasoning, uncertainty, and the like. But to our knowledge, due to the fact that these extension mostly stem on re-interpreting the planning graph structure, they are conceptually mutually incompatible or at least impair GRAPHPLAN's efficiency.

1.1.3 Hierarchical Planning

The primary problem of the planning techniques that have been discussed so far is that of scalability: The complexity of planning problems is typically located at the borderline to an exponential effort (at best) and may escalate with any additional representational feature. Furthermore, the more actions and state properties are defined, the more difficult becomes plan generation, and domain model maintenance in general. These observations motivated the use of abstraction mechanisms in planning, which can be divided into two categories: approaches that use abstraction for the representation and reasoning about states, and approaches that impose abstraction hierarchies on the actions (cf. [292] and [6, Chap. 4]).

Planning by State Abstraction

The principle of state abstraction has been invented very early as one form of giving more structure to planning goals, it has therefore many facets and technical approaches. The probably first approach was realized in the ABSTRIPS system [224], that is still discussed today and, although it performs *linear* hierarchical planning, has influenced many modern planners. Classical state abstraction works by focusing on certain sets of preconditions and effects, thereby defining so-called *criticality levels* for each of which the system plans in the above described STRIPS-like manner. Starting from the lowest criticality level, when a plan for one level has been obtained, the system moves on to the next level and tries to fill the revealed gaps in the plan. If the plan cannot be completed into a solution, ABSTRIPS backtracks over the last criticality level. If the criticality is carefully assigned, a tremendous increase of performance is gained, if not, search is of course completely lost.

Since criticality values have such a big influence on the system behaviour on the one side, but are not always intuitively (and successfully) identifiably, the ALPINE approach proposes to automatically infer these levels from the operator definitions [153]. In doing so it builds abstraction hierarchies that comply with the *ordered monotonicity* property, that means, the detailed action levels do not interfere with the conditions established on the more abstract levels. Similar work in the context of non-linear planning has been done by Yang in the ABTWEAK planner [294]. It uses effect abstraction by distinguishing primary and side effects, and only allows reductions that do not introduce new threads to established preconditions, which is called *monotonic protection*. In any other aspect, the system works exactly like its ancestor TWEAK.

The obvious idea behind these systems is that abstraction might speed up the planning process, even if it is computationally hard to find a good abstraction hierarchy. Another appealing factor stems from state abstraction not working at the control level; as a consequence, it can be easily combined with other search techniques and heuristics. One of the drawbacks, for example, is noted by Giunchiglia in [121]: We can construct example cases with poor performance of abstraction planners, in a way that higher abstraction spaces contain no information about what caused backtracking at less abstract levels.

The underlying problem of the presented state abstraction methods is apparently a lack of semantics, because they are defined in terms of the search algorithm and are not reflected in the modelled domain as such. In general, the lowest criticality is typically assigned to rigid symbols, that means, to the static facts of the world. This makes perfectly sense, but higher levels do not express a hierarchy on "rigidness", they are rather classified into symbols of primary effects versus side effects. We believe that it is not very rational to assume that this classification can be constructed meaningfully and consistently in general. A reduced state description *can* be seen as an abstraction, but without further semantics, these reduction schemata do not necessarily build a hierarchy in the state space in a reasonable manner.

Planning by Action Abstraction

Action abstraction starts with a very simple idea: Why tediously synthesizing action sequences for the same kind of goals over and over again, when knowledge about *how* to achieve a particular goal is available? The basic approach is consequently to define plan fragments for specific *tasks* (abstract actions) in the domain model that are intended to *implement* the task in a plan. Instead of finding a course of action that produces specified state properties, the problem definition in this paradigm consists of an initial state description and an *initial plan*. This plan contains a number of abstract tasks that are to be performed in the given initial state, and the corresponding solution is a completely implemented plan that is obtained from the initial plan and that is executable in the initial state. The implementing plan fragments are usually lifted or parametrized according to the parameters of the task, which makes the domain models more flexible. Since the plan fragments typically contain further abstract actions, substitution has to be performed recursively. For that reason, this kind of hierarchical planners generally works on the plan-space, where causal interactions can be finally sorted out by POCL techniques. The substitution step is called *expansion* or *decomposition* of the abstract task and it induces a corresponding hierarchy on the actions of the domain model such that the actions in the decomposition are considered less abstract than the substituted task. In general, there is

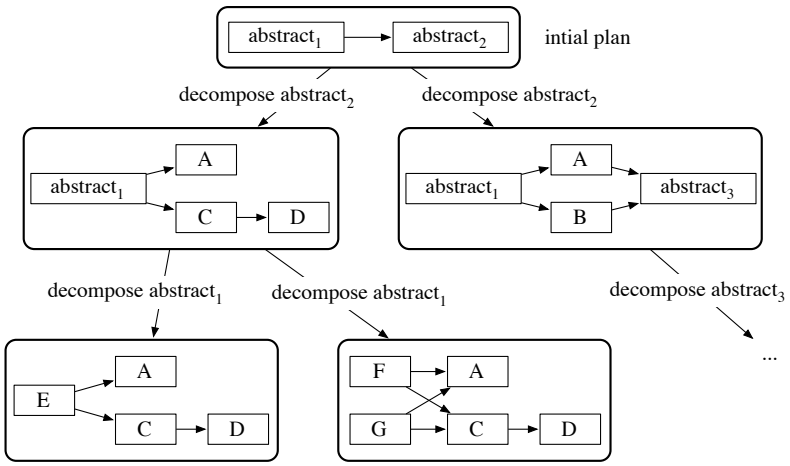


Figure 1.6: Planning with action abstraction: decomposition of abstract tasks.

more than one expansion provided for a task, that is to say, – in terms of plan-space planning – alternative decomposition refinements are available. Fig. 1.6 illustrates plan generation using action abstraction.

Over the years, planning via action abstraction has been subsumed under the label *Hierarchical Task Network* (HTN) panning. The approach was so successful that all but one of the planning applications that we listed at the beginning of this section (pp. 4) have in fact been realized with HTN planning systems. In the HTN literature, the domain model includes so-called *methods*, which relate abstract tasks with appropriate decomposition task networks and necessary parameter bindings, etc. (these data structures correspond to partial-order plans). The first systems that implemented action abstraction, including the above discussed NOAH [225] and NONLIN [255], were mainly focused on a procedural way of plan generation that allowed the planning system to introduce large sub-plans within one single decision. We believe that this view on HTN methods as rules of a grammar for intended solutions, that this relatively small amount of information that is carried by the decomposition schemata per se is the reason why, for a long time, no clear semantics was given to the decomposition process. Abstract action schemata do not bear preconditions or effects, because the intended procedure appears to successively expand all abstract tasks first, and then to deal with causal interactions, executability, etc.,

The O-PLAN system of the Artificial Intelligence Applications Institute of the University of Edinburgh [65, 261] is a direct successor of NONLIN. It is not only one of the most famous HTN planners, but also one of the most successful and industrially exploited implementation of AI planning technology so far [259, 260, 262]. It is characterized by many application-oriented features; for example, the software framework has a strong focus on inter-operability of the platform and the representation language is equipped with a number of application-motivated domain specification features like typed conditions.

Abstract tasks in O-PLAN do not carry preconditions and effects [65]. In order to compensate for this, the system relates conditions of primitive operators over different abstraction levels in the plan generation process by introducing *condition types* in the abstract expansion schemata [258]. These are constraints, similar to causal links, that annotate how conditions for the tasks in the expansion are intended to be achieved. This information has two dimensions: (a) whether the condition is a product of the effect of some task that is inside or outside the current expansion network and (b) whether the condition is introduced at the current plan generation level, above, or below. Unfortunately, techniques like condition types are not very well semantically grounded; they require the domain encoder to structure the task hierarchy very carefully as its pruning affects the system's search space structure rather than the structure of the plan space.

A similar approach to HTN planning has been realized in SIPE [281] and SIPE-2 [282], the SRI-bred brother of O-PLAN. The development was driven by the intention to realize a "practical" system in the context

of command and control scenarios (the system's acronym stands for "System for Interactive Planning and Execution"). Its applications range from military operations [284] to emergency response management in cases of marine oil spills [2]. Like O-PLAN, the SIPE system also employs a number of special-purpose domain model constructs that are not backed by an adequate semantic concept (goal-ordering heuristics inside operator definitions, and the like).

One of the latest directions in the HTN paradigm is represented by the "Simple Hierarchical Ordered Planner" SHOP [200]. This approach proposes an *ordered task decomposition* mechanism that uses if-then-else cascades in its method selection. The main idea is to construct the plan according to the order in which its operators are later executed, that means, SHOP is basically a state-based forward-chaining search algorithm. In order to do so, the SHOP decomposition procedure chooses the first abstract task that occurs according to the step ordering. Concerning the task expansion, the appropriate method is queried and the first decomposition for which the respective if-statement evaluates to "true" is selected for expanding the task. It has to be noted that all execution requirements for operators are intended to be modelled via the method application statements and primitive operators consequently do not have a precondition. By working on a linear ordering on the plan steps (see the discussion of linear planning above) the system is able to compute complete state descriptions, starting from the initial state. SHOP can therefore employ a highly expressive modelling language and even supports the complete Lisp functionality in its method-queries, including arbitrary numeric calculations.

The ordered task decomposition however requires a specific way of modelling: where linear planners synthesize action sequences in the described manner, an HTN system has analogously to be provided with linear-ordered expansion networks in the domain model. Although this requirement is not too restrictive in many cases – there exist numerous examples in which task networks are in fact defined linear – it occurs as a major restriction in practically every realistic application domain. The developers met the criticism regarding their linearity assumption with a modified system, called M-SHOP [201], which can handle planning problems with parallel goal tasks in the initial task network. There may exist realistic domains that meet this partial linearity property, but many, for instance crisis management, do not because task execution itself is highly distributed and the execution order for most tasks is not known in advance. The latest development in this series is SHOP2 [199, 202] that allows non-linearity for both problem specifications and expansion networks. Over the versions, the authors have re-introduced the usual non-determinism that is central to non-linear planning: SHOP2 "guesses" the missing ordering relations and from there, it explores the state space in the SHOP manner. The coverage of failed options in the non-determinism is addressed with a standard backtracking method.

A considerable problem with all the HTN planners we have presented so far is that of incorporating knowledge that belongs to the search-level into the model specification. Let us briefly discuss the main arguments, we will expatiate on some of the issues in a later section dedicated to the design of these classical strategies (Sec. 4.2.2).

As said before, one motivation of employing HTN techniques is to obtain a considerably better system performance. And indeed, if the HTN system is guided adequately, it can produce very large plans (that means, plans containing many plan steps) for comparatively expressive modelling languages in a small amount of computation time. However, the issue of strategic guidance is not automatically settled merely by employing abstract action decompositions; in fact, search in the space of decompositions is only a relocation of the POCL-related issues of selecting the appropriate goal, the appropriate conflict resolution, etc. The underlying problem is that the alternative task implementations will typically interact on the concrete operator level, but this interaction is in general not visible at the decision point of the expansion. This is, on the one hand, due to a lack of semantics and appropriate annotations, and on the other hand an in-depth analysis would nullify the gain of performance, because it is essentially computationally as hard as planning from scratch.

In order to address the problem of providing an effective search strategy, O-PLAN, SIPE, and SHOP enrich their domain model representation with more or less explicit search-control directives. O-PLAN uses typed conditions in order to provide the current implementation level with information about the intended causal structure, thereby anticipating what kind of commitments the system is going to make – this becomes apparent in particular for the annotation that describes the *level of plan generation* that the modeler intends to introduce the condition. The SIPE-2 modelling language contains a large number of search-related features,

for example, elements that provide for a discriminative treatment of operator preconditions that are on the one hand used for matching purposes only and on the other hand goals that are intended to be achieved actively [283]. Last, but not least, the SHOP system has been designed to include so much domain-dependent heuristics in its expansion methods that many people consider a problem-solving programming language rather than a planner.

Although we argue for taking advantage of procedural knowledge wherever this is possible, we firmly believe that the above mentioned ways of explicitly shaping the search tree are counterproductive. The problems begin with semantic issues, for example, what the precise connection between the plan generation process and the modelled domain is (this criticism also includes SHOP). Some parts of the domain knowledge are explicitly represented – that is what used to be the key aspect of planning – some parts are implicitly hidden in search heuristics: thus it remains unclear whether a path in the search space is excluded by the domain model because it does not lead to a legal solution or because for some other reason there is *presumably* no solution. But this dilemma continues up to the very fundamental question what consistency for such domain models means and how it can be verified. The latter would involve checking for consistent operator specifications, verifying whether implementations are executable, analyzing whether method applicability conditions guarantee termination on recursive method definitions, and so on. Last, but not least, any change to the search algorithm puts the functioning of the planning application on that domain at risk.

The practical implications are obviously the following: domain models are harder to develop, harder to debug, and finally harder to maintain, extend, and re-use. For instance, in the AIPS 2000 planning competition, "the SHOP team was developing domain descriptions for SHOP purely by hand, and made some mistakes in writing two of the domains. Thus SHOP found incorrect solutions for some of the problems in those domains, so the judges disqualified SHOP from those domains" [199]. We would like to add that it is not clear in this case what kind of alternative to a "handmade" domain model can be provided. Any tool-support will face the difficulty of including the planning-system specific knowledge in order to detect modelling errors – we believe this is not very realistic to pursue (we are not aware of any existing tool).

We conclude this discussion with an appeal for purely *declarative* domain models, at least for domain-independent planning. As efficiency improving strategic model components may be, we believe that there is enough practical evidence in other fields that cast them into doubt: The success of constraint satisfaction techniques in problem solving, as well as the renunciation of programming methods that are tightly coupled with specific compiler settings (that is to say: standardization of programming languages, reflection on their semantics) have shown the great advantages of decoupling models from solvers and execution environments. Planning should always adhere to this principle.

The practical oriented attitude towards action abstraction, namely using decomposition as a macro-definition for un-expensively pumping steps into the plan, has dominated the field for many years. But there is much more to HTN planning, and the theoretical side of it has been described and analyzed first by Kutluhan Erol [83]. He defined the first semantics for HTN planning and examined its expressiveness.

The reader may have noticed that we introduced the term *task* for referring to abstract actions. In fact, a task is more than only an action for which a set of decompositions exist: it is at same time a plan step and a goal. This small particularity was of course also known to the HTN pioneers, they did however not appear to have a very precise idea what the consequences are: a fundamental change to the way of generating plans, namely addressing the planning problem by a problem reduction search, rather than state space search. Hand in hand with that notion goes a tremendous increase of expressiveness of the underlying planning formalism. This can be proved by reducing the classes of plans that can be generated by a formalism to classes of grammars: In this view, action-based planning is analogous to right-linear (regular) grammars while HTN planning is analogous to context-free grammars [84]. There has also been some work that projected HTN planning into a general refinement-based planning framework and analyzed its behaviour in comparison to partial-order planning [146].

A simple example that demonstrates the limited expressiveness of action-based planning is the following: Try to plan a round-trip transportation such that the leg of the trip from X to Y always uses the same carrier as the leg of the trip from Y to X. An HTN specification of that problem can be modelled by an initial task network in which the legs of the intended trip are represented by appropriate movement tasks and the carriers are assigned by respective parameter bindings. In order to make this problem solvable by a non-hierarchical

approach, the outlined domain model has to be supplemented by artificial operators and conditions. The need for the additional information arises from the state-centered view: In the goal state of the round-trip, the acting entity is in the same location where it started from. Hence, every location on the round-trip has to be represented by an artificial property that encodes whether or not the location has been visited before. From that we get a modified goal state: having visited every location and being "again" in the starting-location. However, this still does not capture the notion of the concept "trip", that means, location *X* has to be visited before location *Y* and again vice versa on the way back. In the end, additional state features have to be introduced that encode the location (sub-) sequences that are to be visited and some more that ensure a consistent binding of the carrier. It can be shown that these extensions lead to an exponential blow-up in the domain model size and accordingly increase the effort for finding a plan.

In compliance with his theoretical considerations, Erol and his colleagues developed UMCP, a fairly puristic implementation of planning by task decomposition that is commonly considered the reference implementation for the HTN paradigm [85]. It uses ordinary STRIPS-style operators for modelling primitive actions, while abstract tasks do not carry preconditions and effects. Task networks consist of plan steps, ordering constraints on the steps, and variable equations. In addition, there are constraints that model causality: conditions can be requested to be true at the beginning of a task, at the end of a task, and between two plan steps. Method selection is done non-deterministically, that means, it is a matter of the search strategy to choose the appropriate one. After a decomposition step, the newly introduced constraints are propagated into the existing plan, until that plan becomes completely primitive. In that case, the pre- and postconditions of the operators are analyzed for causal conflicts and an executable plan is extracted. If that cannot be done or the constraint sets become inconsistent before the primitive level is reached, the system backtracks over previous expansions.

The conceptually clear separation of search-related issues from the declarative domain model representation enabled experiments with different refinement strategies in the UMCP system. We will discuss some of them later in a chapter dedicated to planning strategies.

As an interesting side note, the discussion about knowledge-intensive domain models versus light-weight models has been kept alive ever since. In particular this dispute persists between the traditional knowledge-rich HTN-endorsers and the non-hierarchical planner developers (cf. [285] and some arguments in [146]). The typical criticism on HTN planning is that procedural knowledge is regarded "cheating" because it is not the machine that designs the course of action, and that formalizing such an amount of knowledge about an application domain is not realistic (too many interviews, unclear knowledge engineering process, etc.). It is also a topic for supporters of models with a strong notion of causality versus workflow management researchers, for example, in the field of service construction. Be that as it may, we believe that the preceding sections made clear that procedural, respectively hierarchical knowledge is valuable and an adequate means of representation in many application areas.

1.1.4 Hybrid Planning

Our understanding of "hybrid planning" is the combination of planning with procedural information, as performed by the HTN approaches, with action-based techniques like those provided by partial-order planning. This combination turned out to be most appropriate for complex real-world planning applications, including scenarios like crisis management support [26, 44] and the like. In the hybrid paradigm, the solution of planning problems often requires the integration of planning from first principles with the utilization of predefined plans to perform certain complex tasks. Experiences in developing real-world, fielded planning systems at the Jet Propulsion Laboratory led to the conclusion that it adds the strengths of both, at the same time softening their weak points. This is evinced in both modelling and search efficiency issues [87, 88].

As for the modelling process, task networks of HTN planning represent hierarchy and in some cases modularity more naturally than plain operator-based models do. This enables the user to represent domains in an object-oriented form (including object type hierarchies) that is easier to write and reason about. Decomposition rules can refer to either low- or high-level forms of a particular object or goal, as the information

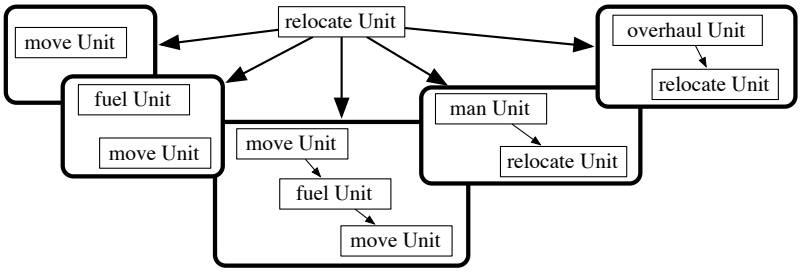


Figure 1.7: An example for the treatment of exceptions in HTN planning models. It may result in over-specified methods.

pertaining to specific entities is contained in smaller, more specialized rules. The drawback of an "HTN-only" technique is that *inter-modular constraints* cannot be represented adequately [87], that means, exceptions or special cases in action execution,because they typically lead to overly specified reduction rules. This becomes evident in the situation from a disaster relief scenario depicted in Fig. 1.7, where classical hierarchical task network planners need to provide expansion schemata for every combination of support tasks to be included in the relocation task: In some cases, a unit just moves to its destination, in others it has to be fuelled in advance, or it has to move to a fuel depot first and can then be fuelled, and so forth.

Operator-based techniques can help encoding implicit constraints by preconditions and effects of actions. Therefore, their kind of plan refinement is more general and provides more compact representations. In the example above, the movement task would just carry preconditions to ensure that the unit is fuelled and manned, and if these cannot be satisfied in the decomposition of the relocation, the necessary tasks are inserted in the plan in the fashion of partial order planning. In addition to the flexibility in model representation, these techniques provide an early detection of inconsistencies at abstract plan levels together with the means for resolving such conflicts. But when relying solely on atomic operators, certain aspects become difficult to represent as we have seen in the previous discussion on the expressiveness of HTN planning. The advantages of a natural mixed domain knowledge representation are obvious, although difficult to evaluate quantitatively:

> "[it is] easier to encode the initial knowledge base, fewer encoding errors occur [...], and maintenance of the knowledge base is considerably easier." [87]

Such hybrid systems had been watched suspiciously a long time, because the planning paradigms were considered to be antithetic such that no algorithmic procedure can implement both at the same time. As we have mentioned before, this was mainly a result of conceiving HTN planning as a procedure for the systematic unfolding of rules in a grammar that describes solutions, which is incompatible with producing exceptions to that solution language by introducing "terminal symbols" on demand. However, new AI textbooks present this method in the style of state abstraction planning in [294], that means, the abstract tasks carry preconditions and effects from a subset of the less abstract tasks. Yang suggests to keep hierarchical models restricted in such a way, that in every reduction schema there is exactly one task carrying the *main effects* of the network and hence those of the associated abstract task [292]. In such domains the *downward solution property* (all consistent abstract solutions can be refined into consistent primitive solutions) holds as a basis for effective search space reduction. A similar approach is presented by Russell and Norvig, who allow the distribution of conjuncts of conditions among the sub-tasks of the decomposition network [223].

One of the very few documented hybrid systems is DPOCL [298]. It decomposes abstract tasks into networks with additional initial and final steps, which carry the conditions of the abstract tasks (this corresponds to the approach of Yang). Some of the techniques used in this context raise the crucial question of *user intent*. The system prunes unused steps and takes condition establishers from every level of abstraction, even from sub-tasks of potential establishers. The problem of when to insert new tasks and where to solely use decomposition rules is very hard to solve because it partly depends on the modeler's intention. Our

proposed solution is to leave it to the user whether or not new tasks are allowed to be inserted for goal achievement. Moreover, premature insertion of new tasks may lead to non-optimal long plans, but we postpone this problem for this time as a matter of in this sense "good" search strategies, like it is solved for classical state-based nonlinear planners – but of course it has to be tackled in the future (cf. Sec. 3.3.1 and 6.6.4).

Another fielded hybrid planning system is the "Deep Space Network (DSN) Antenna Operations Planner" DPLAN, developed at JPL [58]. It automatically generates tracking plans for a set of highly sensitive radio science and telecommunications antennas. DPLAN accepts as input an equipment configuration and a set of requested antenna track services. The system then uses a knowledge base of antenna operation procedures to generate a plan of activities that will provide the requested services using the allocated equipment. The procedures are connected and extricated from negative interactions by action-based techniques.

More closely related to our approach is the work of Kambhampati [149]. It integrates HTN planning in a general framework for refinement planning, thereby making use of operator-based techniques. This unified view is intended to prepare the use of recent progress in planning algorithms, for example, by giving propositional encodings for SAT based planners [173]. In this view of hybrid planning, the algorithm uses reduction schemata if available and primitive actions otherwise. Causal interaction is analyzed at the abstract level as well and refined by mapping conditions and effects of abstract tasks onto conditions and effects in their sub-tasks. Abstract conditions are closed by *phantom establishers* that are to be identified at a later stage of plan development. Conflict detection and resolution can only be done at the primitive level, as in contrast to our methodology there is no "vertical" link between causalities in the different levels of abstraction (cf. our use of axiomatic knowledge, p. 34). Kambhampati addresses user intent by defining a subset of abstract effects explicitly for condition establishment, and by explicitly representing the incompleteness of schema definitions. For the latter, a specific predicate prevents insertion of new steps.

We believe, that it is not necessary to classify effects or conditions of abstract tasks as *primary* and *side* like it is done in nearly all of the presented systems, because we assume that the domain modeler encodes this information by actually building the task hierarchy including our approach to specify abstract conditions. In our view, abstract tasks carry primary effects only, while side effects are the additional effects introduced by their expansions.

Exploiting object hierarchies for action abstraction is a comparatively new technique in planning and rarely used. The main advantages are that it is based on an established modelling paradigm with more or less clear semantics, as well as that it supports reasonable commitment strategy along object components. Semantic foundations for such object oriented approaches can be found in the literature, ranging form reasoning about object database models in the style of terminological logics [40] to specification oriented work [237]. For example, the approach described in [96] uses plans as object methods for an hybrid reactive robot controller. Incoming percepts are mapped on partially specified object templates as plan selection criteria. Although this it is not applied to plan generation in the strict sense, the involved reasoning is close to precondition reasoning from a technical point of view. But there has also been some work more central to the field.

An approach that is based on a well-founded semantic concept is that of *object centered planning* [183]. The corresponding Object Centered Language formalizes plan generation in terms of object oriented concepts and basically grounds operator definitions in state transitions of the *involved objects*. In this paradigm, objects are categorized into static and dynamic sorts, and each instance of a dynamic sort has its own *local state* that is defined by a set of predicates, the so-called *sub-state*. Reversely, these predicates are owned by exactly one sort, the key attribute of the predicate, thereby becoming static or dynamic themselves. For all sorts in the domain model, the legal local states are specified. Transitions over these legal states constitute state automata that specify the behaviour of the objects of that sort. Operator definitions are consequently derived from the state automata of the involved, that is to say, manipulated objects. Building a plan in this approach basically consists of re-constructing the necessary (local) state transitions and indirectly identifying the operators that are responsible for the specified change.

McCluskey's OCL_h [180] extends the object centered formalism to action abstraction by introducing a sort hierarchy, in which dynamic predicates are inherited from super-sorts. So-called *guards* play the role of pre- and postconditions of object-transition sequences that constitute the semantics for abstract tasks. The

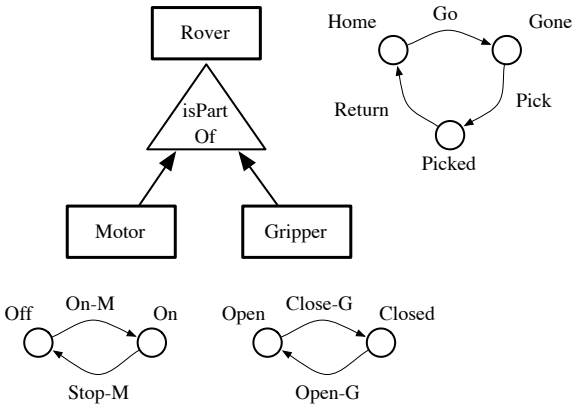


Figure 1.8: An example for an HYBIS agent hierarchy (taken from [44]).

planning algorithm for making this framework operational repeats an *expand-then-make-sound* cycle (EMS): for each abstract action in the current plan one expansion is performed and then the network and its induced state transition sequences are validated against the sub-state specifications. Any potential inconsistencies are repaired, if possible (which corresponds to threat resolution in POCL planning).

The EMS system appears to address a number of important requirements: It has a clear and simple semantic concept, it allows for a formal specification of the action hierarchy and task decomposition, and it seems to perform reasonably well. But although there are application domains for which state automata are most adequate models, for instance technical environments, we feel that for human oriented applications they are not. These domains are naturally activity centered and not object centered. Furthermore, we are intending to extend the concept of abstraction not only on objects but also on state features themselves, and this is not supported in OCL_h. It is also not clear how to extend this formalism to richer representation features like resources, etc. We finally note that we understand that the object-centered formalization overly restricts the strategic options by being tied to the expand-then-make-sound algorithm. We will deal with this particular strategy in later chapters.

The concept of using object-oriented knowledge in task decomposition is also followed in the HTN system HYBIS [44,46]. The originality of this approach lies in its view on the application domain as being composed of a hierarchy of action performing *agents*. An agent is represented by a set of state variables and an automaton that describes its behaviour. The transitions of this automaton are the actions the agent can execute (note the difference to OCL_h, where actions are *derived* from multiple state automata). A domain model consists of a *compositional* hierarchy of agents, that means, that higher level agents are aggregations of lower level agents. While agents at leaf nodes of the hierarchy correspond to real world entities that are able to act, aggregating agents represent abstractions thereof. Their abstract behaviour is related to that of lower levels via an interface and an articulation function that translates abstract actions into their refinements.

Fig. 1.8 shows an example for a compositional agent hierarchy (upper left corner): the root concept is an autonomous robot, the Mars Rover, that can execute three compound actions. On this level of abstraction, the platform can "go" from the home base to a target location, in that state it can "pick" up an object, and finally it "returns" to the home base. The Rover is composed by two primitive agents: a "motor" and a "gripper". Both can perform the lower-level actions of activating and stopping the engine, respectively opening and closing the hand of the manipulator arm. The relationship between properties of the Rover and its components are defined in appropriate interfaces, for example, the location of the motor platform is the same as that of the whole Rover. Typically, not all state properties do have a direct correspondence in the next abstraction level in terms of literals and are therefore preserved for lower levels. Hence, the approach is implementing in this way a knowledge *transformation* in contrast to the discussed state-abstraction methods.

The algorithm starts with an initial plan specification and proceeds down the agent abstraction levels. For

each step in a plan at a given abstraction level, the sub-agents' action repertoire is used to construct a plan fragment that achieves the same goals (modulo interface translation) in the plan for the next abstraction level. The system thereby keeps track of the current plan's (causal) consistency, also taking into account the component agents' state automata. In order to preserve causal commitment during decompositions, the HYBIS approach introduces *hybrid* causal links that represent commitment *between* two levels of abstraction. For example, the abstract "go" in Fig. 1.8 establishes the location of the Rover for the "pick" action. After a translation of the abstract movement into more concrete actions that are carried out by the motor agent, causality is temporarily restored by identifying the translated effect in the motor state with a hybrid causal link. When finally the "pick" action is taken up by the actions of the gripper agent, the system is able to introduce the final (classical) causal link. Note that this kind of expansions differ significantly from the usual HTN methods, since the expansion networks are not user-defined but generated during run-time on demand.

The HYBIS approach is an interesting alternative to the OCL_h framework, since it enables a declarative specification that is based in the commonly used automaton representation and is at the same time action-centered. Like it is the case for the other object-oriented approaches, the formalism is however mainly intended for technical environments. HYBIS is, for instance, explicitly targeted on production planning, where the agent-composition metaphor is highly adequate. In many other scenarios the identification of agents and their components is rather artificial and leads to flat compositional hierarchies, which undermines the application of the divide-and-conquer principle and therefore compromises the efficiency of the approach significantly.

A second drawback of the approach is that it is fixed with respect to its level-wise decomposition strategy. Although there is no mingling of procedural domain knowledge and search-related information, the domain model and the search algorithm depend indirectly upon each other via the construction of the composition-hierarchy layers. This hampers model development, maintenance, and reuse, although to a much lesser extent than it can be observed in other approaches.

Recent developments investigate into combining the two paradigms by simply assembling an HTN and a non-hierarchical plan synthesizing system within a coordinating algorithmic frame [110]. According to the goal type, the hierarchical planning system is thereby called for reducing abstract actions into primitive plans and the non-hierarchical planner is responsible for synthesizing action sequences that satisfy the state-based goal conditions. These kind of approaches are however not in our focus, since they do not employ coherent universal semantics for all involved components.

1.1.5 Other Planning Techniques

In this section, we want to give a brief overview over a set of research directions that are currently pursued in the field of AI planning and that have a high potential regarding integration efforts. We begin with two formal, very general problem solving techniques that are successfully employed for plan generation: theorem proving and SAT solving.

In the above section on classical planning we have mentioned the situation calculus [124]. A first-order predicate-logic language is employed for representing states and state changes. An essential technical element of this formalization is the notion of explicit *situations*, which are added as a state parameter to every state-dependent, flexible atom, and the notion of actions as state changing functions. Two aspects are thereby specified in this calculus: *effect axioms* and *frame axioms* [179]. Effect axioms define the state change induced by an action, and the respective frame axioms specify what aspects of a state are not affected by it and therefore persist. Planning is performed by proving a plan specification from the axioms, which is essentially an existentially quantified formula that claims the existence of a goal situation in which, given an initial situation, a specified goal condition holds. If a constructive proof can be found, the generating terms of the concrete goal situation can be extracted, that means, the sequence of action terms that contribute to the goal state can be identified. To this end, Green and his colleagues McCarthy and Hayes used a dedicated answer literal for accessing the goal situation substitution in a resolution calculus proof.

Although this approach has been first published 40 years ago, it still influences today's view on planning and in particular that of planning by deduction. For a review of deduction-based planning, see [22], among

others. The main advantage of these techniques lies in the precise semantics of *consistent* domain models and the underlying calculi. A formal specification describes the domain model in a declarative and modular manner, which gives the means to a systematical (and tool-supported) modelling [27]. It is even possible to formulate, for instance, safety conditions against which the domain model can be verified. The employed logic is thereby decisive for the expressive power of the domain models, the available options with respect to the usable reasoning techniques, and of course the computational complexity of the procedure. Many formalisms have been examined, ranging from dynamic logic to temporal logic and logic programming (Golog).

The operational software for deductive planning systems is typically a general automatic theorem prover, which has not been specifically designed for the task of planning. Although the structure of the proof space obviously depends on the specifics of plan generation in general and on the problem and domain specification in particular, these dependencies is hard to formalize and to exploit, because a logical proof has no clear relation to the structure of the plan it represents (which makes it hard to benefit from the experience in other fields of planning). Formulating adequate proof tactics for the chosen domain description language is hence a central research topic in this area.

With the availability of very efficient SAT solvers, formulating planning problems as satisfiability problems turned out to be a successful compromise between reduced expressiveness and computational performance [151]. Planning as satisfiability is basically a process that involves three steps: First, the problem and domain specifications are translated into a propositional formula. Fully grounded operators are encoded in propositional axioms that do not only describe the precondition and effects but also the discrete time point at which the operator is potentially executed. For example, there is a proposition for executing operator x at time t, with t ranging over the temporal planning horizon, together with implications like "if x is executed in t, its precondition must hold in t and its effects will hold in $t+1$". Every ground atom that is built in this way, is interpreted as a propositional variable. Since every ground instance of an operator has to be represented by a variable, the number of variables is in the order of the product of the number of schemata, domain objects, parameters per operator, and finally the expected length of the plan. The main efficiency gain for this technology lies in a concise translation. In a second step, a model is constructed that satisfies the generated formula; this is where standard SAT techniques are applicable. As the third and last step, the plan is extracted from the generated model by collecting the "operator executing" variables that have been set to the value "true". For more details on SAT-based planning we refer the reader to the review in [280].

This development lately culminated in the BLACKBOX system [152]. In this approach, STRIPS-style problem specifications are first converted into Boolean satisfiability problems, and then these problems are solved with a variety of state-of-the-art SAT-solver engines. Its efficiency is based on two techniques: It employs the GRAPHPLAN planning system (see above) in a preprocessing step for determining the mutual exclusive actions over the time horizon: From that information, a strongly reduced propositional encoding is obtained. Second, it can change the SAT solver during search; for example, if after a given amount of computation time this subsystem has not produced a result, another solver is activated. This gives BLACKBOX the capability of functioning efficiently over a broad range of problems and it also gave the system its name because the plan generator knows nothing about the SAT solvers, and the SAT solvers know nothing about plans: each is a "black box" to the other.

There exists also an approach on applying SAT techniques on encodings of hierarchical models [173] but we are not aware of any subsequent work on these preliminary findings.

Another direction of research that has been inspired by formal methods is planning based on model checking. The planning domain is represented as a non-deterministic state transition system, hence these approaches address planning problems that deal with uncertainty of various kinds: non-determinism, partial observability, and extended goals (a goal has to be achieved necessarily, eventually, at least in some cases, etc.). Our interests are however more focused on formalisms that support abstraction and resource reasoning, preferably in a more natural, human readable way.

HTN planning has some motivation of capturing best practice of an application domain and encoding it in the planning model. A similar motivation have case-based techniques: A database of problems and solutions is maintained such that if a new problem is encountered, the solution to a similar problem can be retrieved and then adapted to the new problem. The rationale is to benefit from experience. We have mentioned before the

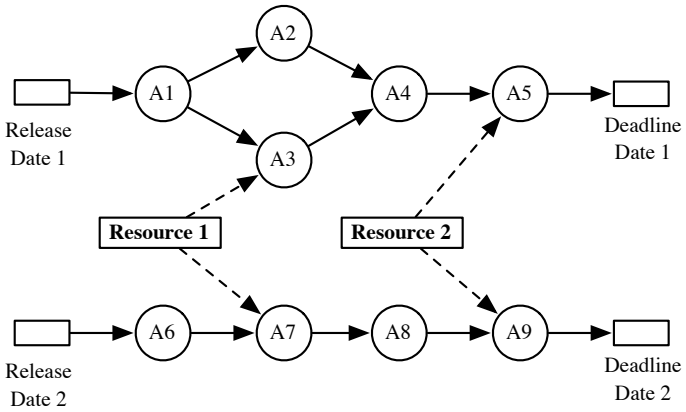


Figure 1.9: A simple scheduling problem.

HACKER system [253] that re-uses plans in a library, if possible. More advanced case-based planners, for instance CHEF [127] and DIAL [167], are very popular in domains where domain knowledge, respectively a solution quality metric is missing. In this context, we distinguish "solutions" and "solutions that are accepted by users". From a technical point of view, the main characteristic of case-based approaches is that they realize the *transformational planning* paradigm. Practically all presented HTN and POCL approaches implement a constructive *refinement planning* method, in which information is only *added* to the current plan. Transformational planning can *remove* choices and is typically realized as a local search in the plan space.

1.2 AI Scheduling in a Nutshell

As it has been mentioned in the introductory section, scheduling is the process of reasoning about *how* a given set of actions can be performed using a limited number of *resources* in a limited amount of time. The term "resource" thereby denotes any object or (typically consumable) substance that induces some constraint on the actions that use it because of its cost or available quantity. The objective of scheduling is to generate a *schedule*, that means, to assign resources and time-points or -windows to actions for given temporal and resource *constraints*. The constraints impose restrictions on the choice an action has between several alternative resources or time assignments: A solution schedule has to meet a deadline, has to take into account a predefined ordering of the actions, has to consider the availability of resources, and the like.

Fig. 1.9 depicts a simple scheduling problem: Actions 1-5 and 5-9 are ordered by precedence constraints, actions 3 and 7 are pre-assigned to resource 1, actions 5 and 9 to resource 2. The problem has defined release dates and deadlines each chain of actions has to meet. We may assume that each action can be assigned any of the two resources. Any resource can be accessed by at most one action at a time. The problem is to find an assignment that meets the deadlines.

The main conceptual difference between planning and scheduling is that in scheduling, the actions in the schedule are known in advance. In addition, scheduling problems are optimization problems, typically minimizing the total duration of the schedule (makespan), minimizing the idle time of resources, or maximizing the utility/cost ratio.

Since optimization has always been a key issue in commercial application environments, there are countless deployed scheduling systems – either as dedicated schedulers or as back-end engines to enterprise software. The most proximate application area is scheduling of industrial processes, which includes production scheduling [226], manufacturing (for example, semiconductor manufacturing [97] or steel manufacturing [74]), and workforce scheduling [73, 227]. Closely related from the technical perspective in terms of

user requirements, etc., is scheduling of logistics in military operations [64], but also air campaign scheduling [78]. Like for planning, the most prestigious application domains are of course space missions, for example, observation planning for the Hubble space telescope [140, 193] and data downlink planning for an autonomous interplanetary spacecraft [49].

1.2.1 Classical Scheduling

The classical scheduling techniques originate in the field of Operations Research (OR), a discipline that particularly addresses optimization problems with mathematical models, statistics, and the like. OR literature established over the years a systematic classification catalogue for scheduling problems that is based on problem features and that enables a practitioner to choose the most appropriate and efficient solving method for the problem at hand. The structure of a scheduling problem thereby mainly depends on its temporal constraints, the number and types of resources, and how resources are allocated. The most important technique available today is *Mixed Integer Linear Programming* (ILP/MIP): Given a set of variables (some of which are constrained to be integer), a system of linear inequations, and a linear objective function, the output of the algorithm is a value for each variable that maximizes the objective function.

$$a_{i,j}, b_j, c_i, x_k \in \mathbb{R}, \text{ with some } x_l \in \mathbb{Z}$$
$$\text{maximize } \sum_i c_i \cdot x_i \text{ subject to } \sum_{i,j} a_{i,j} \cdot x_i \leq b_j$$

Most practically relevant scenarios are covered efficiently by this technique.

Although this just touches the tip of the iceberg and although OR-related techniques are well-established and successfully deployed, we have to omit most of this research as it is not suitable for our needs: All optimization algorithms (a) solely work under the assumption that *all* actions are known in the problem specification, (b) can only produce a "final", completely grounded solution (they do not explore a search space such that they could be synchronized with a planning algorithm), and (c) are often optimized implementation for exactly the target application's problem class. For more details on the problem classification catalogue, algorithms, and application scenarios, the reader may refer to [37, 216].

1.2.2 AI Scheduling

Artificial Intelligence techniques increasingly influence the scheduling research community. Due to their expressive power, AI formalisms and solvers provide a very natural way of modelling for most application areas (sometimes they even make innovative application areas accessible) [299]. Furthermore, a large number of efficient technical solutions is available on the market that can often be easily combined with each other, like search heuristics, inference mechanisms, etc. [141, 161]. In fact, search-based solvers exhibit a dramatically increasing performance, surpassing OR techniques for many problems.

Constraint Satisfaction Problem solving (CSP) has always been the AI technique that is most widely used for solving scheduling problems. A CSP is given by a set of variables together with their respective domains (possible values) and a set of constraints that define the compatible values that the variables may take (subsets of the power set of the domains). The problem is to find a value for each variable within its domain, such that these values meet all the constraints. Solving a CSP includes two techniques: a search algorithm for exploring the possible values for each variable and a propagation algorithm that infers consequences of the value assignment decisions from search and makes the inferential closure of the constraints explicit. Since constraints represent value restrictions on the variables, propagation implies domain reduction, and if a domain is reduced to the empty set, the CSP is found to be inconsistent and hence has no solution. As a technical note, many CSPs only make use of binary relations in the constraint set and can therefore be interpreted as a graph structure with the variables being nodes and constraints being the vertices between the constrained variable nodes. Under certain conditions, these structures allow for highly effective propagation algorithms. This is, for example, the case in temporal relations and hence CSP solvers are often used to manage very expressive temporal constraints efficiently as dedicated subsystems.

For example, in a production scheduling application, a CSP can be set-up as follows: For each action that is to be scheduled, the CSP contains three variables, namely the allocated resource, the start time, and the end time. We may assume that the resources are explicitly enumerated (machine names, etc.) and that the domains of the time variables are intervals over time points. The constraint set includes constraints for relating start and end times, that means, for a given duration d of an action, the associated end time is constrained to be exactly d time "after" the start. But it also contains precedence information such that for any action a_1 that precedes an action a_2, the end time value of a_1 is less or equal to the start time value of a_2. If the search algorithm sets the values for the start time of any action to a specific value, say, the begin of the day shift, the propagation algorithm starts to disseminate this information into the other variables: First, the end time of the action is reduced to the only value that is consistent with the constraint, that is, day shift start time plus duration of the action. Second, setting the end time is reflected in the start time of successor actions, because their lower bound has been updated. This information, in turn, affects the successor actions' end times, and so forth. On the other hand, there are also *global constraints* that are only fulfilled if a resource is used at most once in every time interval, and the propagation of the temporal constraints may therefore reduce the set of available resources for some actions. In this way, changes to variable domains are spread until finally every action is assigned a definite resource and execution time window.

Concerning the search algorithm, scheduling via CSPs has two options: (1) It may employ a constructive search method such that it progresses incrementally assigning values to variables, propagating the constraints, and backtracking when violations appear. Since this is a technique of refinement that operates on a partial schedule, the evaluation of global criteria can only be approximated. The positive aspect is however that either a systematic search can be realized that explores all possibilities or a heuristic non-systematic one that considers only "promising" possibilities. (2) The second option is to perform an iterative repair, respectively local search methods: These algorithms start with a complete assignment of values to variables and then reassign new values to variables in order to resolve violated constraints. The evaluation of global criteria is evaluated with low cost. However, one of the main disadvantages local search methods is that they can suffer from local minima and are often incomplete.

Note that since AI scheduling is basically a search-based approach, it is in line with planning techniques. In fact, constraint satisfaction that uses constructive search is part of planning technology as far as it concerns managing parameter bindings/equations and plan-step orderings.

1.2.3 Integrated Planning and Scheduling

As it has been discussed in the introductory section, an integration of planning and scheduling methods is urgently needed because many applications will benefit from it. Typical real-world problems involve both action planning and scheduling because they contain resource requirements as well as causal and temporal relationships [282]. Beyond the challenge to produce complex courses of action, adequate planning support in domains like crisis management [9, 26, 44, 70, 261], evacuation planning [262], or spacecraft assembly and operation [1, 87] has to consider all kinds of resources; they range from limited time and material to power and supplies and they all define success and efficiency of the mission. In addition to the selection of cited reflections on the issue, Smith, Frank, and Jónsson demonstrate how many difficult practical problems lie somewhere between task planning and scheduling, and that "neither area has the right set of tools for solving these problems" [239].

The importance of the topic has led to a variety of approaches that perform temporal reasoning [6, Chapter 1] and resource computations [281] during plan generation; in this overview section we want to briefly examine the relevant directions. We thereby only focus on approaches that employ a strong methodology in the sense of AI scheduling: A counter-example is the "manual" computation in the SHOP system [202], which allows for resource calculations along the task reduction schemata that are *programmed* by the modeler. This also includes GRAPHPLAN extensions like the *resource time maps* of IPP [159] or numeric enhancements of SAT-based approaches [222] that try to calculate the effects of parallel resource manipulations.

Practically all existing resource-aware planners use resource information for *pruning* the search space, that means, to rule out those plans that do not allow for a consistent resource allocation. In order to do so, the

systems incorporate some form of a constraint-based resource management subsystem (see CSP methods above). The SIPE-2 HTN system, for example, feeds its output to the OPIS scheduling engine and backtracks if the plan cannot be scheduled [284].

But instead of generic schedulers, respectively CSP solvers, we often find system-specific implementations of resource-analysis algorithms: They determine for each resource which are the manipulating operators and classify these into consuming and producing actions. Considering the step ordering information, these resource reasoners try to project potential points of over-consumption of the resource.

O-PLAN [261] performs an *optimistic* and a *pessimistic* estimation for each resource profile [77]. The optimistic scenario for a plan is thereby that all consumption steps are performed as late as possible, allocating the minimal quantity possible, and that all production steps are performed as early as possible, producing as much as possible. The pessimistic estimator makes the inverse hypotheses. If the optimistic profile value drops below zero, that means, if even in the optimistic plan scenario there is at least one point in time at which the capacity of at least one resource is exceeded, then this plan cannot be developed further and search has to backtrack. The pessimistic estimator is of limited usefulness, though, and not considered for any strategic purposes. The system furthermore introduces constraints to evade potentially conflicting plans, for instance, if the optimistic profile drops below a given threshold. Based on O-PLAN's architecture with its constraint managers [18], a pure scheduling system TOSCA has been developed that performs an opportunistic search [17].

In the HYBIS system [43] we find an approach to hybrid planning that makes use of object aggregations as justifications for expansion schemata [44]. This is in some sense similar to a view on aggregated resources that we will be presented in a later section of this thesis (cf. [232]), although we do not restrict ourselves to a passive feasibility check and perform scheduling operations when dealing with this kind of abstraction.

Concerning scheduling-related functionality in POCL planners, the VHPOP system implements PDDL's durative actions [297]. It incorporates temporal information in a way that is similar to the approach we are going to propose: The ordering relation is substituted by a Simple Temporal Network, which allows to efficiently detect temporal inconsistencies [71]. The authors also intended to extend the expressiveness of the temporal constraints, but it appears that neither the system structure nor in particular the results obtained therein regarding POCL planning strategies can be transfered to such an extension.

A similar architecture is implemented in [107], where separate constraint-based planning and scheduling modules share a common memory. This system develops several potential solutions in parallel, using a strategy of stepwise constraint refinement. Again, the resource and temporal reasoning modules are solely used for constraint consistency checking.

The HSTS system [193] is the prototype for a controller of the NASA space telescope Hubble that is designed for operating the platform's observation devices with a minimal reconfiguration time; in order to cope with the amount of information that has to be taken into account, it schedules observation sequences on two levels of detail and time horizons. Conceptually closely related to it is the mission planning module for the autonomous space probe *Deep Space One* [194]. Both systems are constraint-based and generate schedules for parametrized plan fragments, which have to obey larger sets of mission critical constraints. Latest developments in the context of this kind of control software for autonomous spacecraft, the ASPEN platform [55], follow the paradigm of iterative repair of flawed schedules. The focus of these systems is clearly on scheduling and less on plan generation. However, [61] builds on it for an HTN planning approach that uses summarized resource information in tasks (for summarized symbolic conditions and their usage in the planning process see [62]). This form of abstraction recursively deduces bounds for local minima and maxima of resources in abstract tasks from information about resource allocation within more primitive tasks, thereby taking into account a possibly overlapping execution of tasks.

The IXTET temporal planning system [163] integrates planning and scheduling by using temporally qualified expressions throughout the representation formalism. These expressions represent state transitions and state persistences in the planning domain. We share the authors' view of opportunistic scheduling as additional plan modification steps that can be interleaved with other plan generation options: closing open or unachieved preconditions, resolving (resource) conflicts, and adding constraints to evade bottlenecks. The approach is provided with very efficient algorithms for determining potentially conflicting tasks in a

partial plan (also known as *minimal critical set* computation) [164], for performing a least commitment search that quantifies the level of commitment in every modification step, and so on. Another important feature is the dynamic construction of a resource hierarchy that is based on action-condition analysis in the current partial plan [104] (not to be confused with our notion of hierarchical resources as introduced in [232]). The hierarchy represents a partial order on the "importance" of the resources for plan causality, which induces the order in which the different resources should be preferably addressed by the reasoning process.

A complete contrast to the above presented techniques is the approach in [244], where planning and scheduling for symbolic resources are viewed as two processes that have to be completely separated. Therefore, planning is performed on a relaxed problem level where no resource information is available and the resulting plan is given to a scheduler, which performs the necessary resource allocation. The rationale behind this approach is preparing a causal operator skeleton that can be filled by a very efficient scheduling module. This framework is defined for planning problems, in which always the plan with the fewest steps is cost-optimal and resource information is not needed to guide the search. In many realistic domains, however, there are typically situations in which some goods have to be produced on demand. In these situations, the system falls back to a planner-only configuration.

The *parc*PLAN system [81] also performs a pre-planning phase: the durations of the plan steps are minimized in order to determine the necessary overlapping actions and the minimal resource capacities. It introduces meta-variables to represent potential interval overlappings. Their values are manipulated in the planning process according to optimistic and pessimistic estimations: necessary operations are performed because of over-consumptions, more opportunistic ones according to heuristics that aim at avoiding backtracking. This procedure induces ordering modifications, namely setting and excluding interval overlaps. Again, this approach solely ensures resource availability and does not guide the plan generation process actively by resource demands.

Practically all presented systems are limiting themselves to keeping book of the consequences of plan generation steps on the balance of resource availability versus resource consumption of quantities over time. However, this kind of reasoning does only detect the most necessary backtracking points, that means, situations in which backtracking becomes inevitable. It is difficult at least to guide the plan generation process itself towards "better schedules". This motivates methodologies that try to get more information out of the resource analysis and into the planning process.

Our aim is consequently to provide a framework in which planning and scheduling functionality is uniformly integrated. Integration should not be limited to a "two-subsystem" constellation in which a planner delivers plans to a scheduling system that criticizes the intermediary result; it should rather be possible to generate and abandon plans schedule-*driven* and vice versa.

1.3 Vision and Aims

Our introductory scenery motivated planning and scheduling by the cognitive phenomenon that appears to be one of the corner-stones of human intelligence. But beyond the curiosity to investigate and mimic the human mind, this research is in fact stimulated by the perspective to build systems that benefit from exhibiting comparable deliberative capabilities in terms of flexibility and robustness. On the one hand, this means software systems for supporting the human planning task, on the other hand it includes larger systems to which planning and scheduling components are contributing. The latter scenario is thereby not necessarily restricted to software and may be any kind of autonomous platform.

The above brief review of the state of the art in Artificial Intelligence Planning and Scheduling has shown that this discipline has not only developed a rich assortment of concepts and methods but also that its technology is provably a valuable contribution to system-support in key areas of academia and industry. Regarding planning, action-based approaches provide systems with the ability to flexibly synthesize their courses of action, while hierarchical planning techniques offer a great expressiveness, a natural knowledge representation, and nevertheless considerable efficiency. Scheduling methods are very established and

have become an integral part of countless business and industrial systems. We also believe that with *AI-based* scheduling components becoming more and more accepted, a higher degree of integration with other knowledge-based components will be accomplished – the synergy will make the technology even more successful.

We do however observe the tendency that the three areas action-based planning, hierarchical planning, and scheduling are not only pursued by three disjoint research communities, but even that this state stabilizes – this is, of course, only a subjective observation. The selective integration attempts appear to be more a symptom of this condition than a counter evidence: The implementations address specific aspects of an application domain, very often in an ad-hoc manner. This "horizontal fragmentation" of the field thereby increases the vertical one, that means, the traditional gap between basic and applied research.

Our aim is consequently to bring together the strands of the field in order to combine their strengths, open up new application domains, and improve the coverage of existing ones.

We want to accomplish the envisioned consolidation under the roof of a clean formal framework with precise semantics, that means, using *one* unified representation formalism and *one* integrated method that makes the framework operational. We are thereby particularly interested in providing the means for a transparent, declarative domain model specification and for a modular system design. While the former addresses the requirements for modelling support and consistency analysis, the latter will let us flexibly combine the characteristics and functionalities of the framework's method repertoire.

But integrating planning and scheduling techniques raises the question of adequate and efficient search control: While the mainstream systems have been thoroughly analyzed and equipped with highly efficient strategies, we have to explore that terrain for integrated methods. We want to develop the concepts for strategic advice of hybrid systems, but also perform some experimental evaluation to get a notion of their impact.

Last, but not least, we want to present an implementation of the framework that is intended to be more than just a proof-of-concept prototype. We want to demonstrate two aspects: First, that the implementation of the unified framework is feasible without a compromise between formal foundation and practicability. Second, the system is able to serve as an experimental platform for the evaluation of strategies and system components. This shall give evidence to the capabilities of the integrating approach.

We would like to point out that this thesis focuses on the **generation of symbolic plans and schedules**. This naturally excludes some important aspects of planning and scheduling like execution monitoring, plan/schedule repair, interactive planning and scheduling, on-line/dynamic planning and scheduling, multi-agent planning and scheduling, path and motion planning, and the like. We believe that our work lays the foundations for addressing at least some of these facets in future work and may stimulate research for extending our framework accordingly, respectively encouraging appropriate integration initiatives.

In order to accomplish the above goals and to realize the vision of an integrated approach, this thesis has the following agenda:

1. **A formal framework that covers hierarchical and non-hierarchical planning and scheduling:** Clear semantics of domain the model entities should facilitate integration and allow for a formal treatment of model consistency.
2. **Complete integration of hierarchical and non-hierarchical methods:** The hierarchical knowledge is supposed to contribute with highly expressive modelling concepts, while non-hierarchical concepts yield a flexible plan/schedule synthesis.
3. **Complete integration of planning and scheduling:** The combination is supposed to make planning more intelligent due to resource-aware scheduling advice as well as make scheduling more intelligent due to causality-aware planning advice.
4. **Flexible planning strategies:** Planning and scheduling strategies should take advantage of the integrated methods in an efficient and opportunistic way.

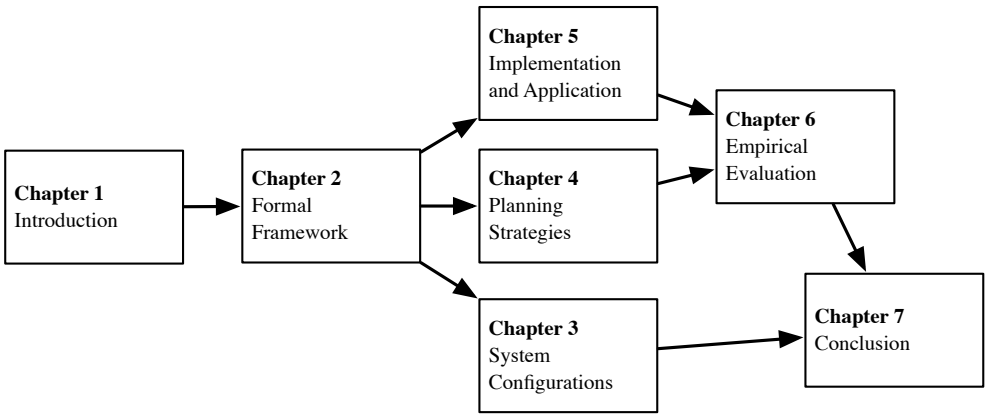


Figure 1.10: Dependencies among the chapters.

5. **An effective software platform:** The system implementation is supposed to allow for a flexible tailoring of planning and scheduling technology as well as for systematic experimentation with and on all components.

1.4 About this Document

The structure of this document corresponds to an implementation of our research agenda. All chapters include a discussion of the results and references to related work on the more specific topics. Fig. 1.10 depicts the dependencies among the chapters.

We will begin with establishing the formal framework in Chapter 2 and define all entities that constitute a domain model. We will describe how planning problems are specified, characterize solutions, and introduce the basic mechanisms for developing plans into solutions. In that chapter, we will also propose our generic algorithmic framework for solving hybrid planning and scheduling problems. Chapter 3 is dedicated to instances of the framework that realize the integration of methods, for example hybrid planning (Sec. 3.3.1). It will be shown how these instances are built by means of the framework and how they can be extended. All relevant issues concerning search strategies will be discussed in the succeeding Chapter 4. We will demonstrate how strategies can be defined in our proposed framework and present a number of new strategy creations that can be employed in all kinds of integrated systems. Chapter 5 deals with the application aspects of our approach, that means, the realization of the framework as a software artefact as well as the specification of models for exemplary application domains. Regarding implementation issues, we develop in Section 5.1 a system design that originates in knowledge-based systems and middleware technology likewise. In Section 5.2, we demonstrate the main issues in hybrid domain model specifications and discuss design options. The technical part of this thesis ends with Chapter 6, in which we document an experimental study that we conducted within our framework on a selection of strategies in the context of hybrid planning. Our results provide a first insight into the performance characteristics of the strategies and into some properties of the examined domain models and problems. In addition, they attest the suitability of our approach to serve as an experimental platform for developing planning and scheduling systems and strategies. We conclude in Chapter 7 with a general perspective on future developments and a summary of our contributions.

2 A Formal Framework For Refinement Planning

THIS chapter presents the formal framework that we have developed for hybrid planning and scheduling. Instead of giving an informal description of "the usual meaning" for states, actions, plans, and the like, we emphasize that the foundations of our approach lie in a sound, logic-based formalism for domain-independent planning and scheduling. We thereby do not only cover the syntax and semantics of the domain entities but also that of a refinement-based plan generation process. This provides our approach with the following three unique characteristics.

Firstly, a logic-based approach per se allows for the unambiguous definition of the planning language elements and the constructs made thereof. It does not only give a clear meaning to all involved terms and concepts, it also provides the means to precisely specify the criteria for consistent domain models, meaningful problem specifications, and correct solutions.

Secondly, it facilitates the construction of a sound reasoning procedure: the plan-generation algorithm itself. We will define precisely what plan refinements are and how they can be operationalized properly.

Thirdly, our framework induces an architecture that enables us to flexibly set-up various system configurations from a broad repertoire of modular planning functionality, ranging from partial-order planning to HTN-planning and more. It is thus guaranteed that any implemented planning and scheduling system will work in a meaningful way, that is to say, the framework realizes a plug-in architecture for highly configurable sound planning and scheduling systems.

In the following sections we will define the framework's syntax and semantics. We begin with the definition of a fragment of first-order predicate logic that we employ as our common planning vocabulary. It is used as the language for describing situations in the application domain, the world states, as well as situation changes. From that, we develop the representation and semantics for actions and finally for plans. We furthermore propose two abstraction mechanisms that are relevant for hybrid planning and scheduling: an axiomatic abstraction of state features and an abstraction of plans by complex actions. It is one of the major novelties of our approach that these abstraction mechanisms are seamlessly interleaved.

As it has been stated in the introductory chapter, *plan generation* is the process of developing a course of action that solves a given problem specification. That means, the plan achieves some goal conditions in the modelled application domain, or it implements an abstract plan description. Such a problem specification can therefore be viewed as an "abstract" or over-generalized solution to the problem, a general idea of what is supposed to be done. In this view, plan generation means concretizing the initial plan sketch until a satisfactory level of detail is reached: a level at which all kinds of conditions are satisfied, all action interactions are sorted out, and all objectives are met. We adopt this notion and consequently introduce the concept of *plan refinement*, which is based on the semantics of actions, plans, and abstractions. From the idea of refinements, the fundamental techniques for building a universal planning and scheduling framework are developed: flaws and plan modifications. The latter are explicit representations of refinements that are used to construct a solution for a given problem specification, and flaws are the guides through the space of plan refinements that explicitly indicate, where the deficiencies lie in the current plan.

This chapter concludes with algorithmic procedures that incorporate the outlined refinement-planning framework and with a discussion of the presented results with an emphasis on search control issues. Concrete instantiations, that means, implementations of this plan-refinement framework and the planning algorithms will be presented in the subsequent chapter.

Throughout the following sections, we will incrementally develop a running example in order to motivate and demonstrate the application of the introduced concepts. To this end, we have chosen to propose a

simple and, as we believe, self-explanatory logistics scenario: computer peripherals are ordered, delivered, and set-up at a customer's place.

2.1 The Logical Language

We use a fragment of an order-sorted first-order predicate logic as the basic formalism for the proposed hybrid planning and scheduling framework. It is the *logical planning language*

$$\mathscr{L} = \langle \mathscr{Z}, \leq, \mathscr{R}_r, \mathscr{R}_f, \mathscr{F}_r, \mathscr{F}_f, \mathscr{V}, \mathscr{T}_p, \mathscr{T}_c, \mathscr{E} \rangle$$

which consists of the following components:

$\mathscr{Z}$:	A finite set of sort symbols,
$\leq$:	a partial order on the sort symbols in $\mathscr{Z}$,
$\mathscr{R}_r, \mathscr{R}_f$:	$\mathscr{Z}^*$-indexed families of finite sets of rigid and flexible relation symbols,
$\mathscr{F}_r, \mathscr{F}_f$:	$\mathscr{Z}^* \times \mathscr{Z}$-indexed families of finite sets of rigid and flexible function symbols,
$\mathscr{V}$:	$\mathscr{Z}$-indexed family of finite sets of variable symbols,
$\mathscr{T}_p, \mathscr{T}_c$:	$\mathscr{Z}^*$-indexed families of finite sets of primitive and complex task symbols,
$\mathscr{E}$:	A finite $\mathscr{Z}^*$-indexed family of elementary operation symbols.

$\mathscr{Z}^*$ denotes a word over the symbols in $\mathscr{Z}$, including the empty word ε. Since we want to employ equality in our formalism, $\mathscr{R}_r$ contains an equality-relation symbol $\equiv_Z$ for every sort symbol in $\mathscr{Z}$. For the sake of a better readability, we will always use the equality symbol in the infix notation.

All of the sets in $\mathscr{L}$ are assumed to be pairwise disjoint. The precise meaning of the terms *rigid* and *flexible* will be discussed later, but for now it is sufficient to regard them as mere labels for the respective symbol sets. For the sake of a brief notation, we also define the following auxiliary union sets: In order to refer to both labeled set variants jointly, we introduce the set of all relation symbols $\mathscr{R} = \mathscr{R}_r \cup \mathscr{R}_f$ and the set of all function symbols $\mathscr{F} = \mathscr{F}_r \cup \mathscr{F}_f$ of the logical language. The set $\mathscr{T} = \mathscr{T}_p \cup \mathscr{T}_c$ denotes all task symbols, primitive ones as well as complex ones. In addition, we define *rigid* and *flexible constant symbols* to be the sets $\mathscr{C}_r = \bigcup_{Z \in \mathscr{Z}} \mathscr{F}_{r_{\varepsilon,Z}}$ and $\mathscr{C}_f = \bigcup_{Z \in \mathscr{Z}} \mathscr{F}_{f_{\varepsilon,Z}}$ in the usual way as subsets of the respective 0-ary function symbol sets. For convenience, $\mathscr{C} = \mathscr{C}_r \cup \mathscr{C}_f$ denotes the set of all defined constant symbols.

The elementary operations-symbol set $\mathscr{E}$ provides for each flexible relation symbol $R \in \mathscr{R}_f$ a so-called add-operation $+R$ together with a corresponding delete-operation $-R$. In addition, the elementary operations include an update operation $:= f$ for each flexible function symbol $f \in \mathscr{F}_f$.

We begin with the definition of a running example that describes a very simple application scenario in which computer peripheral devices are transported and installed. For a start, we formalize the fact that printers are delivered in cardboard boxes; we will extend this scenario in the proceeding sections. For describing this application environment, our initial definition of $\mathscr{L}_{devices}$ may be given as follows:

$$\begin{aligned}
\mathscr{Z} &= \{\texttt{Thing}, \texttt{Printer}, \texttt{Box}, \texttt{Place}\} \\
\leq &= \{(\texttt{Printer}, \texttt{Thing}), (\texttt{Box}, \texttt{Thing})\} \\
\mathscr{R}_f &= \{\texttt{Open}_{\texttt{Box}}, \texttt{Closed}_{\texttt{Box}}, \texttt{On}_{\texttt{ThingPlace}}, \texttt{In}_{\texttt{ThingBox}}\} \quad \mathscr{R}_r = \{\} \\
\mathscr{F}_r &= \{\texttt{box}_{\varepsilon,\texttt{Box}}, \texttt{printer_A}_{\varepsilon,\texttt{Printer}}, \texttt{desk}_{\varepsilon,\texttt{Place}}\} \quad \mathscr{F}_f = \{\} \\
\mathscr{V} &= \{p_{\texttt{Printer}}, b_{\texttt{Box}}, l_{\texttt{Place}}\} \\
\mathscr{T}_p &= \{\texttt{open}_{\texttt{Box}}, \texttt{unpack}_{\texttt{ThingBoxPlace}}\} \quad \mathscr{T}_c = \{\} \\
\mathscr{E} &= \{+\texttt{Open}_{\texttt{Box}}, -\texttt{Open}_{\texttt{Box}}, +\texttt{Closed}_{\texttt{Box}}, \ldots\}
\end{aligned}$$

This is the vocabulary for modelling delivery situations and actions. The intended meaning of the language components is obviously to express properties of boxes (`Open` and `Closed`) and spatial relationships between objects (`On` and `In`).

It immediately occurs to the reader that an adequate specification of spatial relationships requires things to be related on a more abstract level: in contrary to a language that provides a binary relation `On` for every possible object-sort combination, a more concise representation is always preferable.[1] We would therefore like to express that *things* are located in places and subsume, for example, printers and boxes. The $\leq$ relation induces such a sort hierarchy by defining the sorts in $\mathscr{Z}$ to be super-, respectively sub-sorts of each other. The formal background will be given below, for now $Z \leq Z'$ basically means, that every individual of sub-sort Z is also a member of super-sort Z'. In sorted logics terms and quantifiers carry information about the intended scope and denotation. Order-sorted logics extend this principle to a hierarchy of sorts, which allows for a more adequate and concise formalizations (and which supports efficient reasoning by reducing the space of logically sound alternatives).

In the previous example, we specified $\leq$ to contain the two sort relationships `Printer` $\leq$ `Thing` and `Box` $\leq$ `Thing`. This definition induces a sort hierarchy in which printers and boxes are disjunct concepts that are both *things*. Please note that all our examples abbreviate the notation of the sort hierarchy; it is always the transitive reflexive closure of the given relations. Consequently, a sort hierarchy that defines cyclic sub-sort relationships interprets all involved sorts as being identical. Furthermore, we have to stress that there is no restriction on the structure of a sort hierarchy; it allows in particular for the definition of multiple super-sorts. We would also like to point out that our interpretation of sorts and their hierarchies appears to have strong similarities with description logics and their inclusion relation $\sqsubseteq$, it should however not be mistaken for a terminological concept definition, for it has different semantics. Concerning our example above, the existence of the the sort `Thing` is merely a matter of convenience and not a semantic correspondence to the most general concept $\top$ or its counterparts in ontology languages.

As it is usual for order-sorted logics, the *well-sorted terms* over a language $\mathscr{L}$ are defined recursively by the following rules:

- A variable $v \in \mathscr{V}_Z$ is well-sorted with respect to sort $Z \in \mathscr{Z}$.
- If τ is a well-sorted term of sort $Z_1 \in \mathscr{Z}$ and there exists another sort $Z_2 \in \mathscr{Z}$ such that $Z_1 \neq Z_2$ and $Z_1 \leq Z_2$, then τ is a well-sorted term of (super-) sort Z_2.
- If $f \in \mathscr{F}_{Z_1 \ldots Z_n, Z'}$ and $\tau_1, \ldots, \tau_n$ are well-sorted terms of sorts $Z_1 \ldots Z_n$, then the function expression $f(\tau_1, \ldots, \tau_n)$ is a well-sorted term of sort Z'.

Ground terms are terms without variables. In the peripheral devices example, the ground term `printer_A` is of sort `Printer` and consequently also implicitly of sort `Thing`.

The definition of well-sortedness includes a subtle implication concerning the relationship of sorts: In the above paragraphs we made no clear distinction between sort symbols and sorts themselves; in fact, the definition of sorts will be given in the next section, but their intended notion of being a type that represents a specific set of objects is obvious. However, from a formal point of view, all constructs defined so far are annotated with *exactly one* sort signature, which means, that every term can be of exactly one (most specific) sort. Therefore, all sorts that are not in a sub-sort relationship or that do not have a common sub-sort are *disjoint* by definition.

Well-sorted formulae over a language $\mathscr{L}$ are defined as usual:

- The truth values $\top$ and $\bot$ are trivially well-sorted formulae.
- For a relation symbol $R \in \mathscr{R}_{Z_1 \ldots Z_n}$ and a set of well-sorted terms $\tau_1, \ldots, \tau_n$ of sorts $Z_1 \ldots Z_n$, the formula $R(\tau_1, \ldots, \tau_n)$ is a well-sorted formula. It is conventionally called an *atom*. *Ground atoms* are atoms in which all terms are ground terms.
- Given two well-sorted formulae φ and ψ, then $\neg\varphi$ and $\varphi \wedge \psi$ are well-sorted formulae.
- Given a well-sorted formula φ and a variable $v \in \mathscr{V}_Z$ that is free in φ, then $\forall v.\varphi$ is a well-sorted formula.

The remaining commonly used logical operators are defined as the usual shorthand notations:

[1] In fact, most planning systems never employed any type system and relied on respective predicates to indicate object classes. Even the latest version of the International Planning Competition standard language PDDL only supports such a "simulated" sort system, which is however optional (as is the declaration of predicates) [111]. A first step into more complex object typing is given in the PDDL-successor candidate OPT, which allows for a type hierarchy [189].

- $\varphi \vee \psi = \neg(\varphi \wedge \psi)$
- $\varphi \Rightarrow \psi = \neg\varphi \vee \psi$
- $\exists v.\varphi = \neg(\forall v.\neg\varphi)$
- ... and so on.

We also use the standard notation $\forall v_1, v_2 \varphi$ instead of $\forall v_1.\forall v_2.\varphi$ and do so as well for existentially quantified formulae. As it is common in logic terminology, we refer to atoms and negated atoms as *positive* and *negative literals*.

Regarding the example scenario, it is easy to see that the ground atom $\texttt{On}(\texttt{box}, \texttt{desk})$ is a well-sorted literal since the constant `box` is of sort `Box` with $\texttt{Box} \leq \texttt{Thing}$ and constant `desk` is of sort `Place`, which matches the signature of the atom's relation symbol $\texttt{On}_{\texttt{ThingPlace}} \in \mathscr{R}_f$.

The above definitions constitute a fragment of predicate logic that provides all necessary building blocks for describing world states and changes to world states.

2.2 States and State Abstractions

Our planning approach follows a state-based view on planning, which means, that the central means of representing the real-world's dynamics is a synopsis of specific attributes and conditions, expressed in our logical language: a *world state* or *situation*. As we will see later, the execution of actions induces changes to the world state, which makes state-based planning a reasoning process about state-transition systems. In order to express the changes effected by a state transition, we use the flexible symbols that are provided by our planning language. Consequently, we introduce states as interpretations of the flexible symbols.

It is worth noting that we adopt McDermott's notion of states as *chronicles* [185, 186], which provides a subtle contrast to the view of states as "facts about the execution environment at a given point in time". The key difference is that chronicles describe what is true in the world, what was true, and what will be true, provided no further actions are initiated. A state transition, according to the chronicle metaphor, therefore corresponds to the common-sense notion of changing the future course of events without being able to access the past. This persistence quality of states will be picked up in the action-dedicated sections below.

Fig. 2.1 illustrates the construction and usage of states: In the peripheral device delivery example the model focuses on the aspect that the box is closed, that it is standing on the customer's desk, and that it is containing a printer. The model thereby deliberately ignores properties of the box and the printer, for example, the box's inscription, the printer's color, and the like. The model of the *open* action describes the state transition induced by executing the action in terms of the relevant state features. In the example, performing the *open* action opens the box without changing the box's content or position. The chronicle view interprets this state as follows: everything that is true is either true from the beginning (the location of the box) or has been changed during the course of action (the box is now open). In order to provide a referential frame to the states, we begin with introducing the logical model for interpreting the symbols in a state description.

For a given logical language $\mathscr{L}$ a *model* denotes a structure $\mathscr{M} = \langle \mathbb{D}, \mathscr{I} \rangle$. The first element $\mathbb{D}$ is a $\mathscr{Z}$-indexed family of finite *carrier sets* or *domains*. For every sort symbol $Z \in \mathscr{Z}$ there exists an associated non-empty set of objects $\mathbb{D}_Z$ representing the respective domain. Please note that in most cases when we refer to "the sort Z", we mean the carrier set $\mathbb{D}_Z$ or the concept that constitutes it. The sort hierarchy that is induced by the $\leq$ component of the language is reflected in the following domain relationships:

$$\forall Z, Z' \in \mathscr{Z} : Z \leq Z' \Rightarrow \mathbb{D}_Z \subseteq \mathbb{D}_{Z'}$$

$\mathscr{I}$ is a state-independent *interpretation* that assigns functions and relations of appropriate type to their rigid symbols in $\mathscr{L}$. That means, that every $f \in \mathscr{F}_{r_{Z_1 \ldots Z_n, Z'}}$ is interpreted by a respective function $f^{\mathscr{I}} : Z_1 \times \ldots \times Z_n \rightarrow Z'$ and every $R \in \mathscr{R}_{r_{Z_1 \ldots Z_n}}$ by a specific relation $R^{\mathscr{I}} \subseteq Z_1 \times \ldots \times Z_n$. Analogously, for a given

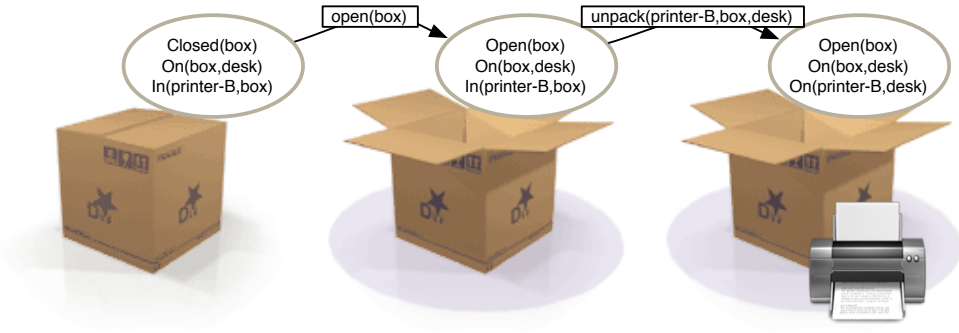


Figure 2.1: States and state transitions as abstractions of the real world in the computer peripheral device scenario. A cardboard box is delivered, which contains a printer. The first action is to open the box and after that to take out the printer and place it on the customer's desk.

model $\mathscr{M}$ a state s is an interpretation of the flexible relation and functions symbols over $\mathbb{D}$. The set of all states over a language $\mathscr{L}$ and a model $\mathscr{M}$ is denoted by $\mathscr{S}$.

In order to evaluate terms and formulae containing variables, we introduce a sort preserving *valuation* $\beta : \mathscr{V}_{\mathscr{Z}} \to \mathbb{D}_{\mathscr{Z}}$. This function provides a mapping from variables onto domain objects. We use $\beta[v/d]$ with variable $v \in \mathscr{V}$ and carrier set $d \in \mathbb{D}$ for the valuation β' that satisfies $\beta =_v \beta'$ (β and β' agree on all arguments except possibly v) and $\beta'(v) = d$.

Now we have the instruments at our fingertips for giving a meaning to terms. Given a logical language $\mathscr{L}$ and a model $\mathscr{M} = \langle \mathbb{D}, \mathscr{I} \rangle$, a well-sorted term τ is evaluated in a state s under a valuation β, written as $[\![\tau]\!]_{s,\beta}$, according to the following recursive definition:

$$[\![v]\!]_{s,\beta} = \beta(v) \text{ for } v \in \mathscr{V}$$

$$[\![f(\tau_1,\ldots,\tau_n)]\!]_{s,\beta} = \begin{cases} f^{\mathscr{I}}([\![\tau_1]\!]_{s,\beta},\ldots,[\![\tau_n]\!]_{s,\beta}) & \text{for } f \in \mathscr{F}_r \\ f^{s}([\![\tau_1]\!]_{s,\beta},\ldots,[\![\tau_n]\!]_{s,\beta}) & \text{for } f \in \mathscr{F}_f \end{cases}$$

Please note that we restrict the set of states $\mathscr{S}$ to those, which assign finite relations to the symbols in $\mathscr{R}_f$. Since our carrier sets are finite as well, this implies that our models are *natural* ones [247].

The state *features* depicted in Fig. 2.1 suggest a world-state characterization that is based on atoms like `Closed(box)`. In fact, the representation of states as sets of positive literals like it has been introduced by the STRIPS formalism [92] is very common in the AI planning area to date (planning graph based systems [28], heuristic state-space planners [136], etc.). Every atom that is explicitly stated in the situation's set is a fact that "holds in the state", while every possible other atom implicitly does not hold – the so-called *closed world assumption*. Contrary to this set-based semantics, our formalism will use *formulae* to express facts about the world states, and a given state will consequently either *satisfy* a given formula or it will not. This representation is clearly more expressive and will allow for a more elegant formalization of actions, etc. The example in Fig. 2.1 thus has to be read as "the conjunction of the given state features holds in that state".

The following definition specifies the satisfaction relation between a formula φ and a state s. For a given language $\mathscr{L}$ and a model $\mathscr{M} = \langle \mathbb{D}, \mathscr{I} \rangle$, a well-sorted formula φ is *valid* (invalid) in a state s under a valuation β,

denoted by $s \models_{\mathscr{M},\beta} \varphi$ (respectively $s \not\models_{\mathscr{M},\beta} \varphi$) according to the following definition:

$$
\begin{aligned}
s &\models_{\mathscr{M},\beta} \top \\
s &\not\models_{\mathscr{M},\beta} \bot \\
s \models_{\mathscr{M},\beta} (\tau_1 \equiv_Z \tau_2) &\text{ if and only if } [\![\tau_1]\!]_{s,\beta} = [\![\tau_2]\!]_{s,\beta} \\
s \models_{\mathscr{M},\beta} R(\tau_1,\ldots,\tau_n) &\text{ if and only if } \begin{cases} \langle [\![\tau_1]\!]_{s,\beta},\ldots,[\![\tau_n]\!]_{s,\beta}\rangle \in R^{\mathscr{I}} & \text{for } R \in \mathscr{R}_r \\ \langle [\![\tau_1]\!]_{s,\beta},\ldots,[\![\tau_n]\!]_{s,\beta}\rangle \in R^{s} & \text{for } R \in \mathscr{R}_f \end{cases} \\
s \models_{\mathscr{M},\beta} \varphi \wedge \psi &\text{ if and only if } s \models_{\mathscr{M},\beta} \varphi \text{ and } s \models_{\mathscr{M},\beta} \psi \\
s \models_{\mathscr{M},\beta} \neg\varphi &\text{ if and only if } s \not\models_{\mathscr{M},\beta} \varphi \\
s \models_{\mathscr{M},\beta} \forall v.\varphi &\text{ if and only if } s \models_{\mathscr{M},\beta[v/d]} \varphi \\
&\quad \text{for every element } d \text{ in the appropriate domain } \mathbb{D}_Z \text{ for variable } v \in \mathscr{V}_Z
\end{aligned}
$$

The converse relationship can be used to *characterize* a set of states by a formula φ, that means those states in which the formula is valid for a given model under a given valuation: $S_\varphi = \{s | s \models_{\mathscr{M},\beta} \varphi\}$.

Since we want to support a practical way of building domain models as well as an efficient plan generation process, an *abstraction* of states that is based on non-logical axioms lends itself for describing real-world situations in a more generalizing manner (cf. Sec. 1.1.3). In the simple running example of Fig. 2.1, it would be a very natural thing to combine the state features `On(box,desk)` and `In(printer_A,box)` into one atomic feature `Delivered(printer_A)`. Detailed information about the identity of the actual box and whether it is currently closed is generally not relevant. The less abstract state description becomes useful only when our reasoning process starts focusing on the unpacking action.

Such a technique of combining features can also be referred to as *knowledge transformation* and only few approaches exist[2] that adopt it. A proper formal integration is essential in order to ensure properties such as executability of plans, etc. These issues will be addressed in later sections, but for motivating the upcoming definitions and design choices, the basic ideas shall be sketched here: Generally speaking, we need to provide criteria that can guarantee that an abstract state description captures the relevant facets of the detailed state model in a coherent way. Then, and only then, a plan that works on the most concrete level of detail provably implements an abstract plan that has been formulated in terms of abstract state features, and eventually this will be used to show that the plan solves a problem specification. In the worst case, not properly grounded knowledge transformations intrinsically permit contradictory state descriptions, hence the modeler may introduce unnoticed inconsistencies due to which no executable plan can ever be developed. We will thus introduce (non-logical) *state-abstraction axioms* for specifying atomic feature abstractions, which is a concept that very naturally blends in our formal framework.

Definition 2.1 (State-Abstraction Axioms)**.** Given a language $\mathscr{L}$ and model $\mathscr{M}$, a *state-abstraction axiom* δ relates an atom with a disjunction of possible refinements. Let such an abstract atom's symbol be $R \in \mathscr{R}_{Z_1\ldots Z_k}$. Let furthermore $v_1,\ldots,v_l$ be the variables occuring in the atom's argument terms $\tau_1,\ldots,\tau_k$ and $w^1_{1_1},\ldots,w^m_{l_{n_m}}$ those of the argument terms of respective atoms $R^1_1,\ldots,R^m_{n_m}$. A state-abstraction axiom for an atom over R is then defined as the following well-sorted formula:

$$
\begin{aligned}
\delta = \forall v_1 \ldots \forall v_l.\exists w^1_{1_1} \ldots \exists w^m_{l_{n_m}}.R(\tau_1,\ldots,\tau_k) \Leftrightarrow & R^1_1(\tau^1_{1_1},\ldots,\tau^1_{k_1}) \wedge \ldots \wedge R^1_{n_1}(\tau^1_{1_{n_1}},\ldots,\tau^1_{k_{n_1}}) \\
& \vee \quad \ldots \quad \vee \\
& R^m_1(\tau^m_{1_1},\ldots,\tau^m_{k_1}) \wedge \ldots \wedge R^m_{n_m}(\tau^m_{1_{n_m}},\ldots,\tau^m_{k_{n_m}})
\end{aligned}
$$

•

The state-abstraction axiom can be read as follows: an abstract state feature over the relation symbol R has m possible concretizations, each of which is conjunctively composed of n_m (more concrete) state features. In order to narrow down intended variable assignments, these axioms can also model a variable to be "cast"

[2] To our knowledge, there is only one piece of work dealing with knowledge transformation, basically by a literal translation mechanism [44]. Not surprisingly, it is also a hybrid planning approach, the HYBIS system (see Sec. 1.1.4).

into a sub-sort. For this reason, the refinements in state-abstraction axioms typically contain additional equations such that the universally quantified variables in the abstract atom are co-designated with some of the existentially qualified variables in the disjuncts that are more specific. We will refer to the disjuncts also as the *possible refinements* for the abstract atom. Each possible refinement is sufficient for the abstraction to hold, which is the usual notion of concrete facts ("⇐"). Since we assume our abstraction model to be a total definition of all permitted alternatives, the axiom is also defined as an implication from the abstract into the concrete ("⇒").

A set of state-abstraction actions consequently induces a hierarchical relationship between state features and their possible refinements and thus also a hierarchical relationship between formulae containing the atoms. The following definition conveys this notion.

Definition 2.2 (Refinements of Formulae)**.** Let Δ be a set of state-abstraction axioms. A formula φ' is called a *refinement* of a (more abstract) formula φ with respect to Δ if and only if φ' entails φ according to the axiom set: $\Delta \cup \{\varphi'\} \models \varphi$. •

The refinement relationship between formulae imposes a hierarchy, and since formulae are used to characterize states, Δ is therefore also called to induce a *state feature hierarchy*.

The computer peripheral delivery scenario of our running example incorporates abstract state descriptions by providing three possible refinements for an abstract printer delivery: the first has been explained before and generalizes from a printer being packaged in a cardboard box that has arrived at the customer site. Alternative refinements may be the following: the printer is a big laser copier, which is personally transported by its vendor and received by an employee, or the printer is a small office model that is bundled to a personal computer. To represent these concepts, we extend the example language $\mathscr{L}_{devices}$ in the following way:

$$
\begin{aligned}
\mathscr{Z} =& \{\texttt{Thing}, \texttt{Printer}, \texttt{Box}, \texttt{Place},\\
& \texttt{LaserCopier}, \texttt{Desk}, \texttt{Person}, \texttt{Vendor}, \texttt{Employee}, \texttt{OfficeModel}, \texttt{PC}\}\\
\leq =& \{(\texttt{Printer}, \texttt{Thing}), (\texttt{Box}, \texttt{Thing}), (\texttt{Desk}, \texttt{Place}), (\texttt{PC}, \texttt{Thing}), (\texttt{LaserCopier}, \texttt{Printer})\\
& (\texttt{OfficeModel}, \texttt{Printer}), (\texttt{Vendor}, \texttt{Person}), (\texttt{Employee}, \texttt{Person})\}\\
\mathscr{R}_r =& \{\texttt{Sells}_{\texttt{VendorThing}}, \texttt{Bundeled}_{\texttt{ThingPC}}\}\\
\mathscr{R}_f =& \{\texttt{Open}_{\texttt{Box}}, \texttt{Closed}_{\texttt{Box}}, \texttt{On}_{\texttt{ThingPlace}}, \texttt{In}_{\texttt{ThingBox}}, \texttt{Delivered}_{\texttt{Thing}}, \texttt{Running}_{\texttt{Thing}},\\
& \texttt{Connected}_{\texttt{ThingThing}}, \texttt{Carries}_{\texttt{PersonThing}}, \texttt{Received}_{\texttt{ThingPerson}}, \texttt{Installed}_{\texttt{PCPlace}}\}\\
\mathscr{F}_r =& \{\texttt{box}_{\varepsilon,\texttt{Box}}, \texttt{printer\_A}_{\varepsilon,\texttt{LaserCopier}}, \texttt{printer\_B}_{\varepsilon,\texttt{OfficeModel}}, \texttt{desk}_{\varepsilon,\texttt{Place}}, \texttt{PC\_5}_{\varepsilon,\texttt{PC}},\\
& \texttt{employee\_Smith}_{\texttt{Employee},\varepsilon}, \texttt{vendor\_X}_{\texttt{Vendor},\varepsilon}, \ldots\}\\
\mathscr{F}_f =& \{\}\\
\mathscr{V} =& \{p_{\texttt{Printer}}, b_{\texttt{Box}}, l_{\texttt{Place}}, \ldots\}\\
\mathscr{T}_p =& \{\texttt{open}_{\texttt{Box}}, \texttt{unpack}_{\texttt{ThingBoxPlace}}\} \qquad \mathscr{T}_c = \{\texttt{prepare}_{\texttt{Box}}\} \qquad \mathscr{E} = \{\ldots\}
\end{aligned}
$$

The adapted sort hierarchy $\leq_{devices}$ is also depicted in Fig. 2.2.

A state abstraction $\delta \in \Delta_{devices}$ for describing the alternative refinements given above can be specified with the extended language by the following state-abstraction axiom:

$$
\begin{aligned}
& \forall p_{\texttt{Printer}}\, \exists b_{\texttt{Box}}, d_{\texttt{Desk}}, l_{\texttt{LaserCopier}}, v_{\texttt{Vendor}}, e_{\texttt{Employee}}, o_{\texttt{OfficeModel}}, pc_{\texttt{PC}}\\
\texttt{Delivered}(p) \Leftrightarrow\ & (\texttt{On}(b,d) \wedge \texttt{In}(p,b)) \vee\\
& (l \equiv p \wedge \texttt{Sells}(v,l) \wedge \texttt{Carries}(v,l) \wedge \texttt{Received}(l,e)) \vee\\
& (o \equiv p \wedge \texttt{Bundeled}(o,pc) \wedge \texttt{Installed}(pc,d))
\end{aligned}
$$

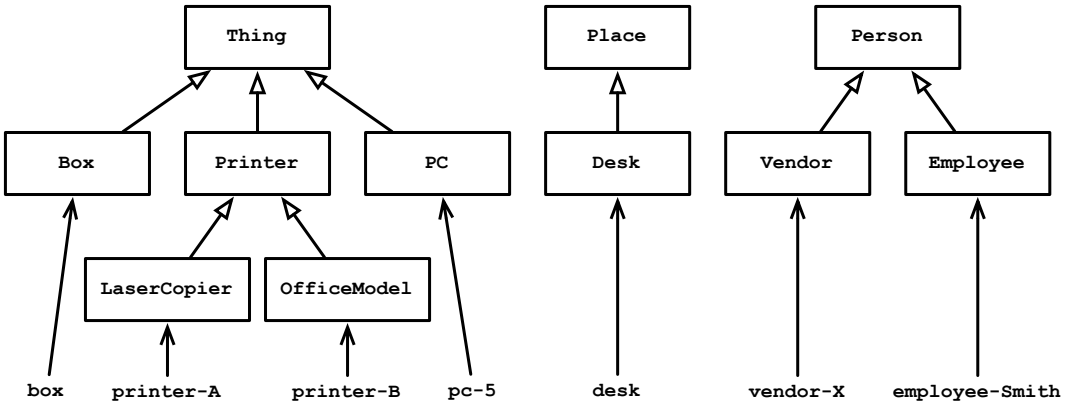


Figure 2.2: The extended sort hierarchy with the associated rigid constants (objects) of $\mathscr{L}_{devices}$.

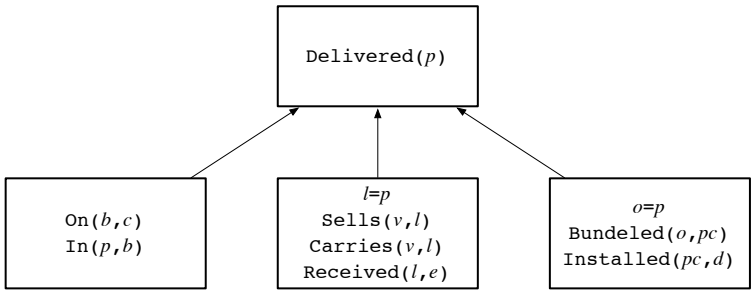


Figure 2.3: A state feature hierarchy induced by $\Delta_{devices}$.

Please note the use of equations in the abstraction axiom: the first conjunct in the last disjunct, for example, co-designates the universally bound printer variable p with that of the existentially bound office model o. This equation achieves a sort conversion of the abstract feature's argument and is the technical way of specifying that the refinement applies for instances of specific sub-sorts (also known as a variable "cast"). An informal graphical representation of the state feature hierarchy that is induced by the example axiom is depicted in Fig. 2.3. Every box displays some state features that are valid in the respective state.

In practice, most of the alternatives that are modelled in state-abstraction axioms will exclude each other. Since any printer is, for instance, either a laser copier or an office ink model, the possible refinements of an abstract state will be disjoint sets of states: there will be no state in which a laser copier is sold in a bundle with a PC. Such a mutual exclusion is however not necessarily given, and the refinement states can occasionally cover several possible refinements of an abstract state "simultaneously".

As it has been stated above, consistency concepts become necessary in order to provide meaningful and sound state-abstraction hierarchies. The following notion of state-abstraction consistency captures the logical consistency of the axioms together with a well-formedness criterion for concretizations.

Definition 2.3 (Consistency of State-Abstraction Axioms)**.** A set of state-abstraction axioms Δ over a language $\mathscr{L}$ and model $\mathscr{M}$ is called *consistent* if and only if the following conditions hold:

1. Δ is satisfiable and
2. every axiom $\delta \in \Delta$ for an atom over a rigid relation symbol $R \in \mathscr{R}_r$ contains in every refinement only atoms with rigid relation symbols.

•

The consistency criteria address crucial aspects of state-abstraction axiom definitions: The first criterion ensures that the axioms are logically consistent, that means, there exists a model satisfying the sentences in Δ. This is an essential requirement for meaningful models of abstraction, since inconsistent axiom sets trivially support arbitrary refinements, including unintended absurd ones (ex falso sequitur quod libet). The second criterion ascertains state-refinements to adhere to a notion of well-formedness; when using consistent state-abstraction axioms for "unfolding" abstract state descriptions, rigid and flexible state features do never alternate. Anticipating the sections below dedicated to the representation of actions, we note that the interpretation of flexible atoms may change from state to state while that of rigid ones does not. In other words, flexible symbols characterize *dynamic* world-state features, and rigid symbols mark *static* world-state features. While it is quite intuitive to abstract multiple rigid state features in one summarizing rigid feature and analogously to do so for flexible features, mixtures of rigid and flexible atoms have to be examined more deeply. On the one hand, it makes perfectly sense to build a flexible abstraction over combinations of rigid and flexible features because the notion of *change*[3] is an abstraction of *stasis* (consider stasis to be represented by a constant change from the same into the same). Our previous example abstraction axiom uses the abstract atom over `Delivered` in one refinement for aggregating a static atom `Sells` (assuming that the vendor is continuously offering the desired kind of printer during our planning horizon) and several flexible ones (the exemplary employee who `Received` the printer in that very state will probably not do so all day long). To put it short: concrete change can imply abstract change under static concrete conditions.

The opposite scenario, that means, to apply flexible state features as refinements of abstract static features, however introduces a paradox: reasoning about such axioms would derive rigid state features from flexible ones. This implied that some rigid features would change at the moment some flexible do and therefore these rigid atoms would become flexible for themselves. This contradiction propagates into our semantics for plans (which will be presented below), which basically states that concrete plans are logical justifications for abstract plan specifications. The reason for abstraction lies in having a reduced model of the world that consistently behaves on its abstract level in correspondence to the way the refined one does on the concrete level. Finding however a concrete plan such that its abstraction uses rigid state features that do not hold consistently, is a contradiction in its own and that concrete plan should have never been allowed to be developed.

Now that we can represent and explain the meaning of a state, we are ready to deal with state transitions, which describe the dynamics of the world.

2.3 From State Transitions to Actions

In Artificial Intelligence planning, actions are the representations of the dynamics of the real world. Recall the exemplary course of action depicted in Fig. 2.1: there is an agent performing actions that first opened the box and then emptied it. This kind of "atomic activities" is going to be called *operators* and we will define them in terms of state transitions that are induced by elementary operations. But we will also investigate abstract forms of actions: so-called *complex tasks* stand for sets of sequences of operators. In our running example, opening the box and unpacking the printer might be part of a larger, more abstract "setting-up" activity. This section first defines the structure of our action representation and shows what it means for an action to be executable and how to determine what world states are going to look like after executing an action. Based on these semantic considerations, it will finally provide a notion of consistency in order to define meaningful action models only.

We will begin with atomic *elementary operations* that represent changes to the interpretation of flexible symbols. These elementary operations are the atomic building blocks for the semantics of actions: This means primitive tasks as well as abstract ones.

In order to characterize a change in the interpretation of a flexible symbol, we introduce for each flexible relation symbol $R \in \mathscr{R}_{f_{Z_1 \ldots Z_n}}$ of a given language and model the following two *relation update-functions*

[3] This more than 25 centuries old discussion is clearly out of the scope of this thesis. However, Heraclitus' view of stasis as a mere illusion of unnoticed change – the only constant – conceptually matches our state-based modelling view. "Potauoisi dis toisi autoisi ouk an embaies etera gar <kai etera> epirreei udata": You cannot step twice into the same river.

$d{-}R : \mathbb{D}_{Z_1} \times \ldots \times \mathbb{D}_{Z_n} \rightarrow \mathscr{S} \times \mathscr{S}$ and $a{-}R : \mathbb{D}_{Z_1} \times \ldots \times \mathbb{D}_{Z_n} \rightarrow \mathscr{S} \times \mathscr{S}$ with

$$s\ d{-}R(d_1,\ldots,d_n)\ s' \text{ if and only if } R^{s'} = R^s \setminus \{(d_1,...,d_n)\} \text{ and } R'^{s'} = R'^s \text{ for } R' \neq R$$
$$\text{as well as } f^s = f^{s'} \text{ for any } f \in \mathscr{F}$$
$$s\ a{-}R(d_1,\ldots,d_n)\ s' \text{ if and only if } R^{s'} = R^s \cup \{(d_1,...,d_n)\} \text{ and } R'^{s'} = R'^s \text{ for } R' \neq R$$
$$\text{as well as } f^s = f^{s'} \text{ for any } f \in \mathscr{F}$$

For each flexible function symbol $f \in \mathscr{F}_{f_{Z,Z_1\ldots Z_n}}$ we provide a so-called *term update-function* $u{-}f : \mathbb{D}_{Z_1} \times \ldots \times \mathbb{D}_{Z_n} \times \mathbb{D}_Z \rightarrow \mathscr{S} \times \mathscr{S}$ with

$$s\ u{-}f(d_1,\ldots,d_n,d)\ s' \text{ if and only if } f^{s'} = f^s_{[(d_1,\ldots,d_n)\leftarrow d]} \text{ and } f'^{s'} = f'^s \text{ for } f' \neq f$$
$$\text{as well as } R^s = R^{s'} \text{ for any } R \in \mathscr{R}$$

The expression $f^s_{[(d_1,\ldots,d_n)\leftarrow d]}$ thereby denotes an interpretation of function f that agrees with f^s except for the arguments $d_1,\ldots,d_n$, in which case f evaluates to d. As we employ natural models, there exists for any two states s and s' a finite sequence of relation and term update-functions $u_1,\ldots,u_n$ such that $s\ u_1 \circ \ldots \circ u_n\ s'$, where $\circ$ denotes functional composition [247].

Based on the definitions of the $a{-}R$, $d{-}R$, and $u{-}f$ update functions on states, we can now define the semantics of elementary operations $\mathscr{E}$ of our planning language as follows. Given a language $\mathscr{L}$, a model $\mathscr{M}$ and a valuation β, a pair of states $\langle s,s'\rangle$ satisfies elementary operations $+R(\tau_1,\ldots,\tau_n)$, $-R(\tau_1,\ldots,\tau_n)$, and $:=f(\tau_1,\ldots,\tau_n,\tau)$ according to the following definition:

$$\langle s,s'\rangle \models_{\mathscr{M},\beta} +R(\tau_1,\ldots,\tau_n) \text{ if and only if } s\ a{-}R([\![\tau_1]\!]_{s,\beta},\ldots,[\![\tau_n]\!]_{s,\beta})\ s'$$
$$\langle s,s'\rangle \models_{\mathscr{M},\beta} -R(\tau_1,\ldots,\tau_n) \text{ if and only if } s\ d{-}R([\![\tau_1]\!]_{s,\beta},\ldots,[\![\tau_n]\!]_{s,\beta})\ s'$$
$$\langle s,s'\rangle \models_{\mathscr{M},\beta} :=f(\tau_1,\ldots,\tau_n,\tau) \text{ if and only if } s\ u{-}f([\![\tau_1]\!]_{s,\beta},\ldots,[\![\tau_n]\!]_{s,\beta},[\![\tau]\!]_{s,\beta})\ s'$$

This means, elementary operations represent single, atomic state transitions that change the symbol interpretation in one specific argument and leave all other aspects unmodified. We finally adopt from [247] the concept of *weakest preconditions* (*wp*) with respect to elementary operations. Let φ be a formula which contains only variables that are distinct from those occurring in $\tau_1,\ldots,\tau_n$.

- The weakest precondition of φ with respect to the elementary addition $+R(\tau_1,\ldots,\tau_n)$ is the formula resulting from φ in which all atomic sub-formulae $R(\sigma_1,\ldots,\sigma_n)$ are replaced by $[(\tau_1 \neq \sigma_1 \vee \ldots \vee \tau_n \neq \sigma_n) \Rightarrow R(\sigma_1,\ldots,\sigma_n)]$.
- $wp(\varphi, -R(\tau_1,\ldots,\tau_n))$ results from φ by replacing all atomic sub-formulae $R(\sigma_1,\ldots,\sigma_n)$ by $[R(\sigma_1,\ldots,\sigma_n) \wedge (\tau_1 \neq \sigma_1 \vee \ldots \vee \tau_n \neq \sigma_n)]$.
- Regarding the term update, $wp(\varphi, :=f(\tau_1,\ldots,\tau_n,\tau))$ is obtained from φ by replacing all occurrences of term $f(\tau_1,\ldots,\tau_n)$ by τ.

The weakest precondition of a formula φ with respect to a *sequence* of elementary operations $e_1 \ldots e_n$ is defined recursively as $wp(\ldots wp(wp(\varphi,e_1),e_2)\ldots,e_n)$.

Building on the above state transition semantics, the following definition introduces the first type of actions that is supported by our framework.

Definition 2.4 (Operator)**.** Given a language $\mathscr{L} = \langle \mathscr{Z}, \leq, \mathscr{R}_r, \mathscr{R}_f, \mathscr{F}_r, \mathscr{F}_f, \mathscr{V}, \mathscr{T}_p, \mathscr{T}_c, \mathscr{E}\rangle$, an *operator* or *primitive task* schema is defined by a structure

$$o(\bar{v}) = \langle \text{prec}(o(\bar{v})), \text{eff}(o(\bar{v}))\rangle$$

where

- $o \in \mathscr{T}_p$ is an operator symbol,

- $\overline{v} = v_1, \ldots, v_n$ is a list of variables from $\mathscr{V}$, the so-called operator *parameters*, that matches the operator symbol's signature,
- $\text{prec}(o(\overline{v}))$ is a well-formed formula over $\mathscr{L}$ in which all free variable occurrences are elements in $\overline{v}$, and
- $\text{eff}(o(\overline{v})) = e_1 \ldots e_m$ is a non-empty, finite sequence of elementary operations over the operation symbols in $\mathscr{E}$. All variables occurring in $\text{eff}(o(\overline{v}))$ are from $\overline{v}$.

An *instance* of the operator schema is a copy of the schema where all parameters are substituted by new variables through a well-sorted variable replacement. •

For a given model $\mathscr{M}$ and a valuation β, an operator $o(\overline{v}) = \langle \text{prec}(o(\overline{v})), \text{eff}(o(\overline{v})) \rangle$ transforms a state s into a state s', denoted by $\langle s, s' \rangle \models_{\mathscr{M},\beta} o(\overline{v})$, if and only if the following two conditions hold:

1. $s \models_{\mathscr{M},\beta} \text{prec}(o(\overline{v}))$, that means, the operator is applicable in s, and
2. there exists a sequence of states $s_0, \ldots, s_m$ with $s = s_0$ and $s' = s_m$ such that $\langle s_{i-1}, s_i \rangle \models_{\mathscr{M},\beta} e_i$ for $1 \leq i \leq m$.

The second condition can also be formalized via the functional composition $s\ u_1 \circ \ldots \circ u_m\ s'$ where u_i is the corresponding update function to the elementary operation e_i for $1 \leq i \leq m$. This definition has the advantage that no intermediate states $s_1, \ldots, s_{m-1}$ have to be taken into account.

The operator *generates* a formula φ_{eff} over $\mathscr{L}$ if in addition $s \models_{\mathscr{M},\beta} wp(\varphi_{\text{eff}}, e_1 \ldots e_m)$.

Please note that our view of state transitions is based on a transformation of the states themselves and is therefore in the spirit of ADL action schemata [213], unlike the *formula transformation* approach of STRIPS operators [92, 169].

Since specific state features are exclusively subject to change by the respective elementary operations, all other features are consequently assumed to *persist*, that is to say, to be *invariant* against an operator's effects. This informal notion of feature persistence can be formally expressed with respect to formulae as follows. For a given model $\mathscr{M}$ and a valuation β, a formula φ is invariant against an operator $o(\overline{v}) = \langle \text{prec}(o(\overline{v})), \text{eff}(o(\overline{v})) \rangle$ if and only if for all pair of states s and s' with $\langle s, s' \rangle \models_{\mathscr{M},\beta} o(\overline{v})$ the following holds: if $s \models_{\mathscr{M},\beta} \varphi$ then $s' \models_{\mathscr{M},\beta} \varphi$.Invariance of a formula against a sequence of operators $o_1 \ldots o_n$ is consequently defined inductively as the invariance of the formula against every single operator over the progression of the sequence.

Finally, a formula φ is *generated* by a sequence $o_1 \ldots o_n$ of operators if and only if it is generated by some o_i $(1 \leq i \leq n)$ and is invariant against the (sub-) sequence $o_{i+1} \ldots o_n$.

Let us continue with the development of our running example and put some of the above definitions in practice. Please recall the two actions depicted in the delivery scenario in Fig. 2.1. Both are intended to be ground instances of the following primitive task schemata ($\texttt{open}_{\texttt{Box}}$ and $\texttt{unpack}_{\texttt{ThingBoxPlace}}$ are operator symbols in $\mathscr{T}_p$):

$$\begin{aligned} \texttt{open}(b) =& \langle \texttt{Closed}(b), +\texttt{Open}(b) - \texttt{Closed}(b) \rangle \\ \texttt{unpack}(p,b,l) =& \langle \texttt{Open}(b) \wedge \texttt{In}(p,b) \wedge \texttt{On}(b,l), \\ & +\texttt{On}(p,l) := \texttt{weight}(b, \texttt{weight}(b) - \texttt{weight}(p)) - \texttt{In}(p,b) \rangle \end{aligned}$$

These task schemata can be read as follows: Opening a box-like object can be performed in a given state s if the box is closed in that state. After the execution of an `open` action the world is in a state s' that describes exactly the same situation except for the interpretation of literal `Closed`, which does not hold for the given box anymore, and a state feature `Open` that became valid.

The unpacking action schema is defined with three parameters: the variable b stands for the box to unpack, the variables p and l are for grounding the operator with respect to the printer inside the box and with the location the box is currently standing on, respectively. After the printer p is unpacked, it is located next to the box in the position "on" the location instance l, and the box's weight is updated to the weight it had before the unpacking minus the unchanged weight of the printer (remember that term updates are evaluated in the state before the transition).

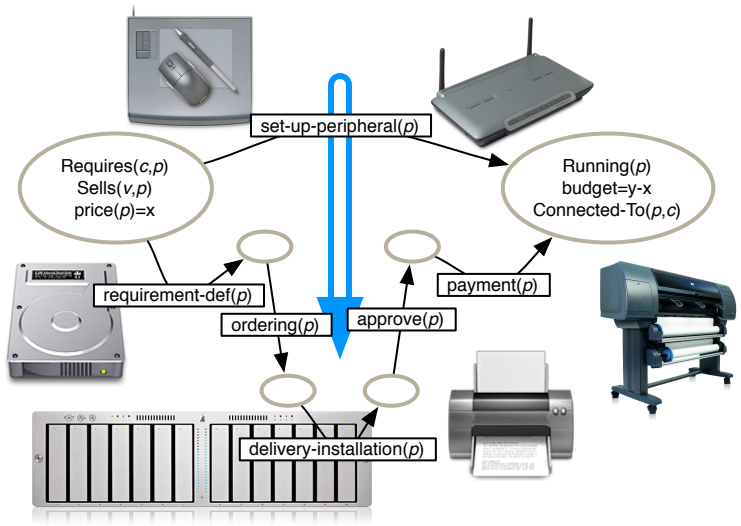


Figure 2.4: Task abstraction in the delivery scenario: setting up a computer peripheral, which is an abstraction of purchasing and installing printers, storage devices, and the like.

While operators describe those actions that are considered to be atomic, complex tasks are the means to aggregate such basic primitive actions into abstract processes and state transition specifications. They intuitively represent an abstraction of possibly multiple primitive courses of action. For example, the concrete procedure in the peripheral device delivery domain (see Fig. 2.1) may be part of the implementation of an abstract process. Let *setting up a computer peripheral* be an abstract action in that domain, and let it cover the purchase and installation routine for various kinds of hardware components (Fig. 2.4). If we imagine corresponding state abstractions, the abstract "set-up" task will subsume various courses of action by capturing an abstract specification of the states in which the subsumed actions are executable and for those states which they generate. We will give an example for that: Analogously to what has been illustrated for the state-abstraction axioms (see example after Def. 2.1), the set-up procedures start off in states that identify the computer to which the peripheral is to be connected, the vendor and price for the peripheral, the available budget, and the like. The common precondition can be formulated in an abstract way, namely that the computer meets the prerequisites for connecting to the peripheral, that the budget allows for the price of the peripheral, that the vendor actually has the peripheral in stock, etc. After executing any concrete course of action subsumed by this set-up task, the budget will be spent, the peripheral will be connected to the computer and running, and so on. Intuitively, there are also several layers in which the *task implementation* can be divided: in a first layer, the set-up consists of the requirement definition (determining which peripheral to deploy), the product ordering phase, the delivery and installation tasks, and finally putting the new peripheral into operation and processing the payment transaction. The second layer deals with the different types of defining the requirements, for example, the choice of a specific printer model depends on the supported operating systems, the volume of printed documents, and so forth. Also in the second layer, the ordering procedure can be made more concrete (on-line shopping, contracted vendor, ...) and the delivery and installation processes are planned. The latter may vary from simply placing the printer on a table and plugging in a cable to integrating a sophisticated hardware in a print shop setting.

The example exposes the nature of abstract actions as abstract state transitions: by addressing abstract state features, they use their preconditions and effects effectively as state specifications for concrete courses of action to start from and to end in. In other words, abstract actions provide input and output "gates" for implementing operator sequences, so abstract state transition means here to conceal the details of the transitioned states as well as the concrete number and configuration of intermediate states. We will see later how this technique accords with the previously described way of modelling in layers (cf. HTN planning in introductory Sec. 1.1.3 and the corresponding framework implementation in Sec. 3.2.3).

Let us now define the structure of abstract actions, which is similar to that of operators.

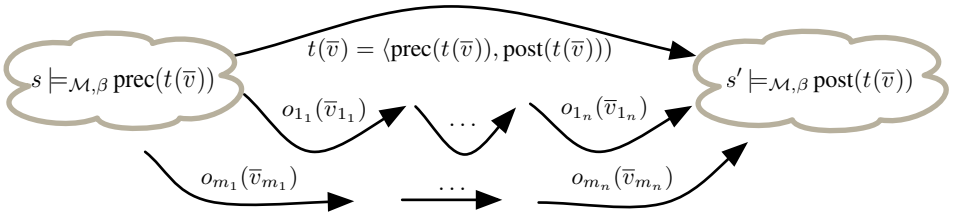


Figure 2.5: Implementations of a complex task.

Definition 2.5 (Complex Task)**.** Given a language $\mathscr{L} = \langle \mathscr{Z}, \leq, \mathscr{R}_r, \mathscr{R}_f, \mathscr{F}_r, \mathscr{F}_f, \mathscr{V}, \mathscr{T}_p, \mathscr{T}_c, \mathscr{E} \rangle$, a *complex task* is the structure

$$t(\overline{v}) = \langle \text{prec}(t(\overline{v})), \text{post}(t(\overline{v})) \rangle$$

where $t \in \mathscr{T}_c$ is a complex task symbol and $\overline{v} = v_1, \ldots, v_n$ is a list of variables from $\mathscr{V}$, also referred to as the task's *parameters*, that matches the task symbol's signature. The precondition $\text{prec}(t(\overline{v}))$ and postcondition $\text{post}(t(\overline{v}))$ are (well-formed) formulae over $\mathscr{L}$ and all free variables occuring in these formulae are taken from the parameter list.

An instance of the task schema is a copy of the schema where all parameters are substituted by new variables through a well-sorted variable replacement. •

We will use the term *abstract task* synonymously.

In the style of the respective definition for operators, for a given language $\mathscr{L}$ and model $\mathscr{M}$, a set of state-abstraction axioms Δ, and a valuation β, a complex task $t(\overline{v})$ transforms a state s into a state s', denoted by $\langle s, s' \rangle \models_{\mathscr{M},\beta} t(\overline{v})$ if and only if the following conditions hold:

1. $s \models_{\mathscr{M},\beta} \bigwedge_{\delta \in \Delta} \delta$ and $s' \models_{\mathscr{M},\beta} \bigwedge_{\delta \in \Delta} \delta$ ($s$and s' comply with the axioms)
2. $s \models_{\mathscr{M},\beta} \text{prec}(t(\overline{v}))$ (the task is applicable in state s)
3. $s' \models_{\mathscr{M},\beta} \text{post}(t(\overline{v}))$
4. There exists a finite sequence $s_1 \ldots s_n$ of states and a finite sequence $o_1 \ldots o_{n-1}$ of operators, where $s = s_1$, $s' = s_n$, $\langle s_i, s_{i+1} \rangle \models_{\mathscr{M},\beta} o_i$ for all $1 \leq i < n$ and $o_1 \ldots o_{n-1}$ generates a formula post$'$ such that $s_n \models_{\mathscr{M},\beta} (\text{post}' \Rightarrow \text{post}(t(\overline{v})))$. Every such operator sequence is called an *implementation* of the complex task.

Given a set of operators $O = \{o_1, \ldots, o_n\}$, the set of all implementations for a complex task schema $t(\overline{v})$ is the set of all operator sequences $o_1 \ldots o_m$ that are implementations for $t(\overline{v})$ over two states $s, s' \in \mathscr{S}$ for which $\langle s, s' \rangle \models_{\mathscr{M},\beta} t(\overline{v})$.

The complex task *generates* a formula post$''$ if and only if in addition $s_n \models_{\mathscr{M},\beta} (\text{post}(t(\overline{v})) \Rightarrow \text{post}'')$. Formula persistence and the treatment of complex task sequences are defined analogously to the corresponding cases for operators.

The above state transformation definition provides the formal foundation for our previous informal presentation: the semantics of abstract actions is based on the state transition sequences that are provided by the available primitive task schemata. Consequently, a complex task represents *all possible* concrete realizations of the specified state transition, and that means, it stands for a disjunctive set of operator sequences, depending on the defined operator schemata and abstraction axioms. This is illustrated in Fig. 2.5. Please note that according to the traditional representations for abstract actions in HTN planning, any operator sequence is trivially an implementation of a complex task without preconditions and effects.

Let us define a complex task for the running example. The abstract action with the task symbol $\texttt{prepare}_{\texttt{Thing}} \in \mathscr{T}_c$ is intended to represent all concrete courses of action that deal with the unpacking procedure of a delivered good. Let us furthermore assume, that $\mathscr{L}_{devices}$ contains a flexible relation symbol $\texttt{ReadyToConnect}_{\texttt{Thing}}$

in order to express that something is connectable to a computer or some other device.

$$\texttt{prepare}(p) = \langle \texttt{Delivered}(p), \texttt{ReadyToConnect}(p) \rangle$$

Let in addition to the previous axiom $\Delta_{devices}$ contain an appropriate state-abstraction axiom for relating the abstract state feature `ReadyToConnect` with at least the following refinement:

$$\forall t_{\texttt{Thing}} \exists p_{\texttt{Printer}}, d_{\texttt{Desk}} \texttt{ReadyToConnect}(t) \Leftrightarrow (t \equiv p \wedge \texttt{On}(t,d) \wedge \forall b_{\texttt{Box}} : \neg \texttt{In}(p,b)) \vee \ldots$$

The delivered good can be connected to another device if it is a printer, stands on a desk, and is not packaged in any box.

It is easy to see that the previously introduce opening and unpacking operators are an implementation of the above abstract task, since they perform precisely the specified state transitions.

Regarding the operator and task schema definitions, it is understandable that even this very accessible model that we have developed so far becomes increasingly complex and soon vulnerable to inconsistencies: What if the primitive task repertoire does not contain appropriate operators so that an item residing in a box cannot be placed `On` something? What if the `In` relation was rigid and therefore used wrongly in effect specifications? If the object referenced by p is `On` another object l, is there an abstraction axiom that (implicitly) prevents $\texttt{In}(p,b)$ or one that deduces it? It is an intrinsic characteristic of deductive planning approaches to enable reasoning about whether such action properties hold or not. Domain constraints for describing state invariants like security requirements can be directly validated [27]. Although our approach is not a deductive one, its logic foundations originate from there and it consequently allows for various profound model analysis techniques, including the above described invariance verification. While domain analysis as such is not in the focus of this thesis (cf. Sec. 7.2.3), we will now survey some notion of consistency and how this issue is addressed by our approach. Based on the presented semantics of action schemata, the following definition establishes formal criteria that characterize *meaningful* task schema specifications and in particular prevent the construction of task schema structures that produce inconsistent state changes.

Definition 2.6 (Consistency of Task Schemata)**.** This definition consists of two parts with the first one addressing operators and the second one abstract actions.

Given a language $\mathscr{L}$, a model $\mathscr{M}$, and a set of state-abstraction axioms Δ, a primitive task schema $o(\overline{v}) = \langle \text{prec}(o(\overline{v})), \text{eff}(o(\overline{v})) \rangle$ is called *consistent* if and only if the following conditions hold:

1. $\text{prec}(o(\overline{v}))$ must be satisfiable. If there does not exist a model for the operator's precondition, there will be no state in which the operator becomes applicable and, hence, the task cannot be carried out in the real world. This is most presumably a misconception of the application domain during the knowledge engineering process; at least, it is a useless task definition.
2. For all atoms $R(\overline{\tau})$ that occur in $\text{prec}(o(\overline{v}))$ or that correspond to a relation update operation in $\text{eff}(o(\overline{v}))$, the state feature hierarchy does not define a refinement, that means there is no axiom in Δ with
$$\forall v_1, \ldots, v_n \exists v'_1, \ldots, v'_k R(\overline{\tau}) \Leftrightarrow \varphi_1 \vee \ldots \vee \varphi_m$$
Operators exclusively use non-abstract state features in their preconditions and effects.
3. The effects are conflict-free, that means, $\forall e, e' \in \text{eff}(o(\overline{v}))$ the following holds:
$$\begin{array}{ll} \text{if } e = +R(\tau_1, \ldots, \tau_n) & \text{then } e' = -R(\tau'_1, \ldots, \tau'_n) \Rightarrow \tau_1 \neq \tau'_1 \vee \ldots \vee \tau_n \neq \tau'_n \text{ and} \\ \text{if } e = :=f(\tau_1, \ldots, \tau_n, \tau) & \text{then } e' = :=f(\tau'_1, \ldots, \tau'_n, \tau') \Rightarrow \tau_1 \neq \tau'_1 \vee \ldots \vee \tau_n \neq \tau'_n \end{array}$$
4. The induced state transition is in accordance with the state-abstraction axioms, that means, for any two states $s$and s' for which $\langle s, s' \rangle \models_{\mathscr{M},\beta} o(\overline{v})$ does hold, if $s \models_{\mathscr{M},\beta} \bigwedge_{\delta \in \Delta} \delta$ then $s' \models_{\mathscr{M},\beta} \bigwedge_{\delta \in \Delta} \delta$.

Given a language $\mathscr{L}$ and a set of state-abstraction axioms Δ, a complex task schema $t(\overline{v}) = \langle \text{prec}(t(\overline{v})), \text{post}(t(\overline{v})) \rangle$ is called *consistent* if and only if the following conditions hold:

1. All atoms $R(\overline{\tau})$ that occur in $\text{post}(t(\overline{v}))$ are built over flexible symbols $R \in \mathscr{R}_f$. Since the state-abstraction axioms in Δ do not map flexible atoms on rigid ones, this property guarantees that the implied state transition only affects state-dependent features.

   There is however one exception to this rule concerning the term updates. Atoms over the rigid equality relation symbols $\equiv_Z$ with $Z \in \mathscr{Z}$ may occur in a complex task schema definition's postcondition if and only if at least one (sub-) term in the atoms' arguments is built over a flexible symbol.

2. Analogously to the operator consistency, $\Delta \cup \{\text{prec}(t(\overline{v}))\}$ must be satisfiable. Since we require the state-abstraction axioms to be consistent (Def. 2.3), unsatisfiability becomes a locally decidable issue for each task's precondition.

3. Just like the precondition, the effect specification of an abstract task must be satisfiable, too, that means $\Delta \cup \{\text{post}(t(\overline{v}))\}$. Since state-abstraction axioms have to be consistent (Def. 2.3), unsatisfiability becomes a locally decidable issue for each task's postcondition.

•

Concerning operator consistency, it has to be pointed out that conflicting elementary operations do exhibit unpleasant peculiarities. First, we cannot distinguish, for instance, a state transition with an elementary delete following immediately an elementary add from a state transition that does not involve the interpretation of the respective relation symbol at all. It would therefore be at least a questionable way of modelling if we allowed for defining actions that yield the same world state, but do so via different intermediate states. Second, there is no interpretation that satisfies multiple updates on the same term, that means, that the operator does not represent a valid state transition.

Please also note that we do allow for tasks with implicit effects, that means, effects that are implied by the precondition alone and rely on persistence. One example is a complex task with a postcondition of the truth value $\top$; the task is applicable in any state and generates an unchanged state. Permitting such quasi-empty abstract state transitions is a technique that seems related to "stuttering" in the area of formal verification using temporal logics [165]. On the abstract action level, no state change is perceivable while operators on the concrete level do induce state changes. Any modeler is however well advised to employ only trivial stuttering tasks (equal pre- and postcondition, postcondition $\top$), because non-trivial implications are most probably neither intended nor pretty useful. The intention aspect is an assessment of the author, the second is a consequence of the above considerations about conflicting operator effects. It is also a question whether or not operators with implied effects do make sense; from our point of view they do not. We therefore recommend to perform appropriate checks on task schemata (see also future work in Sec. 7.2.3).

Task schema definitions can be validated against the above consistency criteria in a computationally and methodologically inexpensive manner. The construction of schemata can be accompanied by recommending sort-safe parameter usages, etc., or a tractable off-line consistency verification can be performed on a given set of schemata (cf. discussion of abstraction-axiom construction in Sec. 2.8). For the sake of simplicity of the presentation, all following sections and chapters will therefore only refer to consistent task schemata without further distinction.

2.4 Domain Models

The term *domain model* has been used so far in an informal way to refer to all the knowledge about the targeted real-world application area that is considered relevant for plan generation purposes. As already described in Sec. 1.1, every planning system uses a specific representation of the general static knowledge and system dynamics of its application domain (and keeps it separated from the knowledge that describes the particular episode or problem for which a plan is generated). Our framework's domain model representation simply combines the previously introduced specifications of the state-abstraction axioms and appropriate actions according to the following definition:

Definition 2.7 (Domain Model)**.** Given a language $\mathscr{L}$, a *domain model* is the tuple

$$D = \langle \mathscr{M}, \Delta, \mathtt{T} \rangle$$

with $\mathscr{M}$ being the logical model structure, Δ a set of consistent state-abstraction axioms, and $\mathtt{T}$ a set of consistent (abstract and primitive) task schemata.

The set of all domain models over a language $\mathscr{L}$ is denoted by $\mathscr{D}$. •

The term *domain* is often synonymously used for the application domain, the planning domain model in general, and the concrete domain model in a specific planning system language likewise. In addition, subsequent chapters will extend the domain-model structures to more advanced plan generation methods. The particular usage of *domain* will however be either given unambiguously by the context or explicitly annotated.

In order to ensure the definition of meaningful domain models, the axiom sets and task schema definitions firstly have to be consistent in themselves and secondly have to match properly. In addition, any meaningful domain model should apparently provide an in some sense complete set of task schemata such that all complex tasks are implementable by the given operators. The following definition of domain model consistency is consequently directly based on the consistency notions of the respective domain model components and the implementation semantics.

Definition 2.8 (Consistency of Domain Models)**.** A domain model $D = \langle \mathscr{M}, \Delta, \mathtt{T} \rangle$ over a given language $\mathscr{L}$ is called *consistent* if and only if the following conditions hold:

1. The state-abstraction axioms in Δ are consistent (Def. 2.3).
2. All primitive and complex task schemata in $\mathtt{T}$ are consistent with respect to Δ and the state feature hierarchy imposed by it (Def. 2.6).
3. For every complex task schema in $\mathtt{T}$ the domain model provides operator definitions such that instances of some of these operators constitute at least one implementation of the corresponding complex task instance (p. 41).
4. Let $\overline{o} = o(\overline{v})1, \ldots, o(\overline{v})n$ be a finite sequence of operator instances from primitive task schemata in $\mathtt{T}$ and let $s_0 \ldots s_n$ be a corresponding finite sequence of states such that $\langle s_{i-1}, s_i \rangle \models_{\mathscr{M},\beta} o_i$ for all $1 \leq i \leq n$. For any two such sequences of operators $\overline{o}$ and $\overline{o}'$ and two appropriate state sequences $\overline{s}$ and $\overline{s}'$ the following equation holds: if $s_0 = s'_0$ then $s_n \neq s'_{n'}$
5. Let $T = \{t(\overline{v})1, \ldots, t(\overline{v})n\}$ be a finite set of complex task instances from those schemata in $\mathtt{T}$ that do not carry a precondition φ and a postcondition ψ with $\models \varphi$ and $\models \psi$. For any two such sets T and T' the following equation holds:

$$\text{if } \bigwedge_{1 \leq i \leq n} \text{prec}(t(\overline{v})i) \Rightarrow \bigwedge_{1 \leq j \leq n'} \text{prec}(t'_j(\overline{v}'_j)) \text{ then not } \bigwedge_{1 \leq i \leq n} \text{post}(t(\overline{v})i) \Rightarrow \bigwedge_{1 \leq j \leq n'} \text{post}(t'_j(\overline{v}'_j))$$

•

The last two consistency conditions are also referred to as the *conciseness* property of the domain model.

Since in general nothing sound can be deduced or inferred from an inconsistent domain model, every reference in this document to "a domain model" should be read as "a consistent domain model" if not explicitly stated otherwise (cf. Sec. 7.2.3). The conciseness property addresses the problem of domain models that accidently include "hidden" duplicates of abstract and primitive task schemata, which induce the identical behaviour in the execution environment (please note that this still allows for alternative ways of achieving goals). The ruled out action isomorphisms are regarded as severe anomalies with respect to the model semantics because they interfere with the refinement concept that is to be introduced later in Sec. 2.16. We would like to point out that we do however allow for models with weak abstraction, that means, non-conciseness between complex and primitive tasks or between complex tasks with trivial conditions. Firstly, we believe that there are justifiable situations in which an abstract action completely specifies the possible state transitions and serves as a mere place-holder for exactly one operator sequence. Secondly, we

want to cover classical HTN planning where there is no semantic justification for appropriate implementations.

Domain model consistency is satisfied with a notion of a minimal implementation of the model's complex tasks, it is sufficient to provide operators for at least one implementation. But since complex tasks have a more expressive effect representation than operators and in addition make use of state-abstraction axioms, the question arises, whether or not the domain's repertoire of primitive actions is able to completely capture the power of the abstract action descriptions.

A set of implementations $\overline{O} = \{\overline{o}_1, \ldots, \overline{o}_n\}$ for a complex task schema $t(\overline{v})$ with non-trivial pre- or post-condition is called *complete* with respect to a domain model $D = \langle \mathscr{M}, \Delta, \mathtt{T} \rangle$ if and only if for every pair of states $\langle s, s' \rangle$ and valuations β with $s \models_{\mathscr{M},\beta} \text{prec}(t(\overline{v}))$ and $s' \models_{\mathscr{M},\beta} \text{post}(t(\overline{v}))$ the following holds: If there exists a sequence of update function calls $u_1 \ldots u_m$ such that $s u_1 \circ \ldots \circ u_m s'$ then $\overline{O}$ contains an appropriate implementation.

A complete set of implementations thereby "covers" all possible state-refinements. Note that a complete set of implementations is unambiguous for correct (and hence concise) domain models, modulo variable renaming. It has to be mentioned, that in classical HTN planning (p. 1.1.3) the set of user-defined refinement methods is regarded to be complete by definition. In this thesis' view, HTN methods would hardly ever be complete, since the abstract tasks of classical approaches do not carry pre- and postconditions (that means, they are the trivial ones $\models \varphi$ and $\models \psi$) which allows for un-constrained state transitions in its implementations. Thus, all possible operator sequences would be considered legal implementations.

With the definition of domain models, we can finally represent all relevant aspects of the application domain: the terminology and concepts, the relationships between objects, and the laws of dynamics to which the world states adhere. Domain models thereby constitute referential frames for the corresponding courses of action. Hence, we are ready to introduce our notion of a "plan" in the following section.

2.5 Plans

The course of action, the *plan*, is the most important data structure for any planning methodology because its design and the reasoning techniques working on it always go hand in hand (see also introductory sections). We have chosen a particular kind of plan representation to adopt in this thesis, and it is introduced in the following definition: the so-called *partial plan*. The term "partial" originates from early works in the field of Partial-Order Planning (Sec. 1.1.2). The name contrasts with "linear planning", a technique that relies on completely ordered plan steps. While partial-order planning originally simply referred to plans with partially ordered steps, we use it in its later meaning to express that not all decisions for the represented plan entity have been made yet: object identities are under-specified, the execution order of tasks or activities is not finalized yet, not all causal dependencies have been cleared, etc. We will not insist on this literal distinction in later sections; if not mentioned otherwise, the term "plan" will always be used instead and in the sense of "partial plan".

Definition 2.9 (Partial Plan)**.** Given a domain model $D = \langle \mathscr{M}, \Delta, \mathtt{T} \rangle$ and a language $\mathscr{L}$, a *partial plan* P is defined by the tuple

$$P = \langle TE, \prec, VC, CL \rangle$$

A plan consists of the following components:

TE: A finite set of *task expressions* $te = l : t(\overline{v})$, which are tagged instances of corresponding abstract or primitive task schemata in T. The tag l is a label that is unique in *TE* for every task expression. All parameter variables that occur in $\overline{v}$ are assumed to be unique in *TE*, too.

Task expressions are also denominated as *plan steps*.

$\prec$: A finite set of ordering constraints $te_i \prec te_j$ with task expressions $te_i, te_j \in TE$. The ordering constraints impose a partial ordering relation on the task expressions in *TE*; it is interpreted as the intended execution order of the actions represented by the task expressions.

VC: A finite set of variable constraints. They include firstly equations and inequations of the form $v \doteq \tau$ and $v \dot{\neq} \tau$ that co-designate and non-codesignate variables $v \in \mathcal{V}$ with terms τ over the language $\mathcal{L}$.

A second family of variable constraints are the so-called *co-typing constraints* $v \dot{\in} Z$ and non-cotyping constraints $v \dot{\notin} Z$ with sort symbols $Z \in \mathcal{Z}$. They restrict the variables to be eventually co-designated to a term of the specified sort, respectively prohibit the co-designation to a term of the specified sort. In this way, they can be interpreted as disjunctive co-designation constraints, respectively sets of non-codesignation constraints, on the particular sets of constants in the domain of the referenced sort Z.

CL: A finite set of *causal links* $te_i \xrightarrow{\varphi} te_j$ between task expressions te_i and te_j from TE. The annotated *causality* φ is a well-formed formula over the language $\mathcal{L}$ and uses only unbound variables that occur in the parameter lists of the linked plan steps te_i and te_j. The causal link represents the causal relationship between a task $te_i = l_i : t_i(\overline{v}_i)$ that *establishes* a precondition φ of a task $te_j = l_j : t_j(\overline{v}_j)$ by its effects.

This information is formally defined as follows: Let σ_{VC} be a *VC*-compatible variable substitution, that means, a variable substitution that is induced by the equations in *VC* and that is consistent with the inequations, the co-typing, and the non-cotyping constraints. The following two conditions then hold for the causal link:

1. te_i is a valid "establisher" of the condition, that means for any pair of states s and s' with $\langle s, s' \rangle \models_{\mathcal{M},\beta} t_i(\overline{v}_i)$ a formula φ' is generated[4] by te_i such that $\Delta \cup \{\sigma_{VC}(\varphi')\} \models \sigma_{VC}(\varphi)$.
2. te_j is a valid "consumer" of the condition if $\Delta \cup \{\sigma_{VC}(\mathrm{prec}(te_j))\} \models \sigma_{VC}(\varphi)$.

The set of all plans over a given language $\mathcal{L}$ is denoted by $\mathcal{P}$. •

Let us briefly discuss the choice of representation in the above plan definition: Although various optimized data structures have been proposed in the literature (see discussion in introductory section 1.1), we opt for the sake of flexibility and human understandability for the described universal representation. Later sections will explain how the representation of partial plans can easily be extended in order to cover aspects of advanced domain characteristics.

For an illustration of the previous definition, let us return to the running example of the peripheral device delivery scenario. Please recall the outlined plan in Fig. 2.1, a simple plan instance that includes two plan steps: one for opening the box and one for unpacking the printer. According to the intended domain model $D_{devices}$, the plan corresponding to the depicted course of action can be represented as follows:

$$P_{Fig.\ 2.1} = \langle \{te_a = l_a : \mathtt{open}(b_a), te_b = l_b : \mathtt{unpack}(p_b, b_b, d_b)\}, \{te_a \prec te_b\}, \\ \{b_a \doteq \mathtt{box}, b_a \doteq b_b, p_b \doteq \mathtt{printer\_B}, d_b \doteq \mathtt{desk}\}, \{te_a \xrightarrow{\varphi} te_b\} \rangle$$

The annotated causality of the causal link is $\varphi = \mathtt{Open}(\mathtt{b_a})$, and an appropriate set of variables $\{b_a, p_b, b_b, d_b\} \subset \mathcal{V}$ is available in the respective language $\mathcal{L}_{devices}$. Plan step l_a opens the delivered box and step l_b unpacks the printer from it. The variable constraints set the action parameters to the respective objects, so that the execution of the induced ground instances unambiguously opens the desired box and unpacks the specified printer. The ordering constraint and the causal link between the two plan steps represents the commitment to an arrangement in which opening the box provides the state feature of an "open box", which is necessary (and in this example sufficient, too) for unpacking the printer out of the box.

If a plan P contains only primitive task expressions, that means, if all task expressions $te \in TE$ are built over primitive task schemata, the plan is called *primitive*, and the semantics of P is given by its so-called *ground linearizations*.

Definition 2.10 (Ground Linearizations)**.** Given a primitive partial plan $P = \langle TE, \prec, VC, CL \rangle$, the *ground linearizations* of P are the set of all operator sequences $o_1(\overline{v}_1) \ldots o_n(\overline{v}_n)$ with associated valuations β that can be obtained from P as follows:

[4] See the definitions of formula generation by operators (p. 39) and complex tasks (p. 41).

1. Let $te_1 \ldots te_n$ be a sequence of the task expressions in TE that is consistent with the plan step ordering in $\prec$. For every task expression te_i, $1 \leq i \leq n$, the operator sequence contains a corresponding primitive task schema instance $o_i(\overline{v}_i)$ with the same parameters.
2. A valuation β that is associated with the operator sequence in 1 agrees with a VC-compatible ground substitution on the task expression parameters in TE.

•

Please note that in general the ground linearizations include multiple pairs with the same operator sequence but different valuations. This is because the variable constraints may not assign constant values to all variables and therefore multiple ground substitutions are compatible with VC. This includes in particular those variables that are provided by the language of the underlying domain model but that do not occur in the plan steps' parameters or constraint sets. When we speak of the ground linearizations of a plan, we consequently treat all pairs as one representative of an equivalence class that have same operator sequences and that have valuations that agree on all the variables occuring in P.

The example plan $P_{Fig.\ 2.1}$ above is a primitive plan. It represents a single ground linearization, because all variables are bound to rigid constants and the tasks are in linear order. We can apply the state transition semantics of operator sequences (p. 39) to the ground linearization in order to compute two important sets of states: firstly, we can identify the states in which the task sequence is executable and secondly we calculate those states that are a result of the actions' execution. In this sense, the following definition introduces the concepts of executability and state generation for partial plans. To this end, it will interpret the abstract plan steps as the disjunctive set of their implementations and these primitive plans in turn as their ground linearizations.

Definition 2.11 (Executability and State Generation of Partial Plans)**.** For a given domain model $D = \langle \mathscr{M}, \Delta, \mathtt{T} \rangle$, let P be the partial plan $\langle TE, \prec, VC, CL \rangle$ and $\mathbb{P}_P$ the set of primitive plans $\{P_1, \ldots, P_n\}$ that is obtained from P by substituting each abstract plan step in P by one of its implementations.
Let furthermore $\overline{O}_P = \{\overline{o}_{11}, \ldots, \overline{o}_{1m}, \ldots, \overline{o}_{n1}, \ldots, \overline{o}_{nm'}\}$ be a set of operator sequences such that every $\overline{o}_{ij}$ with $1 \leq i \leq n$ and $1 \leq j \leq m_i$ is an element of the ground linearizations of plan P_i. The sequences in $\overline{O}_P$ are also called the *primitive ground linearizations* of P.

P is said to be *executable* in a state $s \in \mathscr{S}$ if and only if for every operator sequence $o_1 \ldots o_m$ and associated valuation β of its ground linearizations $\overline{O}_P$, the first operator o_1 is applicable in s and the precondition of every o_i with $1 < i \leq m$ is generated by the sub-sequence $o_1 \ldots o_{i-1}$.

P in addition *generates* a set of states $S_P = \{s_1, \ldots, s_{|\overline{O}_P|}\}$ for which the following holds: For every linearization $o_1 \ldots o_m \in \overline{O}_P$ with valuation β, there is a corresponding state s_i in the set such that there exists a sequence of states $s'_1 \ldots s'_{m+1}$ with $s = s'_1$, $\langle s_k, s_{k+1} \rangle \models_{\mathscr{M},\beta} o_k$ for $1 \leq k \leq m$, and $s_{k+1} = s_i$. •

According to these definitions, a non-primitive partial plan can be seen as a specification for desired operator sequences. The concept of executability also provides a reasonable starting-point for defining consistent plans: A partial plan P is consistent if and only if there exists at least one state s in which P is executable (and the set of states that it generates is not empty, which is given by the implied executability of the ground linearizations). This notion of consistency can be argued for from an application point of view: If there are no situations in the application domain in which a plan can be carried out, then this plan must be faulty.

This semantic-based criterion is, however, too strict and in its applicability too restricted to be useful for plan-generation purposes: Firstly, it is too strict in the sense that the partial plans that are encountered during plan generation are typically not executable, because the planning process is actually trying to find an executable plan. Practically all intermediate results, independent from the implemented planning method, will be inevitably considered to be inconsistent. There is wide range of causes for the deficiency: plans may be completely corrupted due to inconsistent variable bindings, which does not allow for any ground linearizations, or plans need "only" one of the linearization possibilities to be eliminated.

Secondly, the criterion is too restricted with respect to its own application: working out plan executability implies constructing all implementing primitive plans and all possible states (in order to determine whether

or not one can be constructed such that the plan becomes executable). This certainly cannot be expected to be computationally tractable. A realistically useful consistency concept has to evade these drawbacks by relaxing the semantic executability criterion and by focusing more on syntactical features of the plan. On the other hand, it must however not be weakened to an extent where executable plans can get classified as inconsistent ones. As a conclusion, the relaxed executability criterion has to imply consistency or, symmetrically, inconsistency has to imply that the plan is not executable.

In the line of the preceding arguments, the following definition proposes tractable syntactical criteria that characterize meaningful partial plans.

Definition 2.12 (Consistency of Partial Plans)**.** A partial plan $P = \langle TE, \prec, VC, CL \rangle$ is called *consistent* with respect to a domain model $D = \langle \mathscr{M}, \Delta, \mathtt{T} \rangle$ and language $\mathscr{L} = \langle \mathscr{Z}, \leq, \mathscr{R}_r, \mathscr{R}_f, \mathscr{F}_r, \mathscr{F}_f, \mathscr{V}, \mathscr{T}_p, \mathscr{T}_c, \mathscr{E} \rangle$ if and only if the following conditions hold:

1. The transitive closure of the ordering constraints in $\prec$ constitutes an irreflexive, not symmetric relation[5] on the plan steps in *TE*.
2. The variable constraints *VC* represent a globally consistent constraint system, that is to say, there exists a sort-correct valuation β that relates all variables occuring in *VC* with rigid constants $c \in \mathscr{C}_f$ in $\mathscr{L}$, while it agrees with the equations, inequations, and sort restrictions that are formulated in *VC*.
3. The ordering constraints do not contradict the causal links, that means, for any causal link $te_i \xrightarrow{\varphi} te_j$ in *CL*, there is no ordering $te_i \prec te_j$ in the transitive closure of $\prec$.

•

Note that the plan step parameters do not necessarily occur in the variable constraints, but since we only consider consistent domain models, this cannot imply any variable inconsistency.

A partial plan that fails the consistency specification is definitely un-executable. The following theorem confirms this implication.

Theorem 2.1. *Inconsistent partial plans are not executable in any given state.*

Proof. An inconsistent partial plan $P \in \mathscr{P}$ fails to fulfill at least one of the consistency criteria in Definition 2.12. If the ordering relation $\prec$ contains reflexive or symmetric ordering constraints between the plan steps of *P* then no linearization can be found because of the cyclic paths in the transitive closure of $\prec$. Similarly, an inconsistent constraint set *VC* does not produce any ground instance and therefore no ground linearization can be found. □

The consistency criterion for partial plans completes the representational features of domain knowledge and thereby concludes the sections on the planning components and data structures of the formal framework. With this background, we are able to proceed to address plan generation itself and its algorithmic topics.

2.6 Refinement Planning

The following sections provide the concepts for a generic planning approach: planning by plan refinement. Sec. 2.6.1 describes how a planning problem is specified, introduces the notion of plan refinement, and identifies the conditions under which a refinement is a solution to a given problem. The syntactic structure for representing proper plan-refinement operations, the *plan modifications*, is presented in 2.6.2. As a criterion for the application of the modifications, Sec. 2.6.3 defines *flaws* as the syntactic incarnations of the deficiencies that deter a plan from being a solution; they literally "point" to critical components of the plan. Sec. 2.6.4 subsequently examines the connection between flaws and modifications in general, and

[5] The term *partial order* can be easily confused with the mathematical concept that refers to reflexive, antisymmetric, and transitive relations, $\leq: \mathbb{N} \times \mathbb{N}$, for instance. Planning literature derives its specific terminology from the ordering relation $\prec$, which *orders* the plan steps *partially*.

under which circumstances the former call for the application of the latter. With the functional components that provide modifications and flaws for building the search space in a generic planning algorithm, this section finally presents the constituents of a framework for refinement-based planning. It details the design of these components and their interaction during plan generation with particular consideration of search control. The presentation concludes with a discussion on the algorithmic properties of the resulting system.

2.6.1 Planning Problems, Plan Refinements, and Solutions

In the introductory chapter about AI planning (Sec. 1.1) we presented the key steps in applying planning methods: the definition of a formal model of the application domain, the specification of the problem in terms of the formal model, and the deployment of a generic model-based plan generation algorithm that attempts to develop a plan out of the problem specification that satisfies defined solution criteria. As it has been stated before, design choices in all these steps depend on each other. For the sake of readability, we present them in the above order.

While it is common for the planning literature to include the initial situation in the problem specification, the essential difference between practically all planning approaches is the choice between goal oriented and task oriented problem definitions. Goal oriented problems are specifications of *desired world states*, that means, a course of action is to be constructed such that its execution will change the world state until it finally matches the specified state. Task oriented problems specify an *abstract plan* for which concrete courses of action are to be found according to procedural knowledge, a way of specifying that is primarily used in HTN planning. The presented approach relies on a *hybrid* representation that allows for both the specification of an abstract initial plan together with the declaration of a goal state.

Definition 2.13 (Planning Problem)**.** A *planning problem* π is given by the structure

$$\pi = \langle D, s_{init}, \mathfrak{s}_{goal}, P_{init} \rangle$$

where $D = \langle \mathscr{M}, \Delta, \mathrm{T} \rangle$ is a domain model, $s_{init} \in \mathscr{S}$ with $s_{init} \models_{\mathscr{M},\beta} \bigwedge_{\delta \in \Delta} \delta$ for any valuation β is an *initial state*, $\mathfrak{s}_{goal}$ is a *goal state* specification, and $P_{init} = \langle TE_{init}, \prec_{init}, VC_{init}, CL_{init} \rangle \in \mathscr{P}$ is a consistent *initial partial plan*.

The goal state specification $\mathfrak{s}_{goal}$ is a (satisfiable) formula over $\mathscr{L}$.

The set of all planning problems is denoted by $\amalg$. •

It is worth pointing out that goal state specifications may refer to abstract state features, that means, that atoms that occur $\mathfrak{s}_{goal}$ for which Δ provides refinements. In terms of expressivity, abstract goals correspond to disjunctive goals; as the disjunction is combined with a well-founded axiomatic basis, *abstract goal specifications* substantially increase the expressive power and application range of our approach.

In the domain of our running example, a definition for a planning problem can be given as follows:$\pi_{devices} = (D_{devices}, s_{init}, \mathfrak{s}_{goal}, P_{init})$ with

$$\begin{aligned} s_{init} &\models_{\mathscr{M},\beta} \texttt{Sells}(\texttt{vendor\_X}, \texttt{printer\_B}) \wedge \forall p. \neg \texttt{Connected\_To}(p, \texttt{pc\_5}) \ldots \\ \mathfrak{s}_{goal} &= \texttt{Connected\_To}(\texttt{printer\_B}, \texttt{pc\_5}) \wedge \texttt{Running}(\texttt{printer\_B}) \\ P_{init} &= \langle \{ l_a : \texttt{set\_up\_peripheral}(p_a) \}, \emptyset, \{ p_a \doteq \texttt{printer\_B} \}, \emptyset \rangle \end{aligned}$$

The initial state describes the world at the beginning of plan execution: The personal computer initially is not connected to any peripheral device. It is a peculiarity of an initial state that it typically serves as a reference for all rigid terms and atoms as well. It is therefore also used to specify rigid symbols although their meaning is already given by the domain model's interpretation. In the example above, the initial world situation consequently includes that the vendor for a specific printer model is known, but this is assumed to be invariant against every possible course of action ($\texttt{Sells}_{\texttt{VendorThing}} \in \mathscr{R}_r$). Generally spoken, the problem

is to find a primitive partial plan[6] that is a concretization of the initial abstract sketch (the abstract set-up action), that is executable in the described situation, and that finally yields a world state that matches the demands expressed in the goal. In the example this means that a particular printer is attached to a specified personal computer and running. "Finding a primitive plan" is not to be done by a blind putting-together of plan elements but by a systematic and semantically sound tool: plan refinements.

From a technical point of view, the idea behind the refinement of plans is to start with an abstract or rather underspecified plan fragment that represents a large set of candidate action sequences and to manipulate it in a way that narrows down the candidate set (cf. Sec. 2.6 and [147]). A partial plan is seen in this context as a representation of constraints that shape the structure of the candidate set. Partial plans are consequently mapped onto operator sequences that are consistent with the constraint sets (orderings, variable constraints, etc.).

Our presented approach however wants to provide a maximum freedom with respect to the deployed planning techniques and therefore assumes that it is not foreseeable what kind of additional constraints and data structures will be involved in extended or advanced planning methods. In order to retain the necessary flexibility, we do not define refinements over plan candidates but make use of their state transition semantics and therefore describe refinements by the consequences of plans during execution[7]: Which state sequences are plausible to represent an execution trace of the current plan? A refinement is consequently the systematic exclusion of undesired execution paths, that is to say, refining the observable system behaviour.

Firstly, it has to be specified, under which circumstances a state sequence is regarded to represent a "plausible" execution trace. We will call this property *compatibility*, and it has to capture two aspects: when do two succeeding states represent a transition of a task (basically the inverse of the state transformation semantics of operators), and do these transitions occur in an order that is consistent with the action ordering defined in the partial plan? To this end, we introduce the notion of compatibility on operator sequences, which are, for example, the primitive ground linearizations of a partial plan.

Definition 2.14 (Compatible State Sequences)**.** Given a domain model $D = \langle \mathscr{M}, \Delta, \mathtt{T} \rangle$, a finite sequence of operators $\overline{o} = o_1(\overline{v}_1) \ldots o_n(\overline{v}_n)$, and a valuation β, a finite sequence of states $\overline{s} = s_1 \ldots s_m$ is called *compatible* with $\overline{o}$ if and only if the following conditions hold:

1. For each operator $o_i(\overline{v}_i)$ with $1 \leq i \leq n$ there exists an index $1 \leq j \leq m$ in the state sequence such that $\langle s_j, s_{j+1} \rangle \models_{\mathscr{M},\beta} o_i(\overline{v}_i)$ holds. We call the index tuple $\langle i, j \rangle$ a *compatible transition point*.
2. All states with an index k that does not occur in the set of compatible transition points are called *anonymous transition points*, that means, there is an instance of an operator schema $o(\overline{v})'$ in $\mathtt{T}$ and an valuation β' that agrees with β on all variables occuring in $\overline{v}$ such that $\langle s_k, s_{k+1} \rangle \models_{\mathscr{M},\beta'} o(\overline{v})'$ holds.
3. For all compatible and anonymous transition points $\langle i, j \rangle$ and $\langle i', j' \rangle$: if $i < i'$ then $j < j'$.

•

An example for a compatible state sequence is illustrated in Fig. 2.6. Note that a state sequence is only compatible if those state transitions that are not covered by the operator sequence are in principle performable by adding operators from the domain model. The state sequence $\overline{s}$ does consequently not contain any "impossible" transitions.

We are now ready to extend the concept of compatible state sequences to general, non-primitive partial plans, thereby defining what a plausible series of situation changes means.

Definition 2.15 (Intended System Behaviour)**.** Given a domain model D and a partial plan P, the *intended system behaviour* specified by P is the family of state sequences that are compatible with at least one operator sequence in the primitive ground linearizations of P.

The intended system behaviour is denoted by the function $\mathrm{beh} : \mathscr{P} \rightarrow \mathscr{S}^*$. •

[6] See Sec. 2.8 for a discussion about whether the solution plan necessarily has to be a *primitive* partial plan or not.

[7] This flexibility argument is also used in the context of plan metrics that define the similarity of plans. Recent investigations in this area show that viewing plans as state change inducing behaviour generators rather than reducing them to the concrete data structures provides the flexibility that is required to integrate various plan representation and generation paradigms [66].

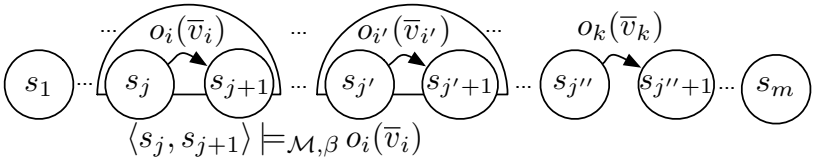


Figure 2.6: An example for compatible state transitions. Operators $o_i(\overline{v}_i)$ and $o_{i'}(\overline{v}_{i'})$ are supposed to originate in the operator sequence, $o_k(\overline{v}_k)$ is a new instance from the domain model's action repertoire.

If a plan P has an empty behaviour, that means, if $\text{beh}(P) = \emptyset$, then the plan is apparently inconsistent and vice versa. For instance, consider the case when the ordering relation becomes cyclic or the variable constraint set co-designates variables to multiple constants: no compatible state sequence can be given for an empty set of primitive ground linearizations. This view obviously coincides with plan consistency as specified in Def. 2.12. The same holds for executability: For a plan $P = \langle TE, \prec, VC, CL\rangle$ that is executable in a state s, *every* subset of P's specified behaviour contains at least one state sequence $\overline{s} = s_0, \ldots, s_m$ with $s = s_0$ and $m = |TE|$. This element of the behaviour is completely explainable by the plan in the sense that it does not contain anonymous transition points.

Given the definition above, refining a plan means concretizing its intended system behaviour. We can also think of viewing the refinement process as the step-wise exclusion of undesired alternatives of a plan's behaviour and therefore a narrowing down of the initial set of possible behaviours to a smaller set that meets the objectives given in the problem specification π. The following definition captures this notion.

Definition 2.16 (Plan Refinement)**.** For a given domain model D, a plan P' is a *refinement* of another plan P if and only if the intended system behaviour specification of P subsumes that of P':

$$\text{beh}(P') \subseteq \text{beh}(P)$$

•

Any plan P' is trivially a refinement of the empty plan $P_\emptyset = (\emptyset, \emptyset, \emptyset, \emptyset)$. An empty plan can be re-formulated as a plan that contains one complex task expression that has been built from a task schema with empty preconditions and effects. As stated before, any plan is an implementation of such a task expression.

Note that plan refinements are defined as *non-strict* reductions of the system behaviour sets. This may appear questionable to some extent as it permits the establishment of plateaux during a sequence of performed refinements. The motivation for our refinement-design being non-strict will be addressed later in the definition of specific plan-refinement operations (see Chap. 3) and other plan manipulation mechanisms (see inferences introduced in Def. 2.40). In all these contexts, auxiliary constraint sets are manipulated, which affects state sequence compatibility of plans only indirectly through subsequent refinements. From the point of view of the partial plan, (temporarily) no change in the intended system behaviour occurs, although a refinement has been performed.

A second important consideration are the implications of domain model consistency (Def. 2.8) on the intended system behaviour of a plan. The conciseness of the domain model accomplishes the unambiguity of state sequences that are induced by operator sequences and thereby provides the unambiguity of the intended system behaviour $\text{beh}(P)$. Following this argument, a development process of partial plans that is based on a reduction of the corresponding behaviour space constitutes a systematic way of plan generation. Firstly, no plan occurs twice in subsequent refinements (there are specific operation sequences associated with the eliminated state sequences and these operations can therefore not be re-introduced) and secondly, every plan has a deterministic set of refinements (there will be no oscillation between refinements that, for example, exchange duplicate tasks).

The first property can also be referred to as the *monotonicity* of refinements and will be addressed in later sections. Concerning conciseness with respect to complex actions, it has to be noted that this property is not mandatory for our notion of consistency. It is however to be considered a good style of model specification

and at least helpful for modelling purposes in order to avoid confusion. Non-concise complex task schemata typically occur in HTN-like planning, when pre-defined implementations for complex tasks are developed "bottom-up". In this modelling anomaly, the domain engineer tries to subsume and structure procedures in abstract action definitions without paying too much attention to an organized knowledge acquisition procedure (see also "methods", their specification principles, and their usage in Sec. 1.1.3). Although the modeler probably intends to produce disjunct subsets of system behaviours, isomorphic complex task schemata may occur, indirectly through the use of state-abstraction axioms or directly by relaxing precondition and effect specifications (for example, when applying the *unique main sub-action* modelling method [291]). From our semantics point of view this ambiguity is not critical, at most only obscure, and at least worth to be reported to a modeler (see Sec. 7.2.3). We nonetheless allow for it, because from a conceptual perspective it is (1) arguable that at some level of abstraction, courses of action may become ambiguous and (2) reasonable in HTN-like planning systems, which bring user-defined refinements to the table that restrict the implementation possibilities to disjunct behaviours.

Now that we have defined what a planning problem specification looks like and what the semantics of a plan refinement are, we are ready to give a meaning to what a solution to a problem actually is when having refinements as plan manipulations at hand.

Definition 2.17 (Solution)**.** Given a planning problem specification $\pi = \langle D, s_{init}, \mathfrak{s}_{goal}, P_{init} \rangle$, a partial plan P over domain model D is called a *solution* to π if and only if the following conditions hold:

1. P is a refinement of P_{init} (Def. 2.16).
2. P is executable in s_{init} (Def. 2.11).
3. All states $S_P = \{s_1, \ldots, s_n\}$ that are generated by P satisfy the goal state specification for any *VC*-compatible valuation β, that means, $\forall s \in S_P : s \models_{\mathscr{M},\beta} \mathfrak{s}_{goal}$ (Def. 2.11).

•

This definition suggests to use plan-refinement generating steps in order to produce partial plans that are candidates for meeting the solution criteria, because this guarantees to meet criterion 1 automatically in contrast to alternative, more complicated behaviour calculations. Please note that according to the refinement definition (any plan is a refinement of the empty plan), a problem specification with an empty initial plan correctly represents a classical problem specification in the partial-order planning paradigm and the solution definitions coincide (cf. Sec. 1.1.2).

A commonly preferred alternative circumscription for a solution in plan-space planning is the so-called *solution plan* (cf. [118, Chap. 5] and [279]). The basic idea is to begin the plan generation process with encoding the problem specification in a so-called *null plan*. This is an ordinary partial plan in which the initial state is represented as a designated start task that bears the initial state features in its effects, thereby "setting up" the initial world state. Similarly, a designated goal task stands for the goal state description and carries the respective features in its preconditions. The null plan is then the input to, for example, a plan-refinement algorithm that uses causal links for goal establishment book-keeping (see Sec. 1.1.2). A solution is found if the current plan's causal structure is complete and unthreatened and if the current plan's ordering constraints and variable bindings are consistent. Translated into the above solution definition of our framework, executability in the initial state is reduced to executability in the *empty state* s_ε in which every flexible relation is interpreted as the empty set and every flexible term as undefined[8]. The generation of a goal state is consequently reduced to executability of the appropriate goal task. Please note that, although the representations are equivalent, there is a subtle conceptual difference between them: it is possible to include causal links in the null plan representation that have their origin in the start task or that end in the goal task. This form of commitment is not directly expressable in our classical problem definition. In terms of expressivity this difference is however neglectable and can be bridged by a simple model adaption[9].

[8] We may assume that the language provides an appropriate symbol for every sort in the functions' ranges. Any equation in which an unknown value occurs cannot be evaluated.

[9] The extension is a transformation that adds unique preconditions and effects to those task schemata that are to be "linked" to the initial and goal states. This "technique" is, for example, applied during expressiveness analyses that compare POCL with HTN planning [83, Chap. 5].

The motivation for the uniform representation of the planning problem and the plan data structure that is used during the search procedure is complexity: It is widely regarded computationally too expensive to verify the current plan's executability and goal satisfaction during the plan generation process by applying sound but costly techniques such as the modal truth criterion [54]. Operations on the causal structure of the plan do not need to distinguish between persistence of initial state facts and task effects, because both are represented by causal links; the analogous argument holds for fulfilling goal state features and task preconditions. We will briefly introduce the idea of a uniform problem representation and a solution plan.

Definition 2.18 (Null Plan)**.** For a given problem definition $\pi = \langle D, s_{init}, \mathfrak{s}_{goal}, P_{init} \rangle$, let the planning language $\mathscr{L}$ contain two designated primitive task labels o_{init} and $o_{goal} \in \mathscr{T}_p$. Given the empty state s_ε and the initial state s_{init}, let $\overline{u} = u_1 \dots u_n$ be a sequence of update functions such that $s_\varepsilon\ u_1 \circ \dots \circ u_n\ s_{init}$ and let $\overline{e} = e_1 \dots e_n$ be the sequence of elementary operations that corresponds to $\overline{u}$. The initial state and the goal state specification induce two operator schemata $o_{init}() = (\top, \overline{e}, \varepsilon)$ and $o_{goal}() = (\mathfrak{s}_{goal}, \varepsilon)$

The so-called *null plan* $P_{null} = \langle TE_{null}, \prec_{null}, VC_{null}, CL_{null} \rangle$ is then obtained from the initial plan $P_{init} = \langle TE_{init}, \prec_{init}, VC_{init}, CL_{init} \rangle$ as follows:

$$
\begin{aligned}
TE_{null} &= TE_{init} \cup \{te_{init} = l_{init} : o_{init}(), te_{goal} = l_{goal} : o_{goal}()\} \\
\prec_{null} &= \prec_{init} \cup \{te_{init} \prec te | \forall te \in TE_{init}\} \cup \{te \prec te_{goal} | \forall te \in TE_{init}\} \\
VC_{null} &= VC_{init} \\
CL_{null} &= CL_{init}
\end{aligned}
$$

•

We do not necessarily require all task expressions of the initial plan in the problem definition to be ordered between the initial and goal task but it is intuitive to do so. The following definition of a solution plan shows that the task artifacts for the initial and goal state are mere auxiliary constructs from the point of view of a plan generation process.

Definition 2.19 (Solution Plan)**.** Given a partial plan P_{null} that has been obtained from a planning problem π, a partial plan $P = \langle TE, \prec, VC, CL \rangle$ is called a *solution plan* if and only if

1. P is a refinement of P_{null} and
2. $P' = (TE \setminus \{te_{init}, te_{goal}\}, \prec', VC, CL')$, with $\prec'$ and CL' containing all constraints and links from P except for those referring to the removed task expressions, is a solution to π.

•

The second solution-plan criterion still necessitates an explicit problem specification. The following theorem therefore enables more handy mechanisms by providing a solution criterion for any plan that is a refinement of a null plan without a reference to a planning problem.

Theorem 2.2. *A solution plan for a null plan P_{null} that has been obtained from a problem specification $\pi = \langle D, s_{init}, \mathfrak{s}_{goal}, P_{init} \rangle$ is a refinement P' of P_{null} that is executable in the empty state s_ε.*

Proof. The theorem follows directly from the construction of the null plan: The initial state task is executable in s_ε and sets up a state that is equivalent to the initial state s_{init} and the goal state description $\mathfrak{s}_{goal}$ is equivalent to the precondition of its task representation. □

This result has practical consequences: Given that only "correct" plan refinements are provided, using the solution plan metaphor reduces the effort of checking for a solution tremendously. Subsequent chapters that deal with implementations of our framework will therefore use synonymously the notions of problem and null plan, respectively of solution and solution plan. For the rest of this chapter's presentation we will however hold on to the "conservative" problem and solution definitions for the the following two reasons: Firstly, the initial task has to be ignored in every correctness consideration, because an initial state – and with it the initial task's effect – typically includes rigid symbols. The same issue arises when abstract goal state specifications are used and the goal state task's precondition consequently contains atoms

for which the decomposition axioms provide refinements. Both is strictly speaking an illicit construction of task schemata (Def. 2.6) but can be tolerated in an implementation for this fixed exception. The second dilemma lies in the initial and goal state being part of the problem description, so the corresponding initial and goal task schemata become **problem-specific** parts of the domain model. Accepting this unintended abuse of model components as part of the problem description is not too problematic, the anomaly should however be considered when referring to problems and domain models that make use of these representations.

The question remains how to transform plans systematically into refinements. The following section introduces an explicit representation of the available plan generation options that accomplish such a plan manipulation.

2.6.2 Plan Modifications

For the representation and eventually mechanization of a *generic* production of plan refinements, we are defining so-called *plan modifications*. They provide a data structure for refinement operations that *describes* the proposed change to the plan and does so in a way that guarantees the changes to result in a refinement. As a prerequisite, some structural entities of partial plans have to be defined: A *component* of a plan $P = \langle TE, \prec, VC, CL \rangle$ is an element of the set $TE \cup \prec \cup VC \cup CL$. This means a component is either a task expression in TE, an ordering constraint in $\prec$, a variable constraint in VC, or a causal link in CL. The term *sub-component* denotes the constituents of components. The sub-components of a task expression are its task symbol, parameters, and precondition and effect formulae. The latter, in turn, consist of further sub-components, eventually this are literals/atoms with their (possibly nested) arguments terms, and so forth. Sub-components of an ordering constraint are the two involved task expressions and their respective sub-components. Similarly, variable constraints have their involved variables, terms and sub-terms, and sort symbols as sub-components, while causal links are constituted of the linked task expressions, their respective sub-components, and the annotated causality literal with its argument sub-components.

Definition 2.20 (Plan Modification)**.** For a given given partial plan $P = \langle TE, \prec, VC, CL \rangle$ and domain model D over a language $\mathscr{L}$, a *plan modification* is the tuple $\mathtt{m} = \langle E^{\oplus}, E^{\ominus} \rangle$ where

1. $E^{\oplus}$ is a set of new plan components over D and $\mathscr{L}$ such that $E^{\oplus} \cap (TE \cup \prec \cup VC \cup CL) = \emptyset$. This set is also called the set of *elementary additions*.
2. $E^{\ominus}$ is a set of components in P with $E^{\ominus} \setminus (TE \cup \prec \cup VC \cup CL) = \emptyset$. It is the set of *elementary deletions*.
3. No sub-component in $E^{\oplus}$ is a component in $E^{\ominus}$.
4. $P' = \mathrm{app}(\mathtt{m}, P)$, is a refinement of P in agreement with Def. 2.16.
5. $E^{\oplus} \cup E^{\ominus} \neq \emptyset$, that means $\mathtt{m}$ does not constitute a trivial refinement.

The set of all plan modifications over a given language $\mathscr{L}$ is denoted by $\mathbb{M}$.

The *application* of a plan modification is thereby characterized by the generic plan transformation function $\mathrm{app} : \mathbb{M} \times \mathscr{P} \to \mathscr{P}$. Its arguments are a plan modification $\mathtt{m} = \langle E^{\oplus}, E^{\ominus} \rangle$ and a plan P for which it returns a plan P' that is obtained from P by adding the elements of $E^{\oplus}$ to and removing those of $E^{\ominus}$ from the respective component set. •

Note that the above definition criteria 1 and 2 imply that an alteration in the execution order of the elementary modifications in $E^{\oplus}$ and $E^{\ominus}$ does not result in a different plan. Combined with criterion 3 this entails that for a plan modification $\mathtt{m}$ and partial plan P, the plan $P' = \mathrm{app}(\mathtt{m}, P)$ is a partial plan in accordance with Def. 2.9. This includes in particular that all components that are referenced in P' do exist, for instance, all task expressions that are sub-components in $\prec'$ will occur in TE'. If in addition criterion 4 holds, the plan modification is not only a mere refinement of the plan but also one that is comprehensibly obtainable from it. Criterion 5 finally rules out empty modifications that do not impose any change to the plan.

Note that there exists only a finite number of distinct proper plan modifications[10] for any given plan over any given domain model and language. Since the symbol sets over $\mathscr{L}$ are finite, all domain models and problem specifications are finite and with them the set of describable (sub-) components.

Definition 2.21 (Applicability and Obtainability Sets)**.** Given a domain model D and a plan modification $\mathtt{m}$, the set of all plans for which $\mathtt{m}$ is a proper plan modification is called the *applicability set* of $\mathtt{m}$. The set of plans that are generated by $\mathtt{m}$ out of the plans in the applicability set is referred to as the *obtainability set* of $\mathtt{m}$. •

It follows directly from definitions 2.20 that for any domain model and any plan modification $\mathtt{m}$, if P is in the applicability set of $\mathtt{m}$ and $P' = \text{app}(\mathtt{m}, P)$ then $\mathtt{m}$ is *not* applicable to P'. This is because $\mathtt{m}$'s E sets are not empty and therefore all elements from $E^{\ominus}$ will be not available in P', respectively will be already present for $E^{\oplus}$. As a consequence, the applicability and obtainability sets are disjoint for any plan modification.

An instance for a plan modification in our running example of the peripheral device scenario is the following plan modification:

$$\mathtt{m}_{exmp} = (\{l_a : \texttt{set\_up\_peripheral}(p_a) \xrightarrow{\texttt{Running(printer\_B)}} te_{goal}\}, \emptyset)$$

It is applicable in the example problem definition on page 49 (remember the problem is given in the null plan representation) and adds a causal link to $\pi_{devices}$.

We will now investigate, how plans can be developed by subsequently applying plan modifications.

Definition 2.22 (Plan Refinement Space)**.** A *plan refinement space* is a directed graph with partial plans as vertices and plan modifications as edges. An outgoing edge from a vertex P in the plan space is a plan modification for the transformation of P into a successor partial plan P' with $P' = \text{app}(\mathtt{m}, P)$.

Let $M_x = \{\mathtt{m}_{x.0}, \mathtt{m}_{x.1}, \ldots, \mathtt{m}_{x.n}\}$ be a set of plan modifications for a given domain model D and plan P_x. The set of plans obtained from P_x by applying the modifications in M_x is then given by the set $\{P_{x.0}, P_{x.1}, \ldots, P_{x.n}\}$ with $P_{x.i} = \text{app}(\mathtt{m}_{x.i}, P_x)$, $0 \leq i \leq n$. In this way, a family of plan modifications $M = \bigcup_x M_x$ generates the plan refinement space over a given plan P_0 as depicted in Fig. 2.7. •

Plan P_0 in Fig. 2.7 is, for instance, in the applicability set of the 0_n-many plan modifications $\mathtt{m}_{0.0}$, $\mathtt{m}_{0.1}$, etc., to $\mathtt{m}_{0.n}$. The result of applying, for example, $\mathtt{m}_{0.0}$ to P_0 is the plan $P_{0.0}$. This plan is in turn in the applicability set of modifications $\mathtt{m}_{0.0.0}$ to $\mathtt{m}_{0.0.n}$, and so forth.

Note that **exactly one** plan is generated from each modification's obtainability set and that all direct successor plans of a given plan are distinct, since the modification application function is unique (and the sets are duplicate-free). It is also worth mentioning that for any plan P with modifications $\mathtt{m}_1$ and $\mathtt{m}_2$, $\mathtt{m}_1 \neq \mathtt{m}_2$ the following holds: $\text{app}(\mathtt{m}_2, \text{app}(\mathtt{m}_1, P)) = \text{app}(\mathtt{m}_1, \text{app}(\mathtt{m}_2, P))$. A plan refinement space can consequently contain duplicates in different subspaces. This issue will be addressed in the discussion of the systematicity property of the plan generation process (Sec. 2.8).

One problem that has to be addressed is that of infinite paths in a refinement space, respectively duplicate plans on a particular refinement path. This problem occurs for plan modification families in which mutually neutralizing modifications are provided for subsequent plans. Although such a refinement space can be regarded as to be pathological, it appears however inevitable to think about precautionary measures: If mutually neutralizing plan modification cannot be ruled out by construction, an accidental incident of the anomaly cannot be prevented in view of a potentially large number of contributing plan modifications. The dilemma behind this phenomenon is rooted in the definition of plan refinements (Def. 2.16) to be *non-strict* behaviour reductions. Manipulations of a plan's causal link set or of other additional constraint sets in extended framework implementations do not necessarily reflect immediately in the intended system behaviour. Restricting the refinement definition is however not an option, because if only proper subsets of intended system behaviours implied a refinement, the modularity and flexibility of the framework would be

[10] In the strict sense, all plan modifications are proper by definition. We sometimes will use the term "proper" to emphasize that the modification structure fulfills all defined requirements or if the given context requires to contrast regular modifications with non-regular structures.

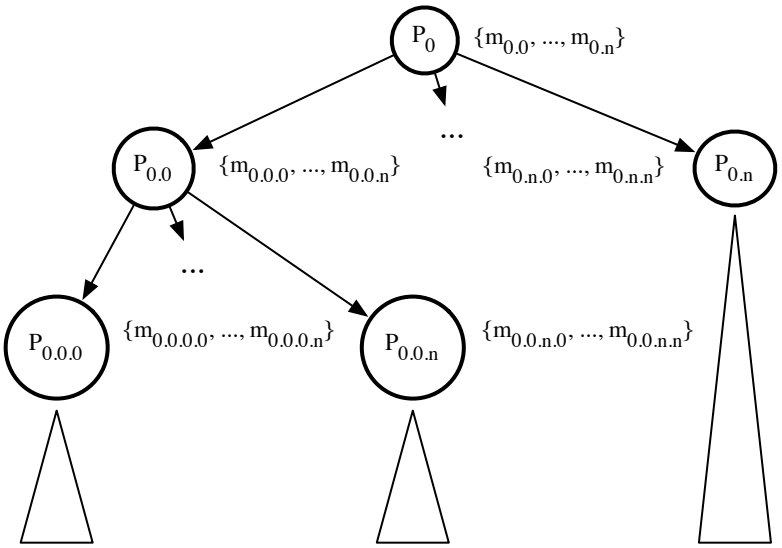


Figure 2.7: The plan refinement space built over a plan P_0.

limited. Later chapters will present some implementations that explicitly benefit from this design. For the following sections we will therefore introduce a necessary criterion for the generation of plan modifications in order to exclusively provide plan modifications that do not induce cyclic refinement paths. We will consequently revisit this topic when dealing with the generation of plan modifications and describe how to construct plan modifications that comply with the criterion. Note that all plan modifications that simply add task expressions, variable constraints, and ordering constraints are necessarily acyclic due to domain model conciseness (Def. 2.8) and due to the monotonicity of the constraint systems; these are in practice the vast majority of plan modifications.

In the line of the above arguments, the following definition specifies the criteria that have to be considered in order to provide meaningful plan modifications. They are also valuable for controlling the size of a resulting refinement space by a finite bound on the length of modification application paths.

Definition 2.23 (Complete and Acyclic Plan Modification Families)**.** Given a planning problem specification $\pi = \langle D, s_{init}, \mathfrak{s}_{goal}, P_{init} \rangle$, a family of plan modifications $M = \bigcup_x M_x$ is called *complete* if and only if all solutions P_{sol} for π are included in the plan refinement space that is generated by M from P_{init}.

A family of plan modifications $M = \bigcup_x M_x$ is called *acyclic* with respect to a given partial plan P if and only if on all paths in the refinement space over P that is generated by M, there are no two modifications $\mathtt{m}_x$ and $\mathtt{m}_{x.y}$ such that $E_x^{\ominus} \cap E_{x.y}^{\oplus} \neq \emptyset$. •

The completeness of plan modification sets is not necessarily related to the completeness of the plan generation process itself – including approaches that are based on traversing the plan refinement space via the above defined plan modification families. In the context of our plan modifications, the term *completeness* means that the generated plan refinement space contains all solutions according to the generic state-transition based view on the domain model. That is to say, the refinement space that is generated over the initial plan contains all refinements that are executable in the given initial state and that satisfy the goal specification. It has to be noted that not all planning paradigms support a complete family of plan modifications in this view. For example, HTN planning (p. 1.1.3) in its purest form is explicitly not interested in finding "whichever" solutions but only those that are covered by the HTN-specific user-defined "expansion" refinements. The concept of completeness with respect to the plan generation process only addresses those solutions that are actually covered by a given not necessarily completed refinement space. This will be discussed in more detail in Sec. 2.6.4. To cut a long story short: It is possible to perform a complete search over an incomplete plan refinement space, and vice versa.

The acyclic property is an alternative formulation of "no plan on any path is equivalent to one of its refinements". Since the verification of plan equality is computationally extremely hard[11], we make use of the above constructive criterion. It is easy to see that the structural restriction for plan modifications implies that for any acyclic family of plan modifications and for any two plans with $P_{x.y} = \text{app}(\mathtt{m}_n, \ldots \text{app}(\mathtt{m}_0, P_x))$ the inequation $P_x \neq P_{x.y}$ holds.

Using acyclic plan modifications yields the desired result: the depth of the refinement space is bound, as it is shown in the following theorem.

Theorem 2.3 (Finiteness of Plan Refinement Spaces)**.** *Any plan refinement space generated by an acyclic family of plan modifications $M = \bigcup_x M_x$ from a partial plan P is finite.*

Proof. Because of the finiteness of the logical model, there exists only a finite number of plan modifications for any plan on the paths in the refinement space (there is only a finite number of components that can be added or removed). That means, the number of branches in the refinement space is bound by the number of proper plan modifications for any node. Since the plan modification family is acyclic, no plan components can be re-introduced once they have been removed from P or one of its refinements. The number of plans that can be generated along a specific path starting in plan P is therefore bound by two times the number of components expressible with the used language and domain model – in the worst case all components are added to some refinement of P and removed later from sub-refinements. □

The upper bound in the previous proof over-estimates the path length by several orders of magnitude, because it does not take into account the refinement property of proper plan modifications. But nevertheless, the result holds and therefore traversing the refinement space is terminating in our framework (assuming a systematic search procedure).

2.6.3 Flaws

A generic planning procedure can be easily formulated with the previously introduced plan modifications: Given the initial plan P_{init} of a planning problem π, firstly the refinement space over P_{init} is generated. After that all paths in the tree are to be followed until either a refinement is reached that satisfies the solution criteria or an inconsistent plan is encountered. In the latter case, search is resumed on another path. It is easy to see that even if plan modifications are selected in an absolute chaotic process, solutions will eventually be found (provided that the search schema is systematic and complete, see Sec. 2.6.4), but of course it has to be expected to do so with a very poor efficiency. Meandering through the refinement space impends to get entangled in, albeit finite, arbitrary long modification paths that do not cover any solution. We therefore aim at performing a systematic way of exploration that uses rational and solution-oriented criteria for the application of plan modifications to a plan structure. In order to provide such criteria, we will now define the flaw data structure for a precise translation of the problem specification into deficiencies in a partial plan, that is to say, a representation of evidence for the (expected) violation of solution criteria.

Definition 2.24 (Flaw)**.** For a given partial plan P and planning problem π over a domain model D, a *flaw* $\mathtt{f}$ is a finite set of (sub-) components in P.

The set of all flaws is denoted by $\mathbb{F}$. •

Chap. 3 will present a variety of flaw definitions that ranges from single component references to larger sets of plan elements. For instance, the null plan obtained from the example problem definition on page 49 does not satisfy the specified goals, and therefore the goal task is not executable: A flaw to express this deficiency can be formulated as $\mathtt{f}_{exmpA} = \{te_{goal}\}$ or $\mathtt{f}_{exmpB} = \{\mathtt{Connected\_To}(\mathtt{printer\_B}, \mathtt{pc\_5}), \mathtt{Running}(\mathtt{printer\_B})\}$, and so on. However, while plan modifications have an operational semantics for their content, flaws do not have any corresponding grounding in the plan structures. This hinders a strong notion of flaw correctness (for instance, "plans of kind x call for flaws with element y") because flaws are so far nothing but unspecified

[11] Plan equality can be reduced to the graph isomorphism problem. Although it is not NP-complete, it is nevertheless in NP and therefore a probably hard problem [156].

deficiency markers. At least, a notion of flaw consistency can be deduced from the intended purpose: flaws should be formulated properly in terms of the given plan and domain model and they shall not give false negative evidence to solutions.

Definition 2.25 (Consistency of Flaws, Flawed Plans)**.** For a given domain model D and planning problem π, a flaw $\mathtt{f} = \{e_1, \ldots, e_n\}$ is called *consistent* with respect to a partial plan P, if and only if all elements that are referenced by the flaw are components or sub-components of P and if in addition P is no solution to π. •

Since the symbol sets over any language $\mathscr{L}$ are finite – and therefore all domain models and problem specifications are finite and with them the set of describable (sub-) components – there exists only a finite number of distinct (consistent) flaws for any given plan and problem.

The notion of singular flaw consistency with respect to a plan can be lifted to families of flaws for sets of plans.

Definition 2.26 (Consistent and Complete Flaw Families)**.** Let $\widehat{P} = \{P_1, \ldots, P_n\}$ be a set of partial plans, and let $\widehat{F} = \bigcup_{i=1}^{n} F_i$ be a family of flaws such that each subset F_i is associated with plan P_i for $1 \leq i \leq n$.

$\widehat{F}$ is called *consistent* with $\widehat{P}$ for a given planning problem π if and only if all flaws $\mathtt{f} \in F_i$ for $1 \leq i \leq n$ are consistent with their associated plan P_i.

A family of flaws $\widehat{F}$ that is consistent with a set of partial plans $\widehat{P}$ for a given planning problem π is furthermore called *complete* with respect to π if and only if for every plan P_i with $1 \leq i \leq n$ the following holds: if P_i is not a solution to π then $F_i \neq \emptyset$. •

Complete plan modification families are intended to systematically build a plan refinement space over a problem description. Analogously, the purpose of the above counter-part of Def 2.23 is to provide a corresponding consistent deficiency assessment: Given a flaw family that is consistent and complete, no solution is erroneously identified as being faulty and no faulty plan ever passes un-marked.

2.6.4 Strategic Refinement Planning

With the above defined flaws and modifications, it is now possible to operationalize a refinement-planning process that is based on plan-modifications. Their explicit representation suggests a mechanism that decides *which* deficiencies in a given plan are to be eliminated *when* and *how*. The "which" part is covered by the flaws, the "how" by the plan modifications. What is still missing, is the "when" part: a systematic selection of *appropriate* plan modification steps for a better informed traversal of the refinement space.

Under which conditions is a plan modification an appropriate candidate to eliminate a given flaw? Apart from the concrete plan at hand, it is known that some *classes* of modifications address particular *classes* of flaws while others do not. In order to create suitable subsets of flaw and modification individuals for a given plan and domain model (and in order to make the plan generation process operational), we will now associate flaw and modification classes with respective groups of functions that are responsible for their computation. This design encapsulates both the detection of solution criteria violations and the computation of possibly solving modifications. Thereafter we will examine their relationship.

Definition 2.27 (Detection Function)**.** Given a partial plan P and a planning problem π, a *detection function* is the function

$$f_x^{det} : \mathscr{P} \times \amalg\amalg \rightarrow 2^{\mathbb{F}_x}$$

that returns flaws of a given class x with $\mathbb{F}_x \subseteq \mathbb{F}$.

The set of all partial plans that are *flawed by* representatives of class $\mathbb{F}_x$ with respect to π is called the *affected set* of $\mathbb{F}_x$. It is defined as $\mathscr{P}_{\mathbb{F}_x} = \{P \in \mathscr{P} | f_x^{det}(P, \pi) \neq \emptyset\}$. •

For the simplicity of presentation, the implementations that are presented in the following chapter assume that the planning problem is encoded as a null plan, that means, the flaws are interpreted as whether or not the current plan is a solution plan (see Def. 2.19 and 2.18). The respective sections will consequently use detection functions with a reduced signature that omits the explicit problem specification. We furthermore suppose that there exists exactly one flaw detecting function for each intended class of flaws, although it is possible to provide function definitions that address only a partition $\mathbb{F}_{x'} \subset \mathbb{F}_x$ in the sense of class-specific specialists. In the following we are consequently presume that every detection function returns *every possible* flaw of its intended class.

The following definition characterizes those detection functions that behave consistent with the definition of flaws as indicators for solution criteria violations.

Definition 2.28 (Soundness of Detection Functions)**.** A detection function f_x^{det} is called *sound* if and only if for every partial plan P and problem π it only creates flaws of class x that are consistent with P. In particular:

$$\forall \pi \in \amalg\!\amalg,\ \forall P \in \mathscr{P} :\ \text{if } P \text{ is a solution to } \pi \text{ then } f_x^{det}(P,\pi) = \emptyset$$

This is equivalent to the statement that no solution plan P is in the affected set of $\mathbb{F}_x$. •

Soundness has two immediate impacts on the perception of detection functions: First, the affected set of any sound detection function is a proper subset of the set of all plans, that means for all $\mathbb{F}_x \subseteq \mathbb{F}$ we find $\mathscr{P}_{\mathbb{F}_x} \subset \mathscr{P}$. This is a direct consequence from the fact that there exist some trivial problems for any consistent domain model. For these problems exist trivial solutions which in turn must not be in the affected set of a sound detection function.

The second implication is not directly provided by soundness, however its usefulness is only given for sound detection functions: sound detection functions are *monotonic*. For any two given plans P_a and P_b with plan b being a refinement of a, let a detection function diagnose a flaw `f` in a but not in b. If the detection function is sound then there is no refinement of P_b in which the flaw will be discovered. Monotonicity is an important characteristics of sound detection functions, because it means that flaws are resolved "once and for all".

This valuable result is worth a dedicated theorem:

Theorem 2.4 (Monotonicity of Detection Functions)**.** *A sound detection function f_x^{det} is* monotonic *with respect to plan refinements:*

$$\begin{aligned}&\forall \pi \in \amalg\!\amalg,\ \forall \mathtt{f} \in \mathbb{F},\ \forall P_i, P_j, P_k \in \mathscr{P} :\\ &\quad \mathrm{beh}(P_i) \supseteq \mathrm{beh}(P_j) \supseteq \mathrm{beh}(P_k) \wedge \mathtt{f} \in f_x^{det}(P_i,\pi) \wedge \mathtt{f} \notin f_x^{det}(P_j,\pi) \Rightarrow \mathtt{f} \notin f_x^{det}(P_k,\pi)\end{aligned}$$

Proof. Let the partial plan P_i be in the affected set $\mathscr{P}_{\mathbb{F}_x}$ of detection function f_x^{det} and consequently flawed by some flaw `f` for a given problem π. That means, that P_i is in an equivalence class for some argument, say Q, that implies the falsification of some solution criterion in P_i. Argument Q thereby sets up equivalence classes of intended system behaviours that are flawed for some common reason. If a refinement of P_i, say plan P_j, is however not applicable to argument Q, its reduced behaviour $\mathrm{beh}(P_j) \subset \mathrm{beh}(P_i)$ does not longer contain members of Q's domain. Consequently, Q will not hold for any intended behaviour of succeeding refinements P_k, hence any such refinement P_k will not be flawed by `f`. □

Fig. 2.8 illustrates monotonicity in a plan refinement space with the red point representing plan P_i in the above proof. The three depicted areas in the refinement space stand (from left to right) for the plans in which the specific flaw is never encountered, the flaw is detected but persists, and the flaw is solved and hence disappeared. Later sections will in particular examine the modifications along the border-line between the flawed plans and their un-flawed refinements: These green transitions are obviously induced by plan modifications that are "appropriate" for the flaw.

The actual reason for a plan not being a solution, that is, the semantics of the plan's deficient elements in a flaw `f`, is only given by the semantics of the flaw generating function and the intended flaw class. We can, for example, formulate a very general flaw class for expressing "the current plan is not executable"

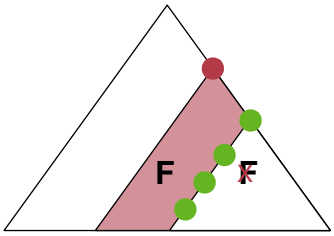


Figure 2.8: Sound flaw detection functions in a refinement space: once flaws that are encountered (red) are resolved (green) in refinements, they do never re-appear.

and define the referenced components as those tasks, the operations of which are not executable in all possible plan linearizations (Sec. 2.5). Alternatively, we can build a very fine granular partitioning of deficiencies, for example dealing with rigid and flexible causal structures separately. The following definition consequently captures the notion of whether such a partitioning covers the solution criteria completely or not.

Definition 2.29 (Completeness of Detection Function Sets)**.** A given set of detection functions $\mathfrak{Det}$ is called *complete* if and only if the following holds:

$$\forall \pi \in \amalg\!\amalg, \forall P \in \mathscr{P} : P \text{ is not a solution to } \pi \Rightarrow \bigcup_{f_x^{det} \in \mathfrak{Det}} f_x^{det}(P,\pi) \neq \emptyset$$

•

Consequently, if a set of sound detection functions is complete, their announcement of flaws becomes equivalent to the currently investigated plan being a solution. When applied to a set of plans, a complete set of sound detection function apparently provides a consistent and complete family of flaws in accordance with Def. 2.26. This motivates the following solution criterion for plans that is based on a complete detection function set.

Theorem 2.5 (Solution Criterion)**.** *Let* $\mathfrak{Det} = \{f_{x_1}^{det}, \ldots, f_{x_n}^{det}\}$ *be a complete set of detection functions. For any plan P and problem* π *the flaw-based* solution criterion *satisfies the following equation:*

$$\forall \pi \in \amalg\!\amalg, \forall P \in \mathscr{P} : P \textit{ is a solution to } \pi \Leftrightarrow \bigcup_{1 \leq i \leq n} f_{x_i}^{det}(P,\pi) = \emptyset$$

Proof. The theorem follows directly from definitions 2.29 and 2.26. □

Since the detection function set is complete, any plan that is not a solution to the given problem would induce at least one of the detection functions to issue at least one flaw. It is also worth noting, that a planning procedure can safely decide to conclude further plan development based on the solution criterion or not: According to Theorem 2.4, the consecutive application of plan modification steps cannot corrupt the current solution and re-introduce any flaw.

One particular impact of flaw semantics has to be pointed out: It is an important issue for finding in some sense "appropriate" plan modifications for the currently investigated plan. Flaw classes, respectively the plans in the *affected* sets of the associated detection function, can be interpreted as characterizations of equivalence classes of plans that are not solutions for specific planning problems for a common reason. Flaw classes are in this sense conceptualizations of why the plan under examination is not (yet) a solution and their instances, the flaws, are the vocabulary for representing arguments about that. In fact, it appears reasonable to use ontologies to represent flaw classes, including subclass relationships and the like. This topic is also picked up in Sec. 5.1.2.

In the same way that flaws are instantiated via the detection functions, plan modifications are provided by a respective generating function.

Definition 2.30 (Modification Generating Function)**.** Given a partial plan P, a flaw $\mathtt{f}$ that is associated with the plan, and a domain model D, a *modification generating function* is defined as the function

$$f_y^{mod} : \mathscr{P} \times \mathbb{F} \times \mathscr{D} \rightarrow 2^{\mathbb{M}_y}$$

that computes applicable plan modification individuals of a given type y with $\mathbb{M}_y \subseteq \mathbb{M}$. •

We may assume without loss of generality that there is a one-to-one relationship between modification classes and generating functions. Like we did for the flaw detection functions, we suppose that the generators are therefore able to produce all modifications of the intended class.

Plan modification classes can be organized in two ways: according to their concrete structure as, for instance, "ordering constraint addition modifications" or according to more abstract descriptions like "narrowing choices in the constraint sets". But howsoever a modification generating function is defined, it has to comply with the following soundness criteria.

Definition 2.31 (Soundness of Modification Generating Functions)**.** A modification generating function f_y^{mod} is called *sound* if and only if it solely generates proper plan modifications that address the argument flaw and that have a positive balance with respect to the plan components in the elementary additions and deletions. More precisely, for all partial plans P and flaws $\mathtt{f}$ over a domain model D the following conditions hold:

1. $f_y^{mod}(P, \mathtt{f}, D)$ returns a (possibly empty) set of plan modifications for P.
2. Let $\mathtt{m} = \langle E^{\oplus}, E^{\ominus} \rangle \in f_y^{mod}(P, \mathtt{f}, D)$ be a plan modification that has been generated in order to address the flaw $\mathtt{f} = \{e_1, \ldots, e_n\}$, and let E_{mod} be the set of all components of P that are referenced by $E^{\ominus} \cup E^{\oplus}$ and their respective sub-components. If E_{flaw} is the union of $\mathtt{f}$ and all of its referenced sub-components then equation $E_{flaw} \cap E_{mod} \neq \emptyset$ holds. This means, that the modification references, directly or indirectly, at least some of the components that are included in the flaw description.
3. Every generated plan modification $\mathtt{m} = \langle E^{\oplus}, E^{\ominus} \rangle \in f_y^{mod}(P, \mathtt{f}, D)$ adds more plan components than it removes, that means $|E^{\oplus}| > |E^{\ominus}|$.

•

An example for a sound modification generating function is the function for adding ordering constraints between task expressions. The function is sound, if it publishes all possible orderings whenever a flaw is provided that contains a task expression of the current plan. It is however not guaranteed that any of the modifications will contribute to the problem the flaw represents, because the semantics of the denoted deficiency is not known to the modification generator. It has to be noted that soundness of a plan modification is deliberately not defined over input flaws that are provided by sound detection functions. Readers who have expected statements like "modifications have to solve the input flaw" may be inclined to question this choice of arguments. The reason for defining the soundness of modification generating function so loosely lies in the intended system design, the highest priority of which is *flexibility*: as the later algorithm sections and the framework incarnations will show, one strength of the approach is that in general, no modification generator has to anticipate the flaws it receives. Having such a loose coupling between detection and modification generating functions allows for an independent development of both and guarantees a proper co-operation in configurations. The issue of finding appropriate candidates amongst the modification generators will be addressed soon.

Modification generating functions impose a system of equivalence classes on plans, similar to the way flaw detection functions do. Based on the applicability and obtainability sets of plan modifications, we define the respective sets for the associated generating functions in a straight-forward manner: the *applicability set* of a modification generating function is defined as the union of the applicability sets of all plan modifications it creates for any flaw structures. In other words, the function's applicability set is constituted by all plans for which the function is able to compute a plan modification for a given addressable flaw. Analogously, the *obtainability set* of a modification generating function is the union of the obtainability sets of all of its plan modifications. It is worth mentioning that the applicability and obtainability sets are modulated by the passed flaw arguments such that they reduce the generators output; there are plans in the applicability set,

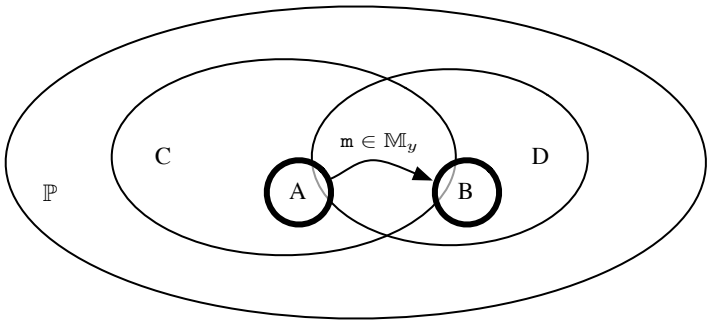


Figure 2.9: The equivalence classes of plans $\mathscr{P}$ for a plan modification class $\mathbb{M}_y \subseteq \mathbb{M}$ as provided by a generating function f_y^{mod} and those for one of the generated instances $\mathtt{m} \in \mathbb{M}_y$.

for instance, that may be solutions for a given problem and therefore there are no flaws to be addressed. The difference between the applicability set and the effective answer set of a modification generating function is subject to the following definition of completeness.

Definition 2.32 (Completeness of Modification Generating Functions)**.** Given a partial plan P, let $\mathbb{M}_y^P$ be the set of all plan modifications of type y that are applicable in P.

A modification generating function f_y^{mod} is called *complete* if and only if for any partial plan P in its applicability set and for any flaw $\mathtt{f}$ that is detected in P for a given problem π, the set $\mathbb{M}_y^P \setminus f_y^{mod}(P,\mathtt{f},D)$ does not contain any modifications that address $\mathtt{f}$.

A set of modification generating functions $\mathfrak{Mod}$ is called *semi-complete* if all of its members are complete. It is called *complete* if and only if for any plan P and planning problem π its accumulated output constitutes a complete family of plan modifications for π (Def. 2.23). •

Completeness of a single modification generating function can be viewed as the property not to cut any branch in the refinement space prematurely. A generator offers every possible refinement for a flaw. A contrasting procedure is discussed for search related topics in Sec. 2.8: In the context of specific information about the current plan development, the flaws, etc., a generation function may retain un-promising refinement options that are derived from some heuristic considerations in order to improve efficiency. The discussion also covers relevant aspects of complete generator sets and their impact.

Returning to applicability and obtainability examinations, Fig. 2.9 illustrates the relationship between the equivalence classes of plans that are imposed by a single modification and its related plan modification class:

- Set A is the set of all plans in which a given plan modification $\mathtt{m}$ of class $\mathbb{M}_y$ is applicable. This set may overlap with sets A' for modifications $\mathtt{m}' \in \mathbb{M}_y$ with $\mathtt{m}' \neq \mathtt{m}$.
- Set $C \supseteq A$ is the applicability set of $\mathbb{M}_y$, respectively that of its associated generating function f_y^{mod}. C may overlap with sets C' for separate modification classes $\mathbb{M}_{y'} \neq \mathbb{M}_y$.
- B denotes the set of all plans that are the result of applying plan modification $\mathtt{m}$ to one of the plans $P \in A$. Since a modification is not applicable in the refinement obtained by its own application, A and B do not intersect and $A \cap B = \emptyset$ holds. Set B may overlap with analogous sets B' for modifications $\mathtt{m}' \in \mathbb{M}_y$ with $\mathtt{m}' \neq \mathtt{m}$ (see Def. 2.22 for a note on commutative modifications).
- $D \supseteq B$ is finally the obtainability set of f_y^{mod}, that means, the set of those plans that are the result of applying a modification $\mathtt{m} \in \mathbb{M}_y$ to a plan $P \in C$. This set may overlap with sets D' for more modification classes $\mathbb{M}_{y'} \neq \mathbb{M}_y$.

Regarding sets A and B, it has to be re-emphasized that the modifications' plan transformations are unique for any two modifications $\mathtt{m} \neq \mathtt{m}'$, even for overlapping applicability and obtainability sets.

Some additional relationships between the plan space compartments have to be mentioned that are represented in the figure. Trivially, A and B are subsets of C and D, respectively. A is not disjunct from D, because the plan in which, for example, some modification $\mathtt{m}$ is applicable could be generated out of a plan in C by a modification $\mathtt{m}' \neq \mathtt{m}$ from the same class. The analogous argument holds for B not being disjunct from C. As a consequence, sets C and D overlap. It is however worth noting that any sequence of modifications of class$y$ will finally leave $\mathbb{M}_y$'s applicability set if f_y^{mod} is sound, hence $C \setminus D \neq \emptyset$ and $D \setminus C \neq \emptyset$. This property is, analogously to the detection functions, called the monotonicity of modification generating functions and more formally defined as follows:

Definition 2.33 (Monotonicity of Modification Generating Functions)**.** Let $P_1 \dots P_n P_{n+1}$ be a sequence of partial plans over a domain model D and let $\mathtt{f}_1 \dots \mathtt{f}_{n+1}$ be a corresponding sequence of flaws such that P_i is affected by $\mathtt{f}_i$ for $1 \leq i \leq n$. Given a set of modification generating functions $\mathfrak{Mod}$, let furthermore $\mathtt{m}_1 \dots \mathtt{m}_n$ be a sequence of plan modifications such that $\mathtt{m}_i \in f_y^{mod}(P_i, \mathtt{f}_i, D)$ for some $f_y^{mod} \in \mathfrak{Mod}$ and $P_{i+1} = \mathrm{app}(\mathtt{m}_i, P_i)$.

The set of modification generating functions $\mathfrak{Mod}$ is called *monotonic* with respect to plan refinements if and only if the first plan modification $\mathtt{m}_1$ is not applicable in the last plan P_{n+1}. •

According to the following theorem the above monotonicity property holds for any set of sound generator functions.

Theorem 2.6 (Monotonicity of Modification Generating Functions)**.** *A set of sound modification generating functions $\mathfrak{Mod} = \{f_{y_1}^{mod}, \dots, f_{y_n}^{mod}\}$ is monotonic with respect to plan refinements.*

Proof. Let $\mathtt{m}_1 = \langle E_1^{\oplus}, E_1^{\ominus} \rangle$ be the plan modification that is returned by a sound modification generating function in $\mathfrak{Mod}$ that has been applied to a plan P_1 and a flaw $\mathtt{f}_1$ for a given problem π. In the refinement plan obtained from this plan, say $P_2 = \mathrm{app}(\mathtt{m}_1, P_1)$, the modification $\mathtt{m}_1$ is not applicable by definition. This is because all elements in $E_1^{\oplus}$ are already present in P_2 and none of the removed elements in $E_1^{\ominus}$ can be removed from P_2. Since $\mathtt{m}_1$ has been issued by a sound generator and therefore added more elements than it removed, *undoing* its induced changes implied that consecutive refinements would have to remove at least more elements than they have to add. This contradicts however the soundness of the modification generating functions and therefore no plan modification can be undone. That means, there is no plan in the sequence of refinements in which $\mathtt{m}_1$ becomes applicable again. □

The property of monotonicity of plan modification generating functions translates directly to an analogous characteristic for the plan refinements that are induced by them.

Corollary 2.6.1 (Monotonicity of Modification Generating Functions)**.** *A plan refinement space that is spanned by plan modifications from a set of sound generating functions is acyclic.*

Proof. This result follows directly from the previous theorem: Given that a set of sound modification generating functions is monotonic, a plan cannot be reconstructed by any sequence of plan modifications from that set such that the very first applied plan modification would become applicable again. This implies that neither can the original plan as such be restored. The argument holds for all plans on all paths in the refinement space, which proves the above proposition. □

With the previous theorem and corollary it is shown that every sound plan modification generating function not only has disjoint applicability and obtainability sets, but also that any chain of modification applications will finally result in a plan from the difference of these sets.

We are now ready to address the initial question of this section, namely: Which conditions qualify a class of modifications to be "appropriate" for a class of flaws? A reflection upon the possible relationships of the equivalence classes gives the answer: Regardless of their meaning, a modification can only address the problem that underlies a flaw[12] if it is applicable in the flawed plan, if it affects the plan components referenced by the flaw, and if the plan that results from applying the modification is not flawed anymore.

[12] That means, a modification addresses the flaw semantically rather than in the syntax-based notion of Def. 2.31 regarding modification generator soundness.

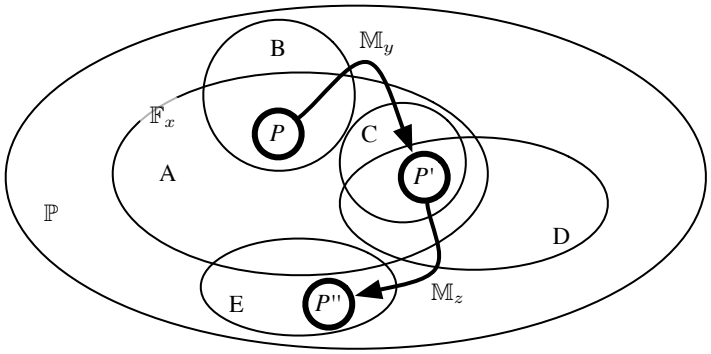


Figure 2.10: Appropriateness of plan modifications: The relation of flawed plans, applicability sets, and obtainability sets.

Lifting this criterion to the class level leads to the statement that a modification generating function is appropriate, if it can generate modifications that are applicable in the flawed plan class and that refine these plans into equivalence classes that are disjoint from the flawed plan class.

But we also have to take into account that the modification generation functions may be specialized to a point at which an *immediate* flaw resolution is not possible. That means, that one or more plan modifications are required to manipulate the flawed plan in such a way that it enters the applicability set of a resolving modification. Lifted to the class level, the definition can be extended: In addition, a modification generating function is appropriate for a class of flaws, if it can generate modifications that are applicable in the flawed plan class and that refine these plans into equivalence classes that have a non-empty intersection with the applicability set of an appropriate plan modification class. Such indirectly appropriate modifications move flawed plans into a direction where the defect becomes immediately resolvable.

Fig. 2.10 illustrates the situations of directly and indirectly appropriate modifications: Set A is the set of all plans that are characterized as being flawed by flaw class $\mathbb{F}_x \subseteq \mathbb{F}$. The class of plan modifications $\mathbb{M}_y \subseteq \mathbb{M}$ has B as its applicability set and C as its obtainability set, class $\mathbb{M}_z \subseteq \mathbb{M}$ has respective sets D and E. For the sake of readability, the applicability and obtainability sets do not overlap for the individual classes in the displayed example and the figure focuses on the necessary relevant intersections. Plan modification class z is directly appropriate, because it includes instances $\mathtt{m}_z$ that can be applied to a plan $P' \in A \cap D$ such that $\text{app}(\mathtt{m}_z, P') = P''$ with $P'' \in E \setminus A$. Regarding plan modification class y, there is no possibility to leave A directly. However, if there are plans $P \in A \cap B$ for which f_y^{mod} provides modifications $\mathtt{m}_y$ such that for the refinement $\text{app}(\mathtt{m}_y, P) \in C \cap D$ holds, y can be considered to be indirectly appropriate, too. The figure can be read as follows: The flawed Plan P is first modified by $\mathtt{m}_y$ into a refinement P' that is still flawed and afterwards modified by $\mathtt{m}_z$ into P'' in which the flaw is finally resolved.

The following function definition gives a more formal representation to the appropriateness-relation between flaw and plan modification classes. It makes explicit that a member of the former can be resolved *in principle* by a member of the latter, thereby enabling us to trigger the generation of specific modifications on the occurrence of specific flaws.

Definition 2.34 (Modification Triggering Function)**.** Given a class of plan modifications $\mathbb{M}_y \subseteq \mathbb{M}$ and a class of flaws $\mathbb{F}_x \subseteq \mathbb{F}$, $\mathbb{M}_y$ is called *suitable* for $\mathbb{F}_x$ if and only if for some planning problem π there exists a partial plan P that is flawed by a $\mathtt{f} \in \mathbb{F}_x$ and which a plan modification $\mathtt{m} \in \mathbb{M}_y$ exists such that $P' = \text{app}(\mathtt{m}, P)$ does not contain $\mathtt{f}$.

Furthermore, given a set of plan modification classes, a class $\mathbb{M}_y$ is suitable for flaw class $\mathbb{F}_x$, if its applicability set has a non-empty intersection with the affected set $\mathscr{P}_{\mathbb{F}_x}$ (Def. 2.27) and if in addition, the set contains a modification class $\mathbb{M}_{y'}$ that is suitable for $\mathbb{F}_x$ such that the obtainability set of $\mathbb{M}_y$ has a non-empty intersection with applicability set of $\mathbb{M}_{y'}$.

The *modification triggering function* is the function $\alpha : 2^{\mathbb{F}} \rightarrow 2^{\mathbb{M}}$ that relates classes of flaws with their suitable modification classes, given a set of plan modification classes, respectively a set of corresponding modification generating functions. •

It has to be emphasized that the triggering function deals with *class relationships* and does not perform analysis on the level of individuals. This means, that given a class of flaws $\mathbb{F}_x$ and a class of modifications $\mathbb{M}_y$ with $\mathbb{M}_y \subseteq \alpha(\mathbb{F}_x)$, the triggering function does not have the competence or knowledge to anticipate whether an appropriate concrete modification $\mathtt{m} \in \mathbb{M}_y$ for resolving a concrete flaw $\mathtt{f} \in \mathbb{F}_x$ in a given plan and problem actually exists or not. The reverse implication is however universally applicable: All modification individuals in $\mathbb{M}_y$ that are not covered by the triggering function are *guaranteed* not to contribute to a solution of any individual of $\mathbb{F}_x$ for any given plan and problem.

Definition 2.35 (Critical Flaws)**.** A flaw class $\mathbb{F}_x$ is called *critical* if for every plan in its affected set $P \in \mathscr{P}_{\mathbb{F}_x}$ the intended system behaviour collapses, that means, if $\text{beh}(P) = \emptyset$. Consequently, $\alpha(\mathbb{F}_x) = \emptyset$ in all domain models, because no refinement will be able to resolve the deficiency that is characterized by $\mathbb{F}_x$. •

An example for a critical flaw class is the problem of inconsistencies in the variable constraints. It is easy to see that the intended system behaviour any plan P is empty and that no refinement can exist. Such a flaw class is also referred to as "plan inconsistency".

The practical impact of the triggering function is an explicit representation of the control flow in a system that implements the refinement-based planning framework. Such a plan generation procedure basically relies on the triggering function as the criterion which modification generation function to choose from a given set in order to process a detected flaw. With the background of the above results it can thereby rely on the fact that every pursued generation option in the refinement space brings search closer to a point at which the current plan turns out to be a solution or a fruitless effort.

In the latter situation, the triggering function can be incorporated into a formal rejection criterion for discarding plans that cannot be refined into a solution. If there exists at least one flaw in a given plan for which either according to the triggering function no suitable modification exists or for which none is issued by a generator then the currently flawed plan cannot be developed into a solution. It has to be emphasized that it does not matter whether or not other flaws are present in a plan that can be solved, because flaws do *persist* over modifications that do not address them. In this sense, a set of sound detection functions and a set of sound modification generating functions span a search tree for which the *upward solution* property holds. From the point of view of the intended system behaviour, planning deals with refining an abstract solution into a concrete one. The arguments for deciding on the downward refinement property therefore hold for our notion of plan refinements [13].

Definition 2.36 (Rejection Criterion)**.** Let $\mathfrak{Det}$ be a set of sound detection functions and $\mathfrak{Mod}$ a set of sound modification generating functions. Let α be a triggering function such that for any $f_x^{det} \in \mathfrak{Det}$ all functions in the set $\{f_{y_1}^{mod}, \ldots, f_{y_n}^{mod}\} \subseteq \mathfrak{Mod}$ are regarded appropriate, that means $\mathbb{M}_{y_1} \cup \ldots \cup \mathbb{M}_{y_n} = \alpha(\mathbb{F}_x)$. For any partial plan P and problem π we define the *rejection criterion* to be

$$\bigcup_{1 \leq i \leq n} f_{y_i}^{mod}(P, f_x^{det}(P, \pi), D) = \emptyset \text{ for } f_x^{det}(P, \pi) \neq \emptyset$$

•

With the functional modularization for the detection of flaws and the generation of appropriate plan modification steps, with the formulation of an explicit flow control mechanism for relating flaws and plan modifications, and finally with the definition of precise solution and rejection criteria for the examined plans, a simple refinement-based planning procedure can be formulated: the *simplePlan* procedure of Algorithm 2.1.

The algorithm takes as an input the current plan P and a problem specification π. The two sets for flaw detecting and modification generating functions, $\mathfrak{Det}$ and $\mathfrak{Mod}$, are assumed to be globally accessible from inside the procedure. For now, $\mathfrak{Det}$ is furthermore assumed to be a complete set of sound detection functions.

Algorithm 2.1 A simple generic refinement-based planning algorithm.

Require: Sets of detection functions $\mathfrak{Det}$ and modification generating functions $\mathfrak{Mod}$

1: **simplePlan**(P,π)**:**
2: $F \leftarrow \emptyset$
3: **for all** $f_x^{det} \in \mathfrak{Det}$ **do**
4: $\quad F \leftarrow F \cup f_x^{det}(P,\pi)$
5: **if** $F = \emptyset$ **then**
6: $\quad$ **return** P
7: $M \leftarrow \emptyset$
8: **for all** $F_x = F \cap \mathbb{F}_x$ **do**
9: $\quad$ **for all** $\mathtt{f} \in F_x$ **do**
10: $\quad\quad$ **for all** $f_y^{mod} \in \mathfrak{Mod}$ with $\mathbb{M}_y \subseteq \alpha(\mathbb{F}_x)$ **do**
11: $\quad\quad\quad M \leftarrow M \cup f_y^{mod}(P,\mathtt{f},D)$
12: $\quad$ **if** $\mathtt{f}$ was un-addressed **then**
13: $\quad\quad$ **return fail**
14: **return** simplePlan(app(**choose**$(M),P),\pi$)

Line 2 initializes the set of all flaws f that are found in P. The first loop (lines 3-4) polls every available detection function and stores the announced deficiencies in the F-set. If the plan is found flawless, that means, if the solution criterion is met (Def. 2.5), it is returned as a solution (lines 5-6).

The second part of the procedure accumulates the modification steps flaw-wise from the responses of the available modification generators. It groups the discovered flaws according to their classes (loop in line 8) and then passes each individual (loop in line 9) to each appropriate plan modification generating function (lines 10-11). Since the detection functions are assumed to be sound (see Def. 2.28) and complete (see Def. 2.29), if a single flaw is found un-addressed and the rejection criterion is met (see Def. 2.36), the algorithm returns a failure (lines 12/13).

When finally all flaws have been answered, one of the generated plan modification steps is non-deterministically chosen (line 14). This is also the backtracking point of the algorithm. The modification is applied to the current plan and the algorithm is recursively invoked with the application's result. The non-deterministic step is practically realized in the usual way: when a recursive call on a refinement with the chosen modification returns a failure, another recursion is performed with a new plan modification; if all alternative plan modifications in M have been tried out unsuccessfully, a failure is returned from that recursion level back to a previous choice point.

Algorithm 2.1 obviously performs a depth-first search in the plan refinement space, and therefore the termination of the procedure becomes an issue. But we have seen in Theorem 2.3 and the related corollary that the plan space induced by plan refinements from sound generating functions is finite, in particular every path in it is cycle-free. If $\mathfrak{Mod}$ is a set of sound modification generating functions, *correctness* and *termination* of the simplified procedure are consequently guaranteed. Since the algorithm cannot descend into infinite paths, it performs an exhaustive exploration of the refinement space and hence becomes a *complete* search procedure. That means, if a flawless refinement of the problem specification – a solution – exists, the algorithm will eventually find it. Note that the algorithm performs a systematic search over plan refinements, it does however not perform systematic planning in the strong sense of [177], because isomorphic or symmetric sequences of plan refinements may yield the same plans (see also discussion in Sec. 2.8).

It is an important feature of the presented approach that an *explicitly stated* triggering function allows to separate the computation of flaws completely from the computation of modifications, and that in turn both computations can be separated from search-related strategic issues (represented by the non-deterministic choose operation which path to follow in the refinement space). A planning system that implements this sketched "architecture" can benefit from this separation in two ways: first of all, function invocation and interplay, that is, the control flow is specified by α, which enforces a strong modularization of flaw and modification generators and hence facilitates the realization of configurations and their implementation (see

Chap. 3 and Sec. 5.1, respectively). The second benefit lies in the availability of all deficiency and modification options at the time of making a strategic decision and in the analyzability of these options. The declarative structure of the collected flaws and modifications (sets F and M) allows for an explicit reasoning step about search (line 14) that can be performed on the basis of flaws and modifications without taking their actual computation into account. The issue of identifying which corrective measure on the plan can eliminate which kind of flaws is already settled at that point. Many counter examples exist in the literature where due to the lack of the discussed features the resulting planning procedures persist in a fixed flaw and modification selection schema; Sec. 4.1 will discuss this issue and introduce more liberal, so-called *flexible* search methods, which can be realized in this framework.

We will see how our approach enables opportunistic search strategy decisions on a plan-for-plan basis. Naturally, the non-deterministic function **choose** would serve as the appropriate entry point for a planning strategy. Since the implied search schema of the generic algorithm is limited to a depth-first procedure, providing a clever modification choice alone seems however not sufficient. The search strategy will therefore be divided up into three compartments: one function for the assessment of the possibly many modifications for dealing with the detected problems in the plan at hand. This option evaluation is performed in a local view. The second component is responsible for the global view on the refinement space, it is a function for the selection of the path in the plan-refinement space that is to pursued. The third function finally decides whether a satisfactory set of solutions has been obtained so far or not.

We begin with the definition of a strategic function that selects all plan modifications that are considered to be worthwhile options, thereby determining the ordered set of successors for the current plan in the plan refinement space. The following function controls the branching behaviour of the refinement-based planning algorithm.

Definition 2.37 (Modification Selection Function)**.** A strategic *modification selection function* prioritizes the elements from a sequence of plan modifications. It is described as a function

$$f^{modSel} : \mathscr{P} \times 2^{\mathbb{F}} \times 2^{\mathbb{M}} \rightarrow 2^{\mathbb{M} \times \mathbb{M}}$$

that selects plan modification steps from its third argument and returns them as a partial order on the carrier set $\mathbb{M}$ for their application to the plan in the first argument. Those plan modifications that do not occur in an element of the result's relation tuples (even not in an reflexive tuple) are regarded as discarded refinement options. •

The idea is to interpret the selection result as a preference relation that induces a linear order on the plan modifications. The necessary linearization will thereby project the modifications that are available in the current plan onto a sequence of plan modifications that is consistent with the preference relation. The actual "selection" is performed on that sequence afterwards. If not mentioned otherwise, the result tuples of a modification selection function contain *all* plan modifications that have been passed. Later sections will discuss the influence of this function on the characteristics of the resulting integrated search procedure and how it can be used to safely skip refinements in a way that does not effect a loss of solutions.

The second aspect of search control concerns the selection of those plans that are next to be processed by the detection and modification generating functions. In other words, concrete implementations of the following function are responsible for the general search schema, ranging from un-informed procedures like depth-first, breadth-first, etc., to informed, heuristic schemata and the like.

Definition 2.38 (Plan Selection Function)**.** A strategic *plan selection function* prioritizes the elements from a sequence of plans. It is described as a function

$$f^{planSel} : \mathscr{P}^* \rightarrow 2^{\mathscr{P} \times \mathscr{P}}$$

and it returns the selected plans as a partial order, that means, tuples over the carrier set $\mathscr{P}$. Those partial plans of the input sequence that do not occur in an element of the result relation (even not in an reflexive tuple) are thereby discarded. •

Plan selection functions are interpreted as generators for preference relations like the above modification selection functions. That means, the selection result will be projected on a sequence of plans that

is consistent with the preference relation. Again, unless otherwise noted, we assume that the selection result includes *all* plans from the input sequence. The realization of a number of exemplary concrete search schemata is discussed at the end of this chapter and detailed in the subsequent chapter, respectively.

The simplified generic refinement planning algorithm returns the first solution to the given problem that is encountered during plan space exploration. This is not always desired, for example, in situations where a solution is sought that satisfies a certain quality measure or where a certain number of alternative solutions has to be obtained. For these cases, we provide a mechanism for constructing more than one solution, respectively for indicating when a set of solutions is or contains a satisfactory one.

Definition 2.39 (Solution Selection Function)**.** A strategic *solution selection function* is a Boolean function over sequences of plans. It is characterized as follows:

$$f^{solSel} : \mathscr{P}^* \rightarrow \{\text{true}, \text{false}\}$$

The function returns true, if the sequence of solutions contains solutions of a specific quantity or quality and false otherwise. •

Two common instances of a solution selection function are, first, the one being satisfied with any solution (the first identified solution is going to be returned), and second, the greedy one that is never satisfied (making an algorithm to return all possible solutions for a given problem). Again, the interplay with a search procedure and the fitting into the bigger picture of an extended refinement planning algorithm will be discussed later.

The presented interaction of detection and modification generating functions in Algorithm 2.1 assumes that plan generation only deals with the concept of strategic *choice* in the planning process: detections report plan deficiencies, which are expected to demand in the majority of cases for a number of alternative modifications to be prepared for their resolution. But apart from that, also those situations have to be addressed in which further information can be deduced directly from the current plan alone and its establishing is not subject to choice and therefore not subject to a search strategy. In the view of plan refinement via plan modifications, this translates into a need for specific "immediate" modifications in situations where no alternative choices are given. This kind of modification does not need the detection of a respective flaw and is not to be selected in a strategic decision process. This procedure can also be interpreted as a kind of rule-based reasoning mechanism, which provides a modular and convenient way of implementing inferences per se. In particular, the inferred knowledge can be naturally shared by the functions participating in the plan generation process, because it explicitly manifests in the plan. A rule-based-like modification establishment thus provides much flexibility for the actual implementations of detection and modification generating function.

Hence, for all inference tasks on the plan that are not subject to *choice* we define inference rules in the following way:

Definition 2.40 (Inference Function)**.** An *inference function* is a function that computes plan modifications of a given type z that represent necessary changes to a partial plan (under a given domain model):

$$f_z^{inf} : \mathscr{P} \times \mathscr{D} \rightarrow 2^{\mathbb{M}_z}$$

The plan modifications are the usual ones as introduced in Def. 2.20. Inference functions may return modifications of shared types with modification generating functions. •

Several of the implementations that will be presented in Chap. 3 confirm the practical impact of this procedure. The delegation of specific reasoning tasks to the inference functions appears to be very convenient for most configurations that implement our refinement-based approach. Their application ranges from synchronizing constraint sets in advanced planning configurations (Sections 3.3.1 and 3.3.3) to deriving user-specific information for plan readability (3.5.5) and *mixed initiative* interfaces (7.2.2). Ambiguous inferences, that means, inferences that involve points of choice have to be modelled via corresponding, in some sense "artificially" introduced, pairs of detection and modification generating functions with their specific inference-related flaws and modifications.

As inference functions avail themselves of the same repertoire of plan modification steps as the plan modification generating functions do, their soundness is defined in a similar way (cf. Def. 2.31).

Definition 2.41 (Soundness of Inference Functions)**.** An inference function f_z^{inf} is called *sound* if and only if it generates proper plan modifications that have a positive plan component balance. More precisely, for any partial plan P over a given domain model D the following holds:

1. The result of calling the inference function $f_z^{inf}(P,D)$ is a (possibly empty) set of proper plan modifications with respect to P.
2. For every generated plan modification $\mathtt{m} = \langle E^{\oplus}, E^{\ominus} \rangle \in f_z^{inf}(P,D)$, more plan components are added than removed: $|E^{\oplus}| > |E^{\ominus}|$.

•

Soundness is an issue because the inferences are going to be calculated iteratively until their inferential closure is reached. Note that the result of Theorem 2.6 can be formulated analogously for inference functions. Sound inference functions are consequently monotonic like sound plan modification generators are. Calculating the inferential closure is therefore guaranteed to terminate. It is also worth pointing out that establishing soundness is a relatively easy task (see inference functions of respective framework implementations in Chap. 3).

2.7 A Generic Refinement Planning Algorithm

With the defined strategy functions and inference mechanisms, we can now precisely formulate a more powerful and adaptable generic planning algorithm. It will be used for the remaining parts of this thesis[13] as the core component of any planning system that is going to be implemented within our architecture. For an overview, Fig. 2.11 sketches the algorithm: Central to the procedure is the cloud-shaped plan data structure. In a first phase, inference modules will operate on it to deduce implicit information. After that, detection functions issue their flaws, which in turn are filtered by the triggering mechanism. At this point, a plan may be found un-flawed and chosen for a solution. The triggering function otherwise dispatches the flaws to the suitable modification generating functions that send their plan modification proposals to the respective strategic selection function. The chosen modifications generate the successors of the current plan in the search space, and finally the plan selection function updates the system's focus according to the plan in the search space fringe that has been chosen to pursue.

Let us now proceed to the concrete algorithmic procedure of the refinement-based planning framework; it is specified as Algorithm 2.2.

The prerequisites for the execution of the algorithm are defined sets of detection functions, modification generation functions, and inference functions. In addition, the strategy functions have to be declared. The procedure itself takes four arguments: the sequence of plans that are subject to examination, the employed domain model, the current problem specification, and a sequence of plans that have qualified as solutions to the initial problem so far. When the algorithm is called, it uses the first plan in the list of un-processed plans as the "current plan". The literature names this input list alternatively *fringe* [148], *agenda* [279], or in more general search frameworks *open node list* [223].

The body of the algorithm is basically divided into five sections: A termination or goal test, an inference loop, a flaw collection loop, the generation of modifications according to the issued flaws, and finally the strategic choices of which modifications to consider and how to proceed in the search space. The algorithm uses a function *linearize* to compute linear sequences of plans, respectively plan modifications that are consistent with the partial orders obtained from the appropriate strategic selection functions.

[13] Sec. 5.1 will introduce a slightly enhanced concurrent modification of this algorithm. These enhancements, however, merely address implementation-specific issues. Conceptually, the planning procedure in the following already exhibits the entire functionality.

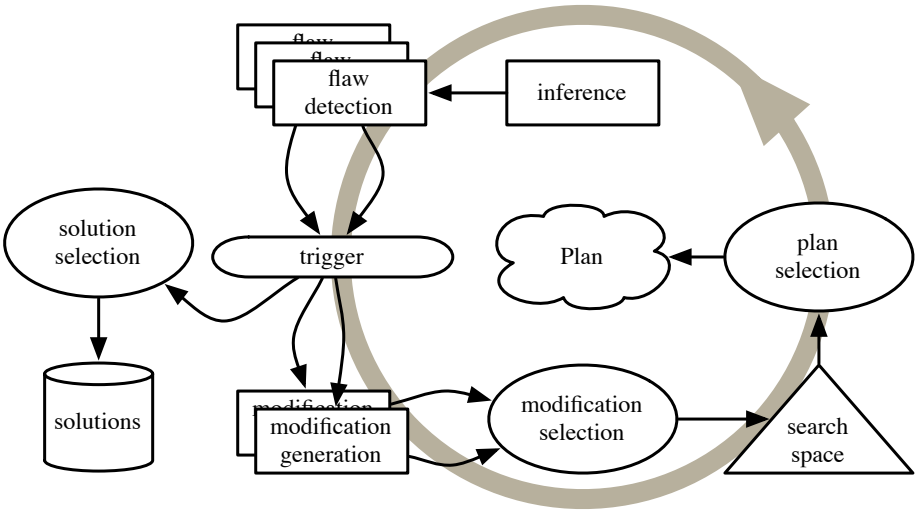


Figure 2.11: A diagrammatic refinement planning process. The intended flow of execution is counter-clockwise.

Algorithm 2.2 The definitive generic refinement planning algorithm.

Require: Sets of functions $\mathfrak{Det}$, $\mathfrak{Mod}$, and $\mathfrak{Inf}$
Require: Selection functions f^{modSel}, $f^{planSel}$, and f^{solSel}

plan$(P_1 \dots P_n, \pi, P_{s_1} \dots P_{s_m})$**:**
if $n = 0$ **or** $f^{solSel}(P_{s_1} \dots P_{s_m}) =$ **true then**
  return $P_{s_1} \dots P_{s_m}$
repeat
  for all $f_z^{inf} \in \mathfrak{Inf}$ **do**
    for all $\mathtt{m} \in f_z^{inf}(P_1, D)$ **do**
      $P_1 \leftarrow \text{app}(\mathtt{m}, P_1)$
until P_1 did not change
$F \leftarrow \emptyset$
for all $f_x^{det} \in \mathfrak{Det}$ **do**
  $F \leftarrow F \cup f_x^{det}(P_1, \pi)$
if $F = \emptyset$ **then**
  return plan$(\mathit{linearize}(f^{planSel}(P_2 \dots P_n)), \pi, P_{s_1} \dots P_{s_m} P_1)$
$M \leftarrow \emptyset$
for all $F_x = F \cap \mathscr{F}_x$ **do**
  for all $\mathtt{f} \in F_x$ **do**
    for all $f_y^{mod} \in \mathfrak{Mod}$ with $\mathscr{M}_y \subseteq \alpha(\mathscr{F}_x)$ **do**
      $M \leftarrow M \cup f_y^{mod}(P_1, \mathtt{f}, D)$
    if $\mathtt{f}$ was un-addressed **then**
      return plan$(\mathit{linearize}(f^{planSel}(P_2 \dots P_n)), \pi, P_{s_1} \dots P_{s_m})$
ExtFringe $\leftarrow \emptyset$
for all $\mathtt{m}$ in $\mathit{linearize}(f^{modSel}(P, F, M))$ **do**
  ExtFringe $\leftarrow$ ExtFringe $\circ$ app$(\mathtt{m}, P_1)$
return plan$(\mathit{linearize}(f^{planSel}(\text{ExtFringe} \circ P_2 \dots P_n)), \pi, P_{s_1} \dots P_{s_m})$

Termination (2-3): If the agenda is found empty, that means, if no more plans in the fringe are due to examination, or if the solution selection function is satisfied with the solutions found so far, all collected solutions are returned and the algorithm terminates.

Inference (4-8): The algorithm iterates over all inference functions and applies their modifications to the current plan P_1 until the inferential closure is reached and no more modifications are issued. The inference modifications are executed immediately and hence bypass the defined strategy functions.

Flaw Detection (9-13): Similar to the simplified Algorithm 2.1, the results of all deployed detection functions $\mathfrak{Det}$ are collected. If no deficiency can be spotted, the current plan is considered to be a solution to the given planning problem (see solution criterion in Def. 2.5) and is therefore added to the sequence of solutions in the subsequent recursive call of the planning procedure (line 13). Before the recursive descent, the plan selection function determines the next current plan and modifies the first argument, the agenda, accordingly.

Modification Generation (14-20): The applicable modification steps are accumulated per flaw class and per class instance from the set of available modification generators $\mathfrak{Mod}$ according to the α-defined assignments. If any flaw is found un-addressed by its associated modification generating functions, the current plan is discarded and the algorithm is called recursively with a newly selected current plan candidate (see rejection criterion in Def. 2.36).

Strategy (21-24): All plan modifications that pass the strategic modification selection function (line 22) are applied to the current plan and thereby constitute the set of its refinements, that is, the set of its successor plans. This fringe extension is established by the strategic decisions in f^{modSel} and inserted at the beginning of the fringe. The plan selection function finally chooses the next current plan and the procedure is called recursively.

The generic refinement planning algorithm is initially called as follows: $\text{plan}(\pi.P_{init}, \pi, \varepsilon)$.

This algorithm obviously subsumes the simplified version Alg. 2.1 because it can reproduce its behaviour by employing the following concrete function definitions:

- $\mathfrak{Inf} = \emptyset$

  The simplified version does not feature any inferences.

- The modification selection function f^{modSel} returns for every call a complete linear order of plan modifications that corresponds to an *iterated call* to the previously deployed non-deterministic function **choose**. Together with the plan selection function, this sequence "anticipates" the decisions of the simplified algorithm during backtracking.

- $f^{planSel}(P_1 \ldots P_n) = \{(P_i, P_{i+1}) | 1 \leq i \leq n-1\}$

  Because new nodes in the search space are added to the beginning of the fringe, selecting the first plan corresponds to a depth-first search schema. The fringe is preserved in order to leave all decisions to the modification selection function.

- $f^{solSel}(P_1 \ldots P_n) = \begin{cases} \textbf{true} & \text{for} \quad n = 1 \\ \textbf{false} & \text{else} \end{cases}$

  Algorithm 2.1 always returns the first solution occuring in the fringe.

It is an essential feature of the presented algorithm that it does not depend on the participating function sets, since the options to address existing flaws by appropriate plan modifications is defined via the α triggering function. In addition, Chap. 4 will demonstrate that even the strategy functions themselves can be designed completely independent from the deployed refinement mechanics. It is also noteworthy that no backtracking point has to be provided in this algorithm, because alternative unexplored choices are stored in the agenda for all depth-levels of the search space simultaneously.

2.8 Discussion and Perspective

2.8.1 General Comments

The syntax and semantics of our formalism have their origins in several strains of planning-related areas of work, which we would like to give proper credit on this occasion: The non-hierarchical aspects of our action representation incorporate ideas from the well-founded ADL action formalism [211–213]. Some semantic concepts, in particular the application of the weakest precondition to operator effects, are originated in deduction-based refinement planning as presented in [247, 248], with additional influence of work in dynamic predicate logic [128, 250]. The formalization and handling of term updates in the action specifications is done analogously to the axiom of assignment (D0) of the Hoare calculus [135]. Since it does neither support hierarchical concepts (any more) nor does it build on a suitable semantics [32, 38, 60, 181], using the de-facto standard language for the International Planning Competition, the Planning Domain Definition Language PDDL [80, 99, 111, 187], was not an option (in any of its versions).

For the sake of a uniform presentation, we have decided not to separate the logical language from a dedicated planning language. We point out, that this deviates from our previous work in [26], where language $\mathscr{L}$ had been restricted to the logical symbol set components, while the planning framework employed a separate planning language that added the syntactical elements for describing plan components, that means, task and elementary operation symbols. This change has no further consequences on the presented results, but a reader who is familiar with the respective literature might be slightly confused if unaware of the adaptation.

Another more general issue that is worth noting arises in the definition of solutions to planning problems (Def. 2.17). We have not addressed the question whether or not a solution to a planning problem necessarily has to be a primitive ground plan. All our definitions and arguments suggest that they have to and that the executable leaves in a refinement space are populated by them. Regarding the executability of partial plans in a given state (Def. 2.11), however, we only require all the primitive ground linearizations of plan to be executable in that state. That means, that our framework is, strictly speaking, able to produce *parametrized plans with abstract actions*. In the same way the non-linear planning systems present a partially ordered plan with the argument that every possible linearization of it achieves the goal, our approach could leave task parameters unbound if every valuation would result in a consistent course of action. Analogously, if every implementation of an abstract action "worked", the solution would be presented in terms of the appropriate complex tasks. There exists however an implementation aspect that has to be taken into account: computing *any* most specific solution is considerably easier than assessing all options with respect to a variable binding, not to mention an abstract action. We believe that the concept of parametrized abstract plans requires some more support from the problem specification side (preference of task schema x to stay abstract, preference of parameters of sort Z instead of concrete instances) before it can be applied in a general planning context. Furthermore, the connections to the field of learning, for instance building generalized plans [91], have to be examined carefully.

It has also to be noted that our generic refinement algorithm (Alg. 2.2) resembles generic POCL algorithms like that in [288]. This particular procedure also uses an explicit strategy-like function for selecting the most relevant flaw in a plan and on for selecting the next plan to operate on. However, authors like Williamson and Hanks never make the modification alternatives explicit.

2.8.2 Declarative Semantics and Consistency

Our approach deploys a relatively broad logic-based formal foundation and in particular realizes state abstraction by introducing non-logical axioms. The consequently required reasoning mechanisms are in general computationally demanding and therefore often avoided in implemented systems. But as pointed out by several authors ([180], amongst others), domain models need to have a *declarative semantics* independent of domain heuristics or planning algorithm. The reasons for this do not only lie in theoretical properties like a sound foundation and explicit consistency criteria but they are also a result of experience with fielded planning systems, for instance, as reported in [194]. Setting up and maintaining consistency in planning

domain models becomes a serious problem in any mission-critical application and plays a major role not only for the question whether or not all plans that are generated by the respective system are correct, but also for cases in which (unexpectedly) no plan can be found [57].

It is clearly a considerable advantage of using declarative models and clear semantics that all these issues can be addressed adequately. In fact, we have developed a prototypical domain model editing tool that verifies many of the presented consistency criteria and that is able to notify the modeler about syntactical as well as semantic errors. In addition, as we have pointed out in various comments in this chapter, we also search for questionable model anomalies and un-referenced parts of the model (for example, unused sorts and predicates), and the like. Bringing this tool support to perfection, in terms of both validation functionality and usability, lies clearly in the focus of future developments (cf. Sec. 7.2.3).

In addition to the efforts concerning the general planning-centered notion of model consistency, future research will have to deal with domain-specific consistency requirements. That means, we want to deduce or verify statements about the plans that can be constructed on the basis of the examined domain model. A typical application scenario is to check whether certain security-relevant model invariants hold, for example whether safety-constraints are satisfied in all states in all constructable plans. These techniques are usually found in deductive planning [27], our well-founded approach does however allow for such formal methods as well. This is a strong contrast to domain model validation via simulating plans [79], and the like.

2.8.3 Non-Limiting Restrictions

The formal framework complies to some premises that imply limitations at first glance, while on closer examination they do not.

Firstly, our logical models $\mathscr{M}$ are restricted to *natural ones*, which in particular entails that only a finite set of objects can be represented. Note that although there is of course a significant difference between natural and un-restricted, infinite domains, but for practical purposes, we do not depend on them. Any implementation of our framework that requires an "infinite" number of objects, for example the natural numbers $\mathbb{N}$, is physically bound by the computer hardware to get on with a finite number of representable symbols. The same argument holds for the application domain as well: There is of course an infinite number of possible values for a point in time, but any model or knowledge base discretizes time into a finite amount of manageable slices, hour time-slots for example. We may therefore safely provide a pool of constants that is just "big enough" or generate the finite number of required constants on demand. Note that every computer uses the latter technique in order to handle real numbers $\mathbb{R}$.

The second restriction appears to be the fact that our formalism being based on the semantics of operator *sequences*. Parallelism as such is not addressed explicitly by our approach. We do however adopt the notion of non-linear planning, that any un-ordered actions in a solution may be executed in parallel. This is legitimate because if two actions interfered according to their preconditions and effects, the system would introduce an ordering between them. Note that this also means that, in compliance with the domain model, one action does not affect the *invariant* parts of the other action's preconditions during the execution. An extension of our framework in order to represent parallelism explicitly is part of future work. For example, additional temporal constraints have to be introduced that symbolically demand for an overlapping execution of two plan steps or that indicate that two steps should never be executed in parallel. We are confident that only a small amount of selected changes becomes necessary to the semantic foundation. Parallelism can be represented as an arbitrarily small difference of time that passes between the beginning of two operator executions. From an observer's point of view it becomes thereby undecidable which action "happened" first (pseudo-parallelism). The semantics of an operator's execution is consequently re-formulated in order to cover overlapping elementary operations, and with that, the state transition model is complete again and subsequent result are restored.

2.8.4 State And Action Abstraction

In contrast to our axiomatic approach, conventional state-abstraction mechanisms generalize by *blending out* less relevant esteemed facts in state descriptions [153, 224, 294]. This technique has obviously less expressive power and the additional drawback that a modeler has to decide (again, without semantic assistance) which subset of the state features is an adequate carrier of *all necessary* information on the higher abstraction level. There exists the so-called false proof problem (together with some interesting observations concerning abstraction in [122]): If an abstract problem space is formed by dropping conditions from the original space, information is lost and axioms in the abstract space can be applied in situations in which they cannot be applied in the original space. It is thus difficult to guarantee that if there exists an abstract solution, there exists also a solution in the base space (cf. downward solution property of HTN planning [13] and ordered monotonicity [155]). Representing state abstraction by (un-) folding state automata [46, 180] has in general a clearer semantics and a greater expressiveness. Since state automata are an established modelling technique, there exists some tool support for building hierarchical domain models in that paradigm [182].

The computational and methodological efforts for building consistent state-abstraction axiom sets are relatively small. Well-formedness of the involved literals and equations can be decided easily, and so can the correct usage of rigid and flexible literals. During axiom construction, refinements can be restricted to the proper atoms, respectively terms. It can also be done off-line by unfolding the axioms (substituting all atoms by their refinements) and checking well-formedness of the explicit refinements. Deciding satisfiability is a harder task in the general case, but the structure of the axioms allows for an efficient reasoning about contradicting refinements.

Sets of state-abstraction axioms can also be seen as *stratified* axiom sets like in the axiomatic extension of the Planning Domain Definition Language PDDL, for which [264] has shown that very efficient reasoning procedures can be provided and that the usage of axioms enhances the expressivity of the formalism and the efficiency of the planning software likewise (the planning algorithms referenced in [264] do however not apply abstraction mechanisms). The purpose of an axiomatic extension of PDDL is to provide compact domain models and describe simple inferential knowledge, the so-called "derived predicates" (cf. "Axioms" in the original PDDL specification [187] and derived predicates in PDDL 2.2 and higher [80]). Consequently, the stratified axiom structure is used to compile efficiently modified operator descriptions in which basically every usage of an atom is recursively substituted by the atoms of associated strata.

The idea of stratification is to recursively "build" derived literals from basic ones, which consequently requires these axioms to adhere to a specific structure, namely the negation normal form (negation occurs only in literals), and that predicates occuring in derived literals in a given stratum never occur in lower strata. The latter allows simple recursion but prevents cyclic definition beyond stratum levels. Such a structural consistency criterion is not explicitly demanded in our Definition 2.3, since we are only interested in logical satisfiability of the axiom set. A modeler is however well advised to adopt a "layered" style of state hierarchies for the sake of conciseness. In the view of our abstraction mechanism, a stratum *roughly* corresponds to one "level of abstraction" with respect to the involved state features. According to the example state-abstraction axiom depicted in Fig. 2.3, literals over `Delivered` would be in the first stratum, and literals from the second stratum level would be derived predicates and hence less abstract in our context. It has to be noted, however, that there does not exist the notion of common abstraction levels or reference strata, except those state features for which no axioms are provided, the concrete level, and those that do not occur in any refinement, the abstract level. All features that do not belong to either, however, cannot be said to be more or less abstract than others.

2.8.5 Refinement-Based Planning and Search

Perceiving plan generation as refining an abstract specification of the desired solution is realized in a number of planning approaches, however mostly in an informal manner. Exceptions to this trend are deductive planning approaches and the refinement-planning framework introduced in [147]. The latter, Kambhampati's refinement-planning framework, follows the objective to develop a generalization of (all of the) planning algorithms that have been introduced in the literature. In this view, its primary objective is orthogonal to

our's, because the focus of our formal framework lies on generalizing from plan manipulations and therefore on generalizing from the actual planning functionality. Kambhampati's algorithmic frame is intended to paraphrase key elements that occur in the respective planning algorithms in order to evaluate alternative approaches and implementations to these elements, for example to solution extraction, to goal selection, and to consistency checking. The semantics is based on the view that the syntactic structure of a partial plan represents a set of so-called candidate plans, which are basically executable plans that include the components of the current partial plan. The notion of refinement is consequently to extend the plan structure systematically by inserting additional components for the purpose of satisfying goals, protecting established conditions, and verifying tractability, thereby defining a categorization of the refinements. Although the author addresses hierarchical refinements as well (also in [149]), the framework does not allow for a seamless integration like in our hybrid approach, because it does not provide a semantic foundation for action abstraction. To our knowledge, no integration of scheduling functionality is supported.

Concerning deductive planning, our semantic foundations share some concepts of the formalization presented in [248]. The main difference lies obviously in the fact that deductive planning uses the formalization of its refinement semantics for actually making the system operational. While formal methods are far superior to our framework implementations with respect to verifying consistency and proving propositions about domain-model specifications (see sections above) and although they enable advanced plan concepts like recursion and other control structures, strategic support and user-comprehensibility are major issues. As we will show in later chapters, our strategy concept enables a solid variety of effective planning strategies, in particular those which are capable of controlling search in implementations of our framework that deal with integrated planning and scheduling functionality. Making the data structures and plan generation choices explainable to human users is an important topic for future research (cf. Sec. 7.2.2).

The presented refinement-planning algorithm addresses a number of algorithmic issues: correctness, completeness, systematicity, and termination.

The **termination** of the algorithm is directly given by the refinement-planning principle. As long as no solution is found, the flaws trigger the plan modification generators. But this is a terminating process for two reasons: Due to the fact that once a plan modification succeeds in eliminating a flaw, that particular flaw will never return on that particular path in the search space (Theorem 2.4), and due to the fact that plan modifications manifest their monotonicity property (Theorem 2.6 and related corollary), the procedure will finally yield a partial plan that either satisfies the rejection criterion (Def. 2.36) or that exhibits no flaws and is considered a solution (Def. 2.5).

It is also easy to see that the planning procedure is **correct** in the sense that any plan that is claimed to be a solution is in fact one. Algorithms 2.1 and 2.2 rely in their correctness on the provided detection functions to be sound (Def. 2.28) – and in turn indirectly on the soundness of modification generating (Def. 2.31) and inference functions (Def. 2.41). If all sound detection functions remain silent then the current plan is a solution to the problem (Def. 2.5). Correctness does in particular not depend on any of the strategic functions, which can in this context be even defined as non-deterministic choices over modification and plan options.

The **completeness** of the algorithm has multiple dimensions in the presented framework. It is influenced by three completeness-aspects, namely

1. Completeness of plan modification families (Def. 2.23): does the spanned refinement space include all refinements that are solutions?
2. (Semi-) Completeness of modification generating functions (Def. 2.32): does each of the generator functions always return all possible refinements from the available repertoire?
3. Completeness of search procedure: does the procedure explore all solutions that are offered in the plan refinement space that is spanned by the plan modifications from the generator functions?

Complete families of plan modifications are typically never provided by the generating functions, for example, compare Sec. 1.1.3 for HTN planning as a paradigm in which solutions are seeked under explicitly defined refinement restrictions (namely refinements that correspond to the specified decomposition methods). The same holds consequently for complete sets of modification generating functions. Regarding their

semi-completeness, however, the realizations of the framework presented in later chapters do all recommend to deploy semi-complete generator functions. This is the case, because we want to make all available options visible to the strategy functions; and in order to conduct research on the plan development in the different configurations, we are interested in not missing solution development paths inherently. Concerning the completeness of the search procedure per se, this is (with semi-complete generator functions present) in general a question of the two strategic selection functions. Since the early days of planning, strategy functions and their algorithmic equivalents normally do not pursue all of the available refinement options (partial modification selection) nor do all resulting plans in the fringe ever get considered (heuristic driven plan selection). Completeness is in general not the primary objective and often sacrificed for the sake of gaining efficiency by eagerly cutting non-promising nodes in the search space.

The last aspect to be addressed is that of **systematicity** of search. Though the algorithm is systematic in the broader sense, that means, in contrast to local search algorithms it traverses the search space in a structured and organized way, it is however intrinsically susceptible to an un-systematic behaviour in a stronger notion. As defined in earlier work on partial-order planning systems [177], systematicity describes a search procedure that never visits the same plan, or even equivalent plans, twice. The therein presented UCPOP system, for instance, arranges every branch in the search tree such that "it divides the remaining possibilities of plan refinements into disjoint sets of potential solutions such that all equivalent plans are down the same branch of the search tree". The primary motivation for researchers to deal with systematicity is the, at first glance, self-evident advantage to avoid redundancy in the search space generated by the planner. Results of later work in the field of refinement planning [147, 148] include some quantification of the saved computational efforts, for example, that the fringe size of any search tree generated by a systematic planning procedure (that is, *strongly* systematic refinement search) is strictly bounded by the size of the potential solution candidate space.

Systematicity is undeniably a useful property of search, however tied to the refinements rather than the strategy; particular classes of plan refinements, applied according to a particular schema, can be proven to provide alternatives to a search strategy that are systematic in the strong sense. Consequently, our plan modification generating functions would have to be restricted to some less flexible and considerably more committing refinement generation schema or, alternatively, the strategy functions would have to be burdened with extremely costly plan space analyses in order to cut-off duplicates of previously encountered plans. It has however to be noted that systematicity is not considered mandatory for efficient plan generation. "Clearly, worst case search space size will have a strong correlation with performance only when the planner is forced to explore a significant portion of its search space in solving the problem. Since the goal of planning typically is to find one, rather than all, solution of a problem, unless the problem is unsolvable; or the solution density is low and the planner's initial choices lead it towards non-solution branches, systematicity may not necessarily lead to improvements in planning performance" [145]. And as it also has been noted by Mc Allester and Rosenblitt: "However, it seems likely that Tate's weaker notion of threat [that does not maintain systematicity] works just as well, if not better, in practice" [177]. This is related to a known result in the planning literature [166], and therefore much research has been targeted at finding a reasonable trade-off between redundancy and commitment. In aiming at providing maximal flexibility to the framework implementation, we decided to opt for complying with the least commitment principle and tolerating a reasonable amount of redundancy.

Concerning the concrete search procedure, and this issue will be picked up again in Chap. 4, the strategic trinity can implement arbitrary search methods ranging from general un-informed schemata like depth-first search and the like to general informed, heuristic, search algorithms like A^*, etc., to planning specific search procedures of various breeds. We will demonstrate how even adaptive, evolving, or multiple cooperating strategies can be realized.

As it has been noted above, when getting closer to an implemented planning system, dealing with the concepts of null plan and solution plan becomes more convenient rather than working with the formally more adequate problem specification and solution criteria. The subsequent chapters will successively phase from the formal framework to a concrete implementation of it, and therefore the signature of the detection function implementations will loose its second parameter, the problem specification. Secondly, the implementation chapter will change the modification generating functions' signatures such that they take sets of flaws (from the set of all flaws in current plan) instead of single ones. The data structure of modifications will be

augmented to this end with references to the addressed flaws so the strategies can recognize the associated pairs.

It goes without saying that these changes do not affect any of the proposed system properties and results.

One last note on the mutual implications of function granularity and the concept of putting flow control of a refinement planning algorithm into the trigger function (Def. 2.34): It becomes apparent that there is a clear correlation between the usefulness of the appropriateness definition and the *sensitivity* of detection functions and the *specificity* of plan modification generating functions. We can say that one detection function is more sensitive than another, if the cardinality of the respective affected set is smaller. A generator function that has an applicability set of smaller cardinality can in the same way be called to be more specific than another. The less sensitive a detection function and the less specific a modification generator is, the less precise we can decide upon appropriateness. There may be too many modification instances in the class that are located outside the intersection of the plans flawed and the plans in which the modifications are applicable (cf. intersection set $A \cap B$ in Fig. 2.10), or many of the obtained plans may be in the (now relative large) set of plans in which the flaw persists.

An implementation of the presented framework is well advised to consider the balance between the specificity of detection functions and the sensitivity of modification generators. Over-specific and over-sensitive functions lead to over-populated function sets in which no re-use of functionality occurs. On the other hand, under-specific and under-sensitive functions cause many false positive appropriateness results.

It is an interesting direction of future research to investigate the conditions, under which in every cycle of the generic algorithm *sets of plan modifications* can be executed on a partial plan. We believe that dependency analyses of flaws and corresponding modifications may allow us to identify compatible pairs of flaws and modifications for which a joint treatment does not cut solutions from the refinement space. Positive examples are groups of flaws for which only one modification exists (this topic will be addressed in a subsequent chapter) and that will remain the only options to solve the respective flaws. Such a technique would, of course, increase the influence of the modification selection function on the overall search control.

2.9 Summary and Conclusion

This discussion concludes the chapter on the introduction of the formal framework. We have presented the theoretical foundations for the representation and semantics of world states, which describe the relevant objects and their relationships, as well as the actions, which induce change by performing transitions on these world states. We have consequently introduced plans as the means for describing courses of actions in terms of state transitions and developed the concepts of their consistency and executability, accordingly. This plan semantics leads to the notion of developing plans by reducing their interpretation spectrum, a methodology that is commonly called refinement planning. In contrast to existing techniques, our approach to plan generation is well-founded and includes for the first time a complete and coherent integration of the notion of abstraction for both world states and actions, thus allowing us to operate on hybrid hierarchical domain models in a sound and meaningful way. It is worth mentioning that such an integration implies two important novel aspects with respect to hybrid planning: Firstly, it provides a semantic basis for defining decompositions of abstract tasks and thus for safely interleaving hierarchical refinements and non-hierarchical ones. Secondly, it allows for novel hybrid domain model features, for example, it enables a specification of complex state abstractions and abstract goal conditions. We will demonstrate implementations of all these new concepts in the subsequent chapter 3.

Furthermore, we have formally defined what a planning problem and its solution are, and, in accordance to the refinement planning paradigm, we have shown how the former can be transformed into the latter by so-called plan refinements. It is a novelty in this paradigm to employ an explicit representation of flaws, which indicate plan deficiencies, and plan modifications, which represent refinement options in terms of changes to the plan structure. From these representations we can deduce a so-called triggering function, which relates

classes of flaws to their resolving modification class candidates. The development of these techniques motivates a generic algorithm for refinement-based planning and scheduling that serves as the methodical basis for realizing an extremely modular system design. In this architecture, planning and scheduling functionality emerges from orchestrating the generic algorithm with the individual functional components for identifying plan deficiencies, for calculating plan-refinement options, and finally for making strategic search-related choices.

The following chapter 3 makes this framework operational by developing concrete incarnations of the different function sets, thereby providing an a assortment of planning and scheduling system components. We will see how a meaningful integration of these components can form system configurations for various purposes. All aspects of search-control are addressed separately in a dedicated strategy chapter (Chap. 4). In Chap. 5 we will finally demonstrate how the framework can be effectively implemented in a software artefact.

3 System Configurations – Instances Of The Refinement-Planning Framework

THE formal framework that has been presented in the previous chapter provides us with a well-found theory for building aggregations of functional modules that establish specific planning and scheduling capabilities. On that account, instead of designing, implementing, and eventually validating the specific software components of a particular planning or scheduling system, we propose a component-oriented approach that constitutes a generic modelling frame for the description and deployment of what we call *system configurations*. The accordingly implemented software artefacts (see Chap. 5) will then operationalize such configurations by means of orchestrating specific flaw detection and modification generating functions. If necessary, they are supplemented by inference functions. In order to demonstrate the full expressive power and adaptability of our framework, we do not restrict our presentation to the intended final product, an integrated hybrid planning and scheduling system. We rather show how a rich assortment of planning and scheduling capabilities – traditionally each of them a design and implementation effort in its own – can be developed in a natural and systematic way as system configurations. These framework incarnations can also be combined to constitute more advanced, high-level system configurations, which are usually referred to as "integrated approaches". The following discourse heads to the tentative climax of such a configuration evolution, which is, of course, the initially proposed fully integrated hybrid: The PANDA-System.

Before we are going into the details of some concrete framework realizations, we will address some aspects about the nature of configurations, their properties, and their interrelations. The following section therefore provides us with the definitions that are necessary to describe meaningful configurations and to work with configurations properly. This chapter will then continue with a road-map of the configurations that are addressed within this thesis and it will try to locate the results in the space of existing approaches and developments. The position finding will then be detailed by the sections on the concrete configurations and their capabilities. The chapter concludes with a brief discussion of the presented results.

Since strategy-related reasoning is independent from the refinement generating process, planning strategies are conceptually decoupled from the configurations. In order to adequately appreciate the complexity and flexibility of our strategic function implementations, their detailed presentation follows in a dedicated subsequent chapter.

3.1 System Configurations And Their Properties

Previous sections have made use of an informal notion of configurations: up to now, they are basically arbitrary collections of flaw detection and modification generating functions, accompanied by inference transformations and strategic search control. The previously presented refinement planning process takes these collections as an input and processes with them the planning problem input: checking for flaws, proposing resolving modifications, chosing a modification, and so forth. According to the proceeding chapter's results, any collection of these functions will in principle perform some sort of planning in the space of refinements – given that the functions meet certain soundness criteria. It is however obvious that such a weak notion of configurations should be reasonably extended to a more formal description of how functional fragments can or should be compiled in order to put only meaningful and, of course, useful aggregations into operation. Therefore, we first introduce what *exactly* is to be understood by the term "configuration" and from that we examine some properties that system configurations exhibit.

Definition 3.1 (System Configuration)**.** A *system configuration* for the generic refinement planning algorithm 2.2 is a quadruple $\mathfrak{C} = \langle \mathfrak{Det}, \mathfrak{Mod}, \mathfrak{Inf}, \mathfrak{Str} \rangle$ consisting of the following components:

- $\mathfrak{Det} = \{f_{x_1}^{det}, \ldots, f_{x_m}^{det}\}$ being a non-empty set of detection functions;
- $\mathfrak{Mod} = \{f_{y_1}^{mod}, \ldots, f_{y_n}^{mod}\}$ being a non-empty set of modification generating functions;
- $\mathfrak{Inf} = \{f_{z_1}^{inf}, \ldots, f_{z_k}^{inf}\}$ being a set of inference functions; and
- $\mathfrak{Str} = (f_p^{modSel}, f_q^{planSel}, f_r^{solSel})$ being a triple of strategy functions comprising the modification, plan, and solution selection functions p, q, and r.

•

It is indeed arguable whether or not to include strategy functions in system configuration definitions. If configurations are merely meant to describe the functional capabilities of the composed planning and scheduling system, then strategy functions are not necessarily part of them. This is because the sum of functionality in the detection and modification generating function sets alone determines the explorable plan refinement space and, consequently, what kind of planning or scheduling problems can be addressed.

On the other hand, and this is the perspective of this thesis, a configuration can be interpreted as the collection of all the entities that are required for running the generic refinement planning algorithm, and consequently the strategy functions belong to a (proper) formal system configuration. But since the previous argument seems better suited for a reasonable configuration classification, the following configuration descriptions are organized according to the planning and scheduling methodology they constitute, and they will therefore speak of *classes of configurations* that are identical *modulo* their respective strategy triples.

In order to ensure that a system configuration constitutes a proper composition of functional entities, and hence is finally resulting in a meaningful software artefact for generating plans, the following definitions offer a more formal concept for sound and complete system configurations. Intuitively, the properness of a system configuration builds upon the soundness and the completeness of the function sets it is comprised of, and the provided detection functions have to "match" the modification generating functions.

Definition 3.2 (Properness of System Configurations)**.** A system configuration $\mathfrak{C} = \langle \mathfrak{Det}, \mathfrak{Mod}, \mathfrak{Inf}, \mathfrak{Str} \rangle$ is called *proper* if and only if for any domain model D the following conditions hold:

- $\mathfrak{Det} = \{f_{x_1}^{det}, \ldots, f_{x_m}^{det}\}$ is a complete set of sound detection functions (see Definitions 2.28 and 2.29).
- $\mathfrak{Mod} = \{f_{y_1}^{mod}, \ldots, f_{y_n}^{mod}\}$ is a semi-complete set of sound modification generating functions (Def. 2.31 and 2.32).
- $\mathfrak{Inf} = \{f_{z_1}^{inf}, \ldots, f_{z_k}^{inf}\}$ is a set of sound inference functions (Def. 2.41).
- For every class of plan modifications $\mathbb{M}_y$ that is characterized by a modification generator of the configuration, say $f_y^{mod} \in \mathfrak{Mod}$, let $\mathbb{F}_{\alpha_y} = \alpha^{-1}(\mathbb{M}_y)$ be those flaws that are in the domain of the triggering function and hence *suitable* flaws for the modification class y (see Def. 2.34). Let analogously $\mathbb{F}_x$ denote the class of flaws that is characterized by a respective detection function $f_x^{det} \in \mathfrak{Det}$.

  The set of modification generating functions $\mathfrak{Mod}$ is then said to *correspond to* the set of detection functions $\mathfrak{Det}$ if and only if for all generated modification classes suitable flaw classes are provided in the configuration. Formally:

$$\forall i, 1 \leq i \leq n : \alpha^{-1}(\mathbb{M}_{y_i}) \cap \bigcup_{j=1}^{m} \mathbb{F}_{x_j} \neq \emptyset$$

•

In order to provide a meaningful configuration, we regard it to be at least an undesired if not improper design of the configuration provider to include modification generating functions for which no suitable detection function is deployed that actually triggers them. The inverse situation, that is, providing flaw detections for which no appropriate resolving modifications are specified, is unproblematic in general. It not only makes

sense to define cut-off criteria that mark undesirable plan developments, it is furthermore inevitable to encounter plans for which no solution refinement exists. For example, there exists no refinement for solving the deficiency of a plan having an inconsistent variable assignment. But later sections will also present practically relevant system configurations that are *incomplete* in the sense that they provide flaw detection functions that are *not covered* by matching modification generators *although* there exist in principle refinements that solve the reported deficiency. In any case we may rest assured: with the results from the previous chapter, that is, monotonicity, correctness, and completeness of flaws and plan modifications, any proper system configuration (in particular independent from the strategy triple) makes the generic refinement-based planning algorithm a *sound plan generation procedure*. Although the here discussed kind of completeness only has a minor practical impact, the following property is nevertheless worth being declared and assessed for individual configurations.

Definition 3.3 (Modification-Complete System Configurations)**.** A system configuration $\mathfrak{C} = \langle \mathfrak{Det}, \mathfrak{Mod}, \mathfrak{Inf}, \mathfrak{Str} \rangle$ with function sets $\mathfrak{Det} = \{f_{x_1}^{det}, \ldots, f_{x_n}^{det}\}$ and $\mathfrak{Mod} = \{f_{y_1}^{mod}, \ldots, f_{y_m}^{mod}\}$ is called *modification-complete* if and only if for any domain model and for every detection function $f_{x_i}^{det} \in \mathfrak{Det}$ with its associated flaw class $\mathbb{F}_{x_i}$ the following holds:

$$\forall 1 \leq i \leq n : \text{Either } \alpha(\mathbb{F}_{x_i}) = \emptyset \ \ (\mathbb{F}_{x_i} \text{ is critical) or } \ \alpha(\mathbb{F}_{x_i}) \cap \bigcup_{j=1}^{m} \mathbb{M}_{y_j} \neq \emptyset$$

for $\mathbb{M}_{y_j}$ being the modification class that is associated with the modification generating function $f_{y_j}^{mod}$ in $\mathfrak{Mod}$. •

Please remember the subtle differences between the involved concepts of completeness: A modification-complete system configuration provides modification generators for all of the deployed detections functions (at least except those dealing with critical flaws as per Def. 2.35). The generator set is however semi-complete, that means, the individual generators may not be able to produce every refinement modification for every flaw instance and therefore the plan generation procedure as a whole may not be complete (cf. discussion in Sec. 2.8.5).

Compiling a proper system configuration can be achieved in two ways: The first is to assemble those modification generating functions that describe the desired plan generation methodology and to collect the appropriate solution criteria fragments in terms of the detection functions. The second is to start out from an existing proper system configuration and to *extend* that configuration: This section will present a formal definition for realising extensions.

A straightforward approach for the definition of configuration extensions is to add more functions in the respective configuration components, which however implied that none of the functions could be simply substituted in extensions. Substitution is a very convenient way though to adapt (some of) the functions to slightly extended requirements, for example, in order to cover additional domain model features. The following definition is therefore founded on the semantics of system configuration components, thereby providing a more generalized notion of configuration extension.

Definition 3.4 (Extensions of System Configurations)**.** Given two system configurations $\mathfrak{C}_a = \langle \mathfrak{Det}_a, \mathfrak{Mod}_a, \mathfrak{Inf}_a, \mathfrak{Str}_a \rangle$ and $\mathfrak{C}_b = \langle \mathfrak{Det}_b, \mathfrak{Mod}_b, \mathfrak{Inf}_b, \mathfrak{Str}_b \rangle$, the second configuration $\mathfrak{C}_b$ is called an *extension* of the first configuration $\mathfrak{C}_a$ if and only if the following conditions hold:

- Let $\mathscr{P}_{\mathbb{F}_x}$ denote the affected set of flaw class $\mathbb{F}_x$ (Def. 2.27) and let f_x^{det} be its characterizing detection function, then
$$\bigcup_{f_x^{det} \in \mathfrak{Det}_a} \mathscr{P}_{\mathbb{F}_x} \subseteq \bigcup_{f_{x'}^{det} \in \mathfrak{Det}_b} \mathscr{P}_{\mathbb{F}_{x'}}$$
- Let $\mathscr{P}_{\mathbb{M}_y}$ and $\mathscr{P}'_{\mathbb{M}_y}$ denote the applicability and obtainability sets of plan modification class $\mathbb{M}_y$ (see Def. 2.21), and let furthermore f_y^{mod} be the respective characterizing modification generation function, then
$$\bigcup_{f_y^{mod} \in \mathfrak{Mod}_a} \mathscr{P}_{\mathbb{M}_y} \subseteq \bigcup_{f_{y'}^{mod} \in \mathfrak{Mod}_b} \mathscr{P}_{\mathbb{M}_{y'}} \quad \text{and} \quad \bigcup_{f_y^{mod} \in \mathfrak{Mod}_a} \mathscr{P}'_{\mathbb{M}_y} \subseteq \bigcup_{f_{y'}^{mod} \in \mathfrak{Mod}_b} \mathscr{P}'_{\mathbb{M}_{y'}}$$

- The inference functions in $\mathfrak{Inf}_a$ and $\mathfrak{Inf}_b$ are analogously compared with respect to their issued modifications' applicability and obtainability sets.

It is a synonymic expression to say that configuration $\mathfrak{C}_b$ *subsumes* configuration $\mathfrak{C}_a$. •

It is easy to see that an extension of a proper configuration in general is not automatically proper by itself and therefore has to be validated individually, in particular against the correspondence property. It has to be noted at this point that in practice it is comparably simple to identify the contributing detection functions for a given modification generator.

The rationale behind the above extension semantics is a notion of *conservativeness* with respect to the covered refinement spaces and respective solutions therein – both are induced by the respective configuration's function sets. Suppose a system configuration $\mathfrak{C}_b$ that subsumes a configuration $\mathfrak{C}_a$. Since the joint affected sets in the detection functions of the b-configuration are a super-set of those in a, the extended configuration incorporates stricter solution criteria than the previous one (more plans are marked to be faulty than before) and in particular any previously flawed plan *stays flawed*. In addition, the applicability sets of the modifications that are provided by the refinement generators and inference functions in b contain all those in a. That means, the extension preserves, for every plan, all refinements that were applicable in the previous configuration. Together with the cross-configuration flaw persistence, this entails that all previously developable plans in the refinement space *stay refineable*. Last but not least, all plans in the previous refinement space that are the product of a refinement operation are present in the subsuming configuration's refinement space as well, because the subsuming obtainability sets in b are super-sets of the initial a configuration: formerly constructable plans *stay constructable*.

If the extended system configuration was presented the same refinement space root as before, the induced refinement space consequently "subsumes" in a certain sense that of the previous configuration. Note that this subsumption does not necessarily mean a complete preservation: It may neither be re-obtained as an entire subspace nor encountered as scattered in fragments along refinement paths (for instance, interleaved with the newly introduced modifications). An example for a partial loss of the refinement space is the following: Let configuration $\mathfrak{C}_a$ provide a modification for a plan P to generate another plan P'. In an extended system configuration $\mathfrak{C}_b$, there are still refinements available for both plans because the applicability sets are included. But there may also be an inference function in $\mathfrak{Inf}_b$ that induces a transformation of plan P directly into P'' such that P' is never subject to any flaw detection and modification generation phase – and consequently none of the previously considered refinements of P' may ever be constructed. A second situation is the presence of extended detection modules. They may issue a flaw for plan P that none of the appropriate modification generators can address. In this case, refinements from configuration $\mathfrak{C}_a$ for generating P' may exist but are never applied when running the b-configuration because any strategy has to discard the plan before.

With the above considerations, the proposed extension-semantics for system-configurations has two major practical implications on planning systems that are realized within the presented framework: Firstly, system configurations can be extended *dynamically*. A slim and efficient configuration, for instance, is used for a couple of plan modification steps in order to explore the refinement space superficially and then the configuration is extended at run-time in order to complete plan generation. This thesis deals with static configurations only, but some of the issues will be discussed later in Sec. 3.5. The second effect is that extensions are not necessarily restricted to the structure of the functional fragmentation of the configurations they are built from. Thus we are able to *merge* configurations in a proper way, not only by adding new functional components, but also by fusioning or redesigning them, and thereby construct really integrated or hybrid planning functionalities.

Before we are going into the detailed configuration descriptions, let us have a look at an informal "taxonomy" for planning and scheduling capabilities, depicted in Fig. 3.1. The diagram shows some particularly interesting configuration individuals within the huge space of possible system configurations (with increasing capabilities from the top to the bottom) and how they can be assembled and recombined to constitute new and higher valued planning capabilities. Every system configuration is represented by a box that consists of four compartments: the kinds of plan components that are involved or introduced, the deployed flaw detection functions, the used modification generation functions, and the included inference

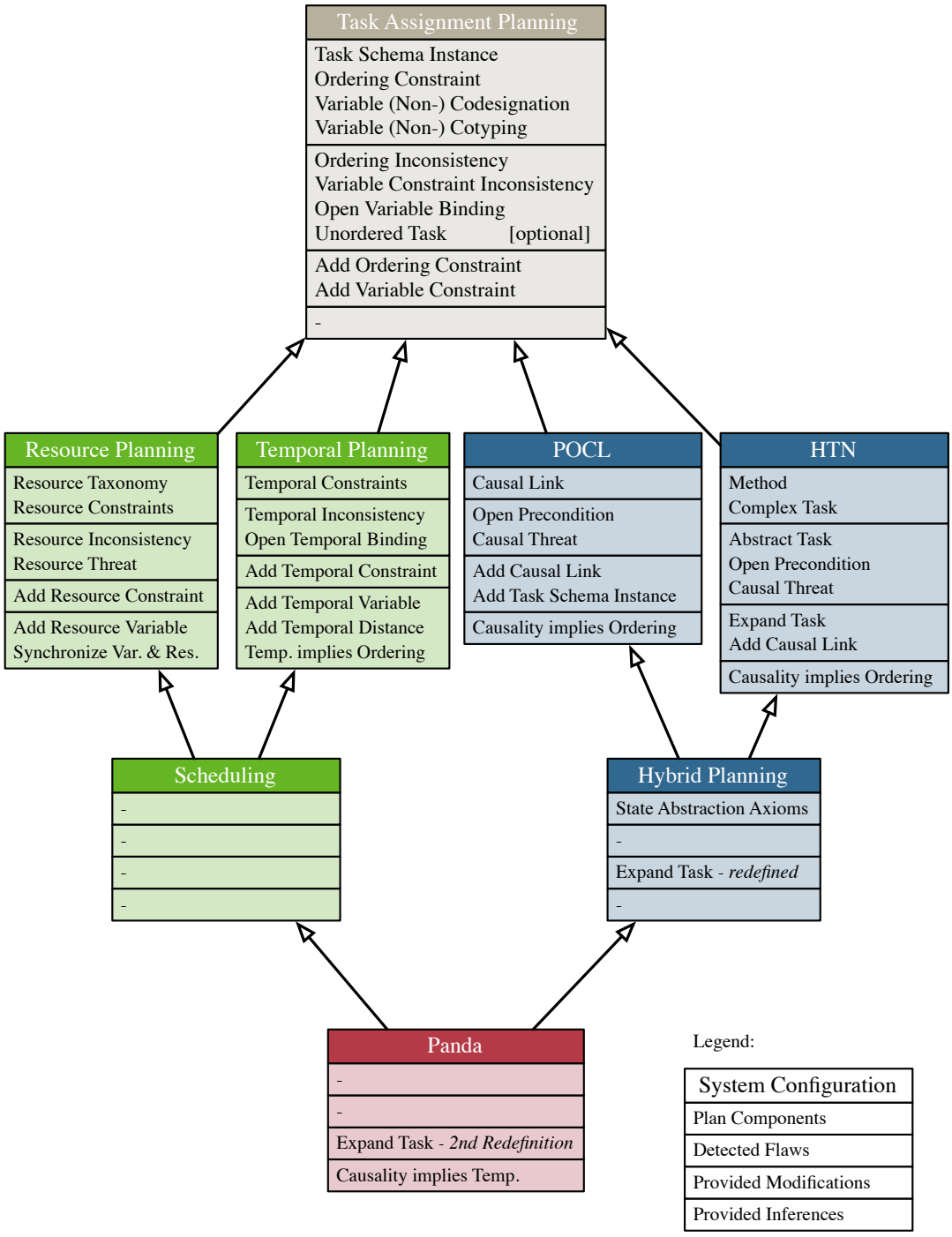


Figure 3.1: A taxonomy of the planning and scheduling system configurations presented in this chapter. It shows the involved plan components and participating configuration constituents. Details on the composition of extensions are given in the respective sections 3.2.1 to 3.4.

functions. The arrows denote the "extension" relationship between two configurations, that means, a configuration at the starting end of an arrow subsumes all functionalities of the respective predecessor it is pointing at.

The taxonomy's classification reflects a perspective based on the capabilities that are provided by the different system configurations, but there is also an alternative interpretation of this systematization: Let us assume that each system configuration is associated with the set of all the possible problem specifications over all possible planning domains for which a complete planning strategy triple is able to find all offered[1] solutions. Such problem sets are uniquely identified by the respective refinement space that is induced by the detection and modification generating functions in the system configuration (see also Def. 2.22). Let furthermore $\mathbb{PP}_a$ denote all problems that are decidable in the previous sense for a system configuration $\mathfrak{C}_a$ and let $\mathbb{PP}_b$ be the corresponding set for a configuration $\mathfrak{C}_b$ that is an extension of a. Given the above definition of extensions, it follows directly that the more advanced configurations are always able to solve all the problems of their ancestor configurations, while the opposite is not true. From this result we can infer that necessarily $\mathbb{PP}_a \subseteq \mathbb{PP}_b$. A further formal treatment of the relationship between configurations and their associated problem sets is beyond of the scope of this thesis, but it is an intuitive consequence that the subsumption property can be translated from the configuration view into that of associated problems and solutions: the problems in set a may therefore appear as so-called "sub-problems" in (all of) the instances of set b and accordingly the solutions for these sub-problems can be utilized in finding solutions or identifying non-solutions, respectively.

We will begin this survey at the taxonomy's root, a configuration that performs a primitive form of planning, namely *Task Assignment Planning*. As we will see, this kind of planning provides the common basis for reasoning about task instances, ordering relations and parameter assignments. Partial-Order Causal-Link Planning (POCL), Hierarchical Task Network Planning (HTN), *Resource Planning*, and *Temporal Planning* are specialisations that conceptually cover the majority of representatives of the approaches discussed in Chap. 1. These differentiations are merged into higher level configurations *Hybrid Planning* and *Scheduling*. They combine the strengths of task and state abstraction mechanisms, respectively integrate reasoning about temporal phenomena and resource consumption. We finally present the overall peak of our integration efforts, the PANDA system.

Since our formal approach guarantees the independence of any plan generation reasoning in terms of flaws and modifications from the actual search strategy, all of the mentioned sections only deal with the plan defect detectors and plan space generators for the respective planning methodology. All strategy issues are consequently addressed outside the configuration descriptions, and are presented separately in a dedicated strategy chapter 4, which also deals with the possible interactions between the choice of strategy and the configuration components that constitute the planning functionality. The introduction of concrete planning strategies will not only show translations of some of literature's most prominent search schema proposals, but it will primarily expound our own developments: Novel planning strategies that (a) have been stimulated by the possibilities that are given by our formal framework's explicit representation of plan deficiencies and solution options and that (b) opportunistically navigate through the refinement space and are hence called *flexible strategies*. A representative cross-section will be evaluated later in Chap. 6.

Fig. 3.2 tries to capture the flexibility of such an assembling of planning technology. The main configurations from the above classification taxonomy are depicted as white areas in a two dimensional space: one dimension represents the plan generation principles of building plans from first principles by a causal analysis and of refining abstract plan specifications into concrete courses of action. The second dimension addresses using a purely symbolic representation in contrast to working on (more or less continuous) numeric representations when dealing with resources.

The "classic" configurations are cornerstones of that plane, while the integrated approaches bridge the representational and methodological gaps between them. Resource Planning, for instance, extends the purely symbolic plan synthesis technique of POCL planning by reasoning about resource consumption and a richer time model. In doing so, it meets Scheduling, which in turn does not reason about the causal interactions of the involved activities but is able to assign time slots and resources to them. The integrated PANDA approach finally covers all areas.

[1] That means, the strategy discovers all solutions in the induced refinement space.

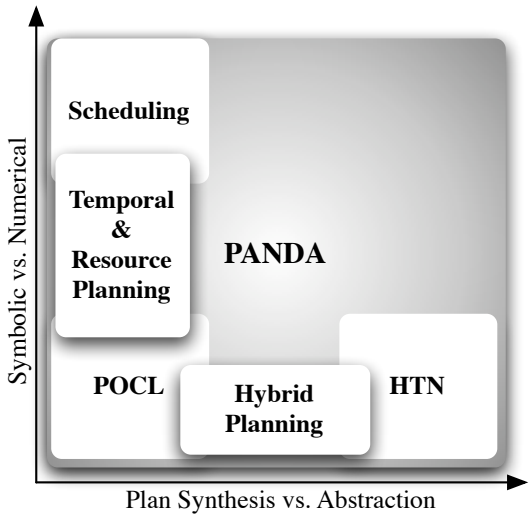


Figure 3.2: Flexibility of the PANDA modularization concept: the plane of possible system configurations that is spanned along the dimensions of "hierarchization" of domain concepts and the granularity of involved objects (from purely symbolic representations to nearly-continuous numerical quantities).

Fig. 3.2 might suggest that the chosen configuration individuals represent a complete coverage of the framework's options. It has however to be noted that the configuration space is in fact much more populated than it appears to be and allows for more discriminable values along its axes. We refer to the respective sections but the reader may imagine a symbolic plan synthesizing configuration to which only *a few* resource reasoning capabilities are added, say, reasoning about symbolic resources. The resulting individual would have to be placed *between* POCL and Resource Planning. In addition, we can think of more advanced extensions that add further dimensions to the configuration space. For example, Sections 7.2.1 and 7.2.2 will give a perspective on future work that covers the ability of reasoning about uncertainty or the involvement of the human users' problem solving competence (also known as interactive or *mixed initiative planning*). Against that background, it has to be emphasized once more that the formal framework allows for search strategies which are completely separated from the actual plan generation methodology. It is an essential achievement in this context, since there are numerous strategies developed for the basic planning techniques, that is to say, the corner-stones of the configuration plane, but definitely not for the majority of the integrating hybrid approaches.

The following sections will detail the representation of the concrete system configurations from the discussed taxonomy of Fig. 3.1 in our proposed formal framework. We will give a motivation and application examples for each of them in order to show their potential and limitations. Each presentation includes the constituting configuration components, that means, the flaw detection, modification generating, and inference functions, together with the corresponding triggering function definitions. Every system configuration therefore constitutes a self-contained, fully functional planning system when plugged into the generic algorithm (Alg. 2.2 on page 70), modulo the strategic function triple. The forthcoming presentation will pursue the following road-map for introducing the discussed system incarnations:

1. Task Assignment Planning (Sec. 3.2.1): As it has been noted before, this configuration provides basic representations and capabilities for dealing with actions and their parameters.
2. Partial Order Causal Link Planning (POCL, Sec. 3.2.2): The first extension of the task assignment planning configuration adds the treatment of causal dependencies in a plan.
3. Hierarchical Task Network Planning (HTN, Sec. 3.2.3): Planning by finding consistent implementations for abstract actions. This extension adds procedural knowledge about actions to the task assignment configuration.
4. Hybrid Planning (Sec. 3.3.1): We extend POCL and HTN planning by merging the two configurations into the first integrated approach. Hybrid planning considers the causal interactions and dependencies

of abstract and concrete actions likewise, and it is in addition capable to simultaneously implement abstract actions by predefined partial plans. This configuration will demonstrate how a merge of configurations can be realized for highly interdependent function sets.

5. Resource Planning (Sec. 3.3.3): The reasoning about plans that deal with resource consumption is based on task assignment planning. We introduce a non-intrusive representation for symbolic resources that are allocated for certain tasks as well as for numerical resources that can be consumed and produced in defined quantities by plan activities.
6. Temporal Planning (Sec. 3.3.2): Based on task assignment planning, the temporal planning configuration introduces a much richer temporal representation for the actions. Like for resource planning, additional "meta-constraints" transparently define time windows and execution durations for tasks.
7. Scheduling (Sec. 3.3.4): The second hybrid approach, a merge between the temporal reasoning and the resource-aware planning methods. This configuration illustrates in particular how to integrate entirely different planning capabilities in a transparent way.
8. Integrated Hybrid Planning and Scheduling (PANDA, Sec. 3.4): Scheduling is combined with the state and task abstraction mechanisms of hybrid planning. This configuration is able to deal with causal dependencies and resource interactions simultaneously, even across multiple levels of abstraction.

The chapter concludes with a brief discussion of the presented material.

The following sections will make use of various application domains in order to display the specific techniques the respective system configuration brings to the table. Please note that all scenario's domain model fragments are only intended for demonstration purposes and do not constitute a completely worked out, integrated model across the following sections. The accentuated alternative modelling principles may rather imply inconsistencies between the the respective fragments.

3.2 Basic System Configurations

3.2.1 Task Assignment Planning – $\mathfrak{C}_{\mathsf{TAP}}$

Task Assignment (TAP) Planning will serve as a basic system configuration $\mathfrak{C}_{\text{TAP}}$ and the root anchor of the configuration taxonomy (Fig. 3.1). It is a very rudimentary form of planning that reflects what is typically done on a piece of paper during during manual project management, personal organization, etc. The specification of a TAP plan resembles specifying a flow-chart: We basically write down what tasks have to be done and arrange them chronologically. For each task we note who the responsible person is, which part of the equipment is used, and the like. In cases where the additional information is not definite or known, we may use place-holders and start to co-designate them, for instance, two tasks have to be carried out by the same person. Fig. 3.3 shows such a TAP plan, not on a piece of paper, but modelled in a commercially available project planning tool. Although such tools do neither support the definition of something equivalent to variable constraints nor to task parameters, the basic notion of a task is comparable. While TAP planning understands tasks (or equivalently: jobs, activities) as actions that are to be performed in the real world and as processes the execution of which takes time, the actual evolution of the execution environment is not explicitly captured by TAP's representation. In the pen-and-paper approach, the modeler has to keep in mind all task dependencies and to consider them when formulating parameter constraints and job orderings. Causality reasoning is done on a "best-practice" basis or supported by suitable, tried and tested, predefined templates (sub-plans). Finally, the overall objective is un-represented as well and only known to the human user who sets up the plan. In the TAP perspective, any given TAP plan is the objective by itself and is regarded to be executable if the choice of parameter assignments is consistent with the specified constraints. It is worth noting that executability exclusively relies on the modeler's competence to consider all task interactions during the modelling process.

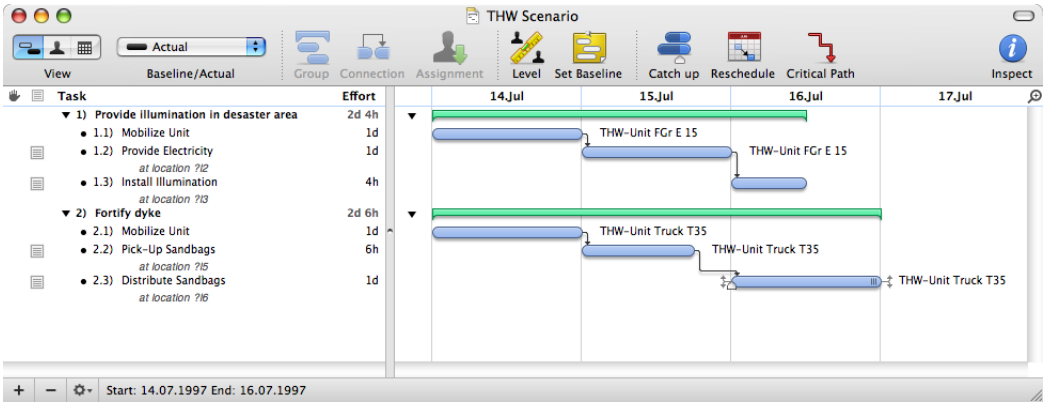
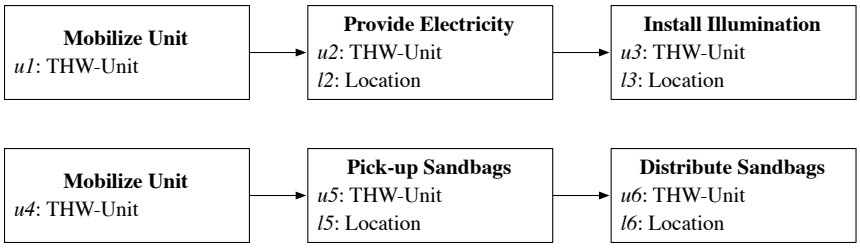


Figure 3.3: An example scenario for Task Assignment Planning, represented in commercially available project planning software. The screenshot is taken from OmniPlan, ©2006 The Omni Group.



$$VC = \{\ u_1 \doteq u_2, u_2 \doteq u_3, u_3 \dot{\in} \text{Taskforce Electricity (FGr E)}, u_1 \dot{\neq} u_4, u_4 \doteq u_5, \\ u_5 \dot{\in} \text{Truck}, u_5 \doteq u_6, l_2 \doteq \text{Dyke Segment 5}, l_2 \doteq l_3, l_3 \doteq l_5, l_5 \doteq l_6\}$$

Figure 3.4: An initial plan of a task assignment planning problem in the disaster relief mission domain.

Domain Model Specifics

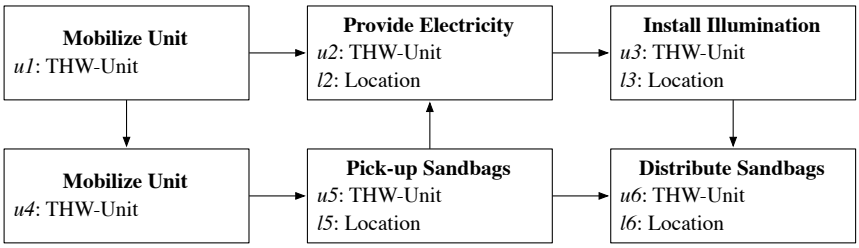
In terms of the refinement planning framework, a TAP planning system configuration works on a reduced domain model representation and is responsible for maintaining consistency on the task orderings and the variable constraints. The formal definition of a TAP domain model is given by the triple $D_{\text{TAP}} = \langle \mathscr{M}, \emptyset, \mathtt{T} \rangle$, that means, no state-abstraction axioms are provided. All task schemata in $\mathtt{T}$ are primitive and do not carry preconditions or effects, since causality is not involved when deciding about executability. TAP operator schemata are therefore structures like $o(\bar{v}) = \langle \top, \varepsilon \rangle$, so that their instances will be executable in any state (see primitive task schema Def. 2.4). A TAP plan is consequently a reduced representation of a partial plan (Def. 2.9) and basically consists of a partially ordered set of parametrized tasks: $P = \langle TE, \prec, VC \rangle$. It makes no sense to provide causal links, because no causality is to be documented.

Problems and Solutions

A TAP problem specification is given by the tuple $\pi_{\text{TAP}} = \langle D_{\text{TAP}}, s_\varepsilon, \top, P_{init} \rangle$, that means it is an ordinary planning problem with the empty initial state[2] and a trivial goal state specification. The initial plan P_{init} is thereby restricted to a TAP plan.

Fig. 3.4 shows an example task assignment planning problem in the context of the disaster relief mission scenario. The initial plan contains two parallel[3] threads of activities: one in which a unit of the relief organi-

[2] In state s_ε every flexible relation is interpreted as the empty set and every flexible term as undefined (Def. 2.18).



$$VC = \{\ u_1 \doteq u_2, u_2 \doteq u_3, u_3 \doteq \text{Taskforce Electricity (FGr E)}, u_1 \dot{\neq} u_4, u_4 \doteq u_5,$$
$$u_5 \doteq \text{Truck}, u_5 \doteq u_6, l_2 \doteq \text{Dyke Segment } 5, l_2 \doteq l_3, l_3 \doteq l_5, l_5 \doteq l_6,$$
$$u_1 \doteq \text{FGr E Neuwied}, u_4 \doteq \text{9t Truck, FGr BrB, 2.TZ Dresden}\}$$

Figure 3.5: A solution to the task assignment planning problem of Fig. 3.4.

sation is mobilized and finally sets up an electric lighting in order to support a nightly dyke fortification. The second thread deals with a second unit that transports sandbags to an endangered location and distributes them. The given constraints on the tasks parameters specify that the two threads involve two different units by the non-codesignation $u1 \dot{\neq} u4$. In order to ensure that each thread consistently uses the same unit during the mission, the co-designation constraints induce that $u1 = u2 = u3$ and $u4 = u5 = u6$. Furthermore, the TAP plan use co-typings for restricting the assigned units to the sorts that seem to be appropriate for the given task: The activity "provide electricity" has to be carried out by a special task force of the relief organization that is equipped with high-power generator technology, the so-called *Fachgruppe Elektroversorgung* (power supply task force), and that is denoted by by the co-typing constraint $u3 \dot{\in} \texttt{FGrE}$.

Given the TAP problem definition, a solution can be characterized by a TAP plan that has consistent ordering constraints and a set of variable constraints that are globally consistent. In addition, we require that every variable is assigned to a rigid constant value (see Sec. 2.8.1 for a discussion of parametrized plans). The remaining solution criteria of the framework's general Definition 2.17 are trivially true, due to the extremely reduced executability concept.

A solution to the example task assignment problem is depicted in Fig. 3.5. The "open" variable assignments from the problem specification, namely the identity of the involved relief organisation units, are solved by additional variable co-designations. In this example, a heavy truck of a group in the city of Dresden is used for the transport, and illumination is provided by a special craft of the Neuwied group. Supplementary ordering constraints between the activities linearize the plan steps such that the result represents a sequence of actions.

Detection Functions – $\mathfrak{Det}_{\mathbf{TAP}}$

The set of flaw detection functions for task assignment planning covers three flaw classes: ordering relation inconsistencies, variable constraint inconsistencies, and unbound variables. The three associated detection functions are defined as follows:

Definition 3.5 (Ordering Inconsistency)**.** For a given task assignment plan $P = \langle TE, \prec, VC \rangle$ and TAP planning problem-specification π, the function $f^{det}_{\texttt{OrdIncons}}$ for detecting flaws that represent an inconsistency of P's ordering constraint set is defined as: $\{te_1, \ldots, te_n\} \in f^{det}_{\texttt{OrdIncons}}(P, \pi)$ if and only if for $1 \leq i \leq n$: $te_i \in TE$ and $te_i \prec^* te_i$. That means, the flawed elements are all plan steps, that are on a cyclic path in the transitive closure $\prec^*$ of P's partial order. •

Definition 3.6 (Variable Constraint Inconsistency)**.** Given a task assignment plan $P = \langle TE, \prec, VC \rangle$ and problem specification π, the function $f^{det}_{\texttt{VarIncons}}$ for detecting flaws that represent an inconsistency of P's variable constraint set is defined as: $\{v_1, \ldots, v_n\} \in f^{det}_{\texttt{VarIncons}}(P, \pi)$ with $v_i \in \mathscr{V}$ (for $1 \leq i \leq n$) if and only if

[3] See discussion on parallelism in Sec. 2.8.3.

$VC \models v_i \neq v_i$ holds. That means, the flawed variables are those for which the inferential closure of the inconsistent variable constraint set infers that they are unequal to themselves. •

The detection functions are provably sound, because if the ground linearizations exist, this means in particular that the constraint sets are consistent and no flaw is issued. Note that both flaw classes $\mathbb{F}_{\texttt{OrdIncons}}$ and $\mathbb{F}_{\texttt{VarIncons}}$ are critical flaws in compliance with Def. 2.35.

As it has been stated above, it is one problem during task assignment to find appropriate values or assignments to all unbound task parameters. While a variable is not yet definitely assigned a constant value, that means, while more than one single variable assignment is consistent with the variable constraints, the following detection function issues its flaws.

Definition 3.7 (Open Variable Binding)**.** For a given task assignment plan $P = \langle TE, \prec, VC \rangle$ and TAP-planning problem π, the flaw detection function $f^{det}_{\texttt{OpenVarBind}}$ signals the occurrence of variables in P that have not yet been assigned to a constant by VC. It is defined as follows: $\{v\} \in f^{det}_{\texttt{OpenVarBind}}(P, \pi)$ with $v \in V$ being a variable occurring in P, if and only if there exist constants $c_1, c_2 \in C$ with $c_1 \neq c_2$ and $VC \models v = c_1$ as well as $VC \models v = c_2$. •

Note that this definition is not restricted to task parameters but also to variables solely occurring in VC. Since this detection function directly incorporates a solution criterion of task assignment planning, it is trivially sound.

If the application relies on linear plans only, the following detection function that informs about partially un-ordered plan steps is optionally deployed.

Definition 3.8 (Unordered Task)**.** For a given task assignment plan $P = \langle TE, \prec, VC \rangle$ and TAP problem specification π, the flaw detection function $f^{det}_{\texttt{UnordTask}}$ returns all pairs of plan steps in TE that are not temporally related by the transitive closure of $\prec$: $\{te_i, te_j\} \in f^{det}_{\texttt{UnordTask}}(P, \pi)$ if and only if $te_i, te_j \in TE$ and neither $te_i \prec^* te_j$ nor $te_j \prec^* te_i$. •

Depending on the chosen TAP-variant, this detection function is sound.

Now that we have defined the plan deficiencies it is time to introduce the plan modification generators.

Modification Generating Functions – $\mathfrak{Mod}_{\texttt{TAP}}$

The available refinement generators for the task assignment planning configuration manipulate the ordering and variable constraints in a straightforward way.

Definition 3.9 (Add Ordering Constraint)**.** For a given task assignment plan $P = \langle TE, \prec, VC \rangle$, domain model D, and flaw $\texttt{f}$, the modification generating function $f^{mod}_{\texttt{AddOrdConstr}}$ proposes to include an appropriate ordering constraint. $\langle \{te_i \prec te_j\}, \emptyset \rangle \in f^{mod}_{\texttt{AddOrdConstr}}(P, \texttt{f}, D)$ for $\{te_i, te_j\} \subseteq TE \cap comp(\texttt{f})$ and $te_i \prec te_j, te_j \prec te_i \notin \prec$. •

In order to address the flaw argument properly, the above definition introduces a function *comp*, which returns all components and sub-components of a set of plan components (see Sec. 2.6.2). Including it in the detection function's definition enables it to extract indirectly flawed task expressions, for instance, that are referenced by a passed ordering constraint.

Definition 3.10 (Add Variable Constraint)**.** For a given task assignment plan $P = \langle TE, \prec, VC \rangle$, domain model D, and flaw $\texttt{f}$, the modification generating function $f^{mod}_{\texttt{AddVarConstr}}$ suggests to include appropriate variable constraints. Let τ be a term over D's language $\mathscr{L}$ and Z a sort symbol in the respective set of sort symbols $\mathscr{Z}$:

1. $\langle \{v \doteq \tau\}, \emptyset \rangle \in f^{mod}_{\texttt{AddVarConstr}}(P, \texttt{f}, D)$ for $v \in \mathscr{V} \cap comp(\texttt{f})$ and $v \doteq \tau, v \dot{\neq} \tau \notin VC$
2. $\langle \{v \dot{\neq} \tau\}, \emptyset \rangle \in f^{mod}_{\texttt{AddVarConstr}}(P, \texttt{f}, D)$ for $v \in \mathscr{V} \cap comp(\texttt{f})$ and $v \dot{\neq} \tau, v \doteq \tau \notin VC$

3. $\langle\{v \dot{\in} Z\}, \emptyset\rangle \in f^{mod}_{\texttt{AddVarConstr}}(P, \texttt{f}, D)$ for $v \in \mathscr{V} \cap comp(\texttt{f})$ and $v \dot{\in} Z, v \dot{\notin} Z \notin VC$
4. $\langle\{v \dot{\notin} Z\}, \emptyset\rangle \in f^{mod}_{\texttt{AddVarConstr}}(P, \texttt{f}, D)$ for $v \in \mathscr{V} \cap comp(\texttt{f})$ and $v \dot{\notin} Z, v \dot{\in} Z \notin VC$

•

The produced plan modifications drop some trivial inconsistent proposals. A deeper analysis of the variable constraints is of course possible. For the sake of simplicity, we however decided to keep every function design as simple as possible and to put a special emphasis on distributing the competences properly. That means, we tolerate that $f^{mod}_{\texttt{AddVarConstr}}$ may issue some useless refinements because the "specialist function" $f^{det}_{\texttt{VarIncons}}$ will be able to detect any unwanted consequence immediately. Pursuing this principle allows us to concentrate on the specific functionalities without replicating reasoning processes throughout the system configuration.

Both modification generating functions are sound: Since their produced plan modifications exclusively *add constraints* to the respective sets, both induce a proper plan refinement (although the reduced domain model does not allow for a monotonic restriction of the intended system behavior) and both have a positive balance of elementary additions. They also meet the final soundness requirement as both provide only constraints that include the flawed elements due to the use of the *comp* function.

Triggering Function $\alpha_{\texttt{TAP}}$

The triggering function for task assignment planning is relatively simple, since the changes that are induced by the respective modification generating functions directly address the flawed elements of the detection functions. We therefore define the TAP triggering function as follows:

$$\alpha_{\text{TAP}}(\mathbb{F}_x) = \begin{cases} \mathbb{M}_{\texttt{AddVarConstr}} & \text{for } x = \texttt{OpenVarBind} \\ \mathbb{M}_{\texttt{AddOrdConstr}} & \text{for } x = \texttt{UnordTask} \\ \emptyset & \text{otherwise} \end{cases}$$

Inference Functions – $\mathfrak{Inf}_{\texttt{TAP}}$

The task assignment planning configuration does not need deductive support. However, it is convenient for an implementation to have an inference function at disposal that facilitate variable constraint inferences by making the transitive closure of *VC* explicit. This reduces implementation work and computational effort considerably. Since it is optional and does not contribute to the solution generation process in particular, we do not list it in the current configurations. It is nonetheless a sound inference function, because it returns proper plan modifications with a positive balance of elementary additions.

Summary of the TAP Configuration

With the above function set definitions we get for the TAP configuration

$$\begin{aligned} \mathfrak{C}_{\text{TAP}} = \langle &\{f^{det}_{\texttt{OrdIncons}}, f^{det}_{\texttt{VarIncons}}, f^{det}_{\texttt{OpenVarBind}}, f^{det}_{\texttt{UnordTask}}\}, \\ &\{f^{mod}_{\texttt{AddOrdConstr}}, f^{mod}_{\texttt{AddVarConstr}}\}, \\ &\emptyset, \mathfrak{Str}\rangle \end{aligned}$$

Theorem 3.1 (Properness of $\mathfrak{C}_{\text{TAP}}$)**.** *$\mathfrak{C}_{\text{TAP}}$ is a proper system configuration in the sense of Def. 3.2.*

Proof. It is easy to see that $\mathfrak{Det}_{\text{TAP}}$ is a complete set of sound detection functions. If a plan is no solution, according to the reduced representation of TAP, this means that no primitive ground linearization exists. This in turn implies that either the variable or the ordering constraint set (or both) is inconsistent, which is announced by the respective detection functions, and the function set is therefore complete. Since in addition, it has been shown above that the individual detection functions are sound, the assumption holds.

Note that TAP employs one additional solution criterion, namely definite variable assignments, and one optional (sequential plans).
$\mathfrak{Mod}_{\text{TAP}}$ is a semi-complete set of sound modification generating functions, firstly because no modification generator passes over a flaw. Note that if a flaw is not answered (for example, not adding an existing ordering constraint) this happens only because no properly built refinement can be constructed (existing components must not be added). Secondly, all of them are proven to be sound.
$\mathfrak{Inf}_{\text{TAP}}$ is trivially a set of sound inference functions, because it is an empty set.
$\mathfrak{Mod}_{\text{TAP}}$ corresponds to $\mathfrak{Det}_{\text{TAP}}$, because every generated modification class is provided with a suitable flaw class. □

Theorem 3.2 ($\mathfrak{C}_{\text{TAP}}$ is Modification-Complete)**.** $\mathfrak{C}_{\text{TAP}}$ *is modification-complete according to Def. 3.3.*

Proof. For the non-critical flaw classes $\mathbb{F}_{\texttt{OpenVarBind}}$ and $\mathbb{F}_{\texttt{UnordTask}}$, the configuration provides the modification classes $\mathbb{M}_{\texttt{AddVarConstr}}$ and $\mathbb{M}_{\texttt{AddOrdConstr}}$, respectively. □

3.2.2 Partial Order Causal Link Planning – $\mathfrak{C}_{\text{POCLP}}$

With Partial Order Causal Link Planning, accomplished by the system configuration $\mathfrak{C}_{\text{POCLP}}$, we present the first extension to the simple Task Assignment Planning configuration that addresses "real" planning problems. As described in the introductory section 1.1.2, reasoning about preconditions and effects of actions is central to POCL planning. Actions are put in a plan for a specific reason, namely to ensure the establishment of a state that satisfies a goal specification. The actions' preconditions themselves are regarded as new goals that may require further actions, and so on. Apart from constructing a causal chain, it has to be guaranteed that all established causal connections are safe from interferences with (side-) effects of other actions. To this end, the causal structure of a plan is made explicit by so-called *causal links* that are used as book-keeping entities for documenting the commitment to a specific causal interaction. In adopting this concept, verifying the executability of a plan becomes reasoning about completeness and threat-situations of the causal linking.

Fig. 3.6 displays a POCLP-plan as it is shown in an interactive planning tool for our system. The scenario is taken from our adaptation of the IPC satellite domain (see Sec. 5.2.1): A satellite instrument is first switched on, calibrated on a specific calibration target, the satellite then turns into the scientific target direction, and finally the instrument takes an image of the phenomenon of interest. The icons represent the plan steps and the black arrows the ordering relation on them. The red arrows denote causal links, but since the annotated conditions obstruct readability these are only visible when the user marks the link with the mouse. In the example, the switch-on operator provides energy to the scientific instrument, which enables taking a thermogram, and so on. The variable constraint set of the plan is textually represented at the bottom of the screen; the currently displayed portion contains exclusively co-designations. During plan generation, the graph structure is animated in order to be able to replay the complete refinement process and to comprehend the development of the plan.

Domain Model Specifics

The POCLP representation of tasks corresponds to that of operator schemata (Def. 2.4). As it is common in the POCL-community to use PDDL syntax and for the sake of a more uniform presentation, our pictorial examples sometimes deviate slightly from the framework's notation and employ the representation of effects as a conjunction of positive and negative literals[4]. It thereby corresponds to actions with the unconditional effects in PDDL [187] and its potential successor OPT [189]. We do not make assumptions about the usage of terms, rigid or flexible, however note that the commonly realized POCL-systems deal with literals in which only variables or rigid constant terms occur. The translation of this simple structure into our formalism is trivial: an action effect $\varphi_1 \wedge \ldots \wedge \varphi_n$ corresponds to the elementary operation sequence $e_1 \ldots e_n$, where e_i is

[4] Although this representation is technically equivalent to the even more common add and delete lists as introduced by STRIPS [92, 169], we regard this set-based style of modelling unappropriate for our framework.

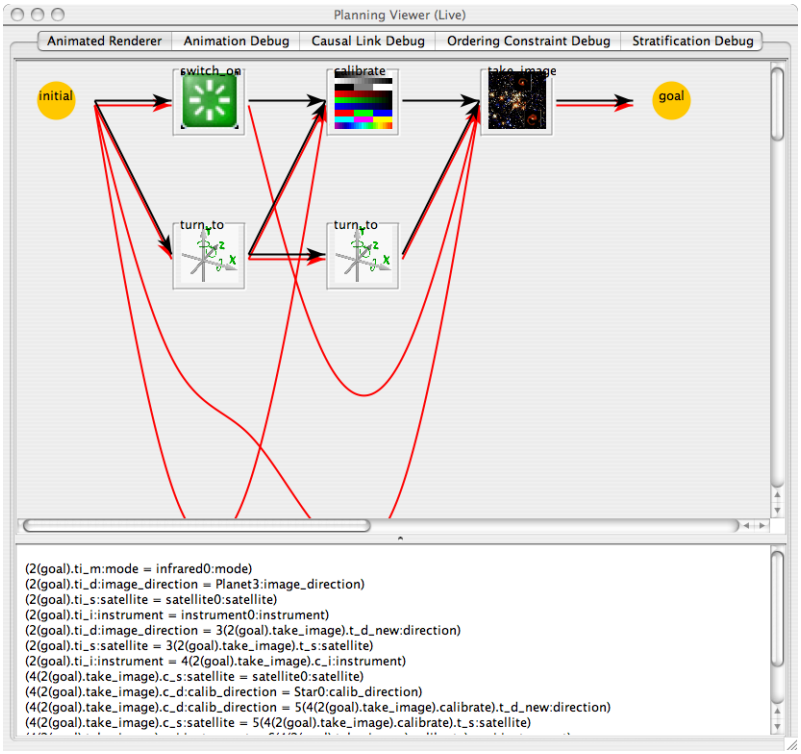


Figure 3.6: An example plan used in $\mathfrak{C}_{\text{POCLP}}$ (screenshot taken from an interactive prototype of our system). The upper panel displays temporal (black arrows) and causal dependencies (red arrows) between plan steps (pictograms). The part below shows a textual representation of variable constraints.

an operation over the respective relation symbol of φ_i, $1 \leq i \leq n$. If $\varphi_i = R(\overline{\tau})$ is a positive literal, then e_i is the elementary operation $+R(\overline{\tau})$ and $-R(\overline{\tau})$ otherwise.

The action depicted in Fig. 3.7 shows the operator for turning the operation platform in the satellite domain from a (symbolic) direction into another one as it is presented in our domain editing software. The precondition is the positive literal for describing the satellite's orientation before the operation, the effects section is the conjunction that specifies a state update such that the previous orientation is changed into the new orientation. Note the use of order-sorted variables and appropriately declared relation symbols (not explicitly visible in the figure, however implemented as background consistency checks in the editor, see also Sec. 7.2.3).

Since $\mathfrak{C}_{\text{POCLP}}$ only deals with primitive tasks, it does not use state abstraction axioms and a domain model for partial-order causal-link planning is therefore given by $D_{\text{POCLP}} = \langle \mathscr{M}, \emptyset, \mathtt{T} \rangle$. The above described operators are specified in the set of task schemata $\mathtt{T}$ and the logical model $\mathscr{M}$ is given as defined on page 32. Consistency of POCLP domain models can be directly reduced to consistency of the general domain model as given in Definition 2.8.

A *plan* is exactly the partial-plan structure of Def. 2.9, that means $P = \langle TE, \prec, VC, CL \rangle$. Please note one subtlety regarding the interpretation of causal links: while it is common to the POCL literature to define causal links as annotated ordering constraints, we regard them as structures that *imply* an ordering constraint if the linked task expressions are primitive ones. We believe that this view is also the intention of McAllester and Rosenblitt, who define corresponding ordering constraints as a requirement for causal links and not as being subsumed by them [177].

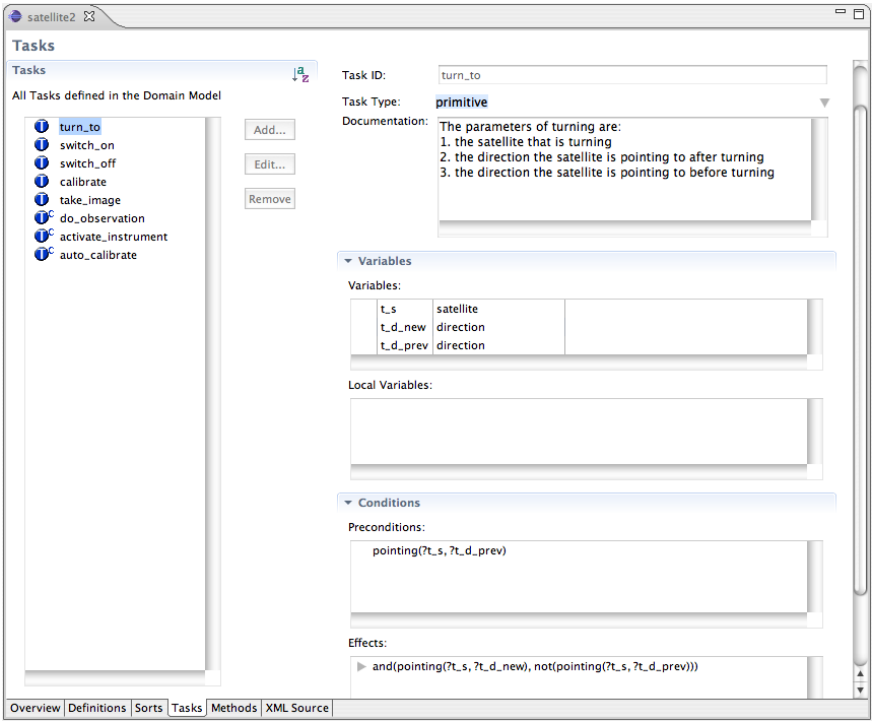


Figure 3.7: A POCL action specification (screenshot taken from our prototypical domain model editor).

Problems and Solutions

A problem specification for POCLP is defined via an initial state and a goal state specification: $\pi_{\text{POCLP}} = \langle D_{\text{POCLP}}, s_{init}, \mathfrak{s}_{goal}, \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle \rangle$. As it is common in POCL planning, we will transform this problem description into the *null plan* representation (Def. 2.18), which encodes the initial and goal states as artificial actions in the plan. This enables us to apply causal links from the initial state and to the goal state.

We consequently search for a *solution plan* (Def. 2.19), which reduces the general solution criteria to a single executability principle. As it has been pointed out in the introductory sections, partial order planning uses causal links for an efficient tractability analysis[5] and regards causal link threats as refutation arguments [54]. That means, the solution criteria for a plan are translated into the causality-aware detection functions of the following section.

Detection Functions – $\mathfrak{Det}_{\text{POCLP}}$

Regarding the basic consistency criteria, this configuration relies on the previous results for task assignment planning. It therefore employs all detection functions of $\mathfrak{Det}_{\text{TAP}}$. In addition, the flaw classes of the following two detection function definitions are supported.

The idea of causal links is to document which effects of which *producer* task are intended to establish the necessary features of a state in order to make a *consumer* task applicable. If there is no recognizable commitment that completely satisfies the consumer's requirements, this situation is called an "open precondition" of the consumer.

[5] Kambhampati discusses in [147] causal links in particular and tractability analysis in general, some of the relevant issues are also covered by formal efficiency analyses conducted in [16]. The implicitly underlying modal truth criterion as it has been formulated by Chapman [54] is surveyed with respect to its usability and practicability in [150].

Definition 3.11 (Open Precondition)**.** Given a partial plan $P = \langle TE, \prec, VC, CL \rangle$ and planning problem $\pi = \langle D, s_{init}, \mathfrak{s}_{goal}, P_{init} \rangle$, the flaw detection function $f^{det}_{\texttt{OpenPrec}}$ indicates that the causal support for a task expression $te \in TE$, documented by the respective causal links in CL, is not sufficient in order to guarantee that te will be applicable at execution time. More precisely: Let $\{te_1 \xrightarrow{\varphi_1} te, \ldots, te_n \xrightarrow{\varphi_n} te\}$ be the subset of CL that contains all n causal links that support te.

$\{te, \varphi\} \in f^{det}_{\texttt{OpenPrec}}(P, \pi)$ if and only if the following conditions hold:

1. For every state s and for every valuation β that is compatible with VC

$$s \models_{\mathscr{M},\beta} (\text{prec}(te) \Rightarrow \varphi)$$

2. There exists a state s and a VC-compatible valuation β such that

$$s \models_{\mathscr{M},\beta} \bigwedge_{1 \le i \le n} \varphi_i \quad \text{and} \quad s \not\models_{\mathscr{M},\beta} \text{prec}(te) \quad \text{and} \quad s \not\models_{\mathscr{M},\beta} \varphi$$

We may assume that the detection function's answer is the most specific one, that means, that there is no φ' such that $\{te, \varphi'\} \in f^{det}_{\texttt{OpenPrec}}(P, \pi)$ and $\varphi \Rightarrow \varphi'$. •

Intuitively, the first flaw condition states that φ is "part" of the task expression's precondition and the second condition specifies what it means for a number of causal links not to "cover a precondition completely" and to leave out the φ part. Soundness can easily be verified for $f^{det}_{\texttt{OpenPrec}}$: Let P be a solution to π and furthermore $\{te, \varphi\} \in f^{det}_{\texttt{OpenPrec}}(P, \pi)$. Since the flaw has been issued, both conditions have to be valid: φ holds in every state in which the task expression's precondition holds and it is possible that te is going to be executed in a state and under a valuation such that its promised support holds, though not its precondition. This means that there exist ground linearizations of P that are not executable (because of te), which is a contradiction to POCLP's solution criteria and the assumption is therefore refuted.

The other new detection function examines whether or not the assured conditions can be confirmed to actually *persist until* they are consumed by the linked task. If a state change is confirmed that undoes the effects of a producer task such that the consumer's precondition cannot be satisfied at execution time, the plan fails to provide a solution to the given problem. If however the state change is *potentially* occurring and *potentially* undoing the effect, respectively, POCLP calls this situation a *causal threat* that might be faced up to successfully.

Definition 3.12 (Causal Threat)**.** Given a partial plan $P = \langle TE, \prec, VC, CL \rangle$ and planning problem $\pi = \langle D, s_{init}, \mathfrak{s}_{goal}, P_{init} \rangle$, the detection function for flaws $\mathbb{F}_{\texttt{Threat}}$ points to those constellations of causal links and tasks in which a task's effects are able to falsify the annotated condition.

$\{te_i \xrightarrow{\varphi} te_j, te_k\} \in f^{det}_{\texttt{Threat}}(P, \pi)$ if and only if for task expressions te_i, te_j, and $te_k \in TE$ with $te_i \xrightarrow{\varphi} te_j \in CL$:

1. Neither $te_k \prec te_i$ nor $te_j \prec te_k$ are in the transitive closure of the ordering constraint set of P.
2. There exists a VC-compatible valuation β such that for all states s and s' the following holds:

$$\text{if} \quad s \models_{\mathscr{M},\beta} \varphi \quad \text{and} \quad \langle s, s' \rangle \models_{\mathscr{M},\beta} te_k \quad \text{then} \quad s' \not\models_{\mathscr{M},\beta} \varphi$$

•

The above definition captures the notion of *being potential* of the threat in its first condition with respect to the execution order of the plan (the threatening step *may* occur between producer and consumer) and with respect to the consistent variable assignments (*if* φ holds in the state in which step k is executed, it *may* become invalid afterwards). The detection function is sound because of the following argument: Given a plan P that is a solution to a problem π, let us assume that the causal threat analysis publishes a flaw with $\{te_i \xrightarrow{\varphi} te_j, te_k\} \in f^{det}_{\texttt{Threat}}(P, \pi)$. According to the flaw detection function, the ground linearizations of P thus include at least one sequence of operators in which an operator ground instance that corresponds to plan step te_k changes the world state such that the ground instance corresponding to plan step te_j is not executable. This contradicts P being a solution and therefore falsifies our assumption.

Please note that the causal link technique is a more restrictive implementation of the modal truth criterion [54] because the links "protect" the commitment for their annotated literals over the complete execution time interval between producer and consumer steps. Hence, it does not allow for the notion of a *White Knight* condition reestablishment (see also discussion in [300]).

Modification Generating Functions – $\mathfrak{Mod}_{\text{POCLP}}$

The POCLP-plan offers some more manipulatable elements than the language fragment for task assignment planning. The most prominent refinement operators of the current configuration introduce new steps in a plan for the purpose of goal establishment and establish causal links for explicitly documenting causal interactions.

Definition 3.13 (Insert Task)**.** For a given partial plan $P = \langle TE, \prec, VC, CL \rangle$, flaw $\mathtt{f}$, and domain model $D = \langle \mathscr{M}, \Delta, \mathtt{T} \rangle$ over a language $\mathscr{L}$, the modification generating function $f^{mod}_{\mathtt{InsertTask}}$ proposes to add a new plan step to P, a causal link that serves as a justification for that insertion, and a set of appropriate variable constraints. More formally:

$\langle \{te_p, te_p \xrightarrow{\varphi} te_c, v_1 \doteq \tau_1, \ldots v_n \doteq \tau_n\}, \emptyset \rangle \in f^{mod}_{\mathtt{InsertTask}}(P, \mathtt{f}, D)$ with

1. $te_c \in TE \cap comp(\mathtt{f})$ and $\varphi' \in comp(\mathtt{f})$ being a well-formed formula over $\mathscr{L}$.
2. $te_p \notin TE$ being a new task expression over a task schema in $\mathtt{T}$.
3. For all states s and all valuations β that are compatible with $VC \cup \{v_1 \doteq \tau_1, \ldots v_n \doteq \tau_n\}$ ($v_i \in \mathscr{V}$ and τ_i terms over $\mathscr{L}$ for $1 \leq i \leq n$) the following conditions hold:
   - te_p generates a formula φ'' such that $s \models_{\mathscr{M},\beta} (\varphi'' \Rightarrow \varphi)$.
   - $s \models_{\mathscr{M},\beta} (\text{prec}(te_c) \Rightarrow \varphi)$
   - $s \models_{\mathscr{M},\beta} (\varphi' \Rightarrow \varphi)$.

•

The third construction element of the insert task modification consists of three parts: First, the newly added task establishes at least the annotated condition φ. Second, the annotation matches the precondition of the consumer task (cf. plan consistency). Third, the annotation represents a necessary condition for satisfying the flawed formula φ' and thus a necessary element for establishing the consumer's precondition.

This definition may appear unfamiliar at the first glance, which is however due to our slightly more expressive task representation. If we assume that open precondition flaws as well as the preconditions and effects of tasks are given by lists of literals, the above conditions can be easily rephrased in terms of standard approaches as follows: the effect list of the new producer plan step te_p contains exactly the flawed literal φ (modulo variable substitutions) and so does the precondition of the flawed plan step. If we transfer this notion into our approach, we have to take into account that the open fragment of the task precondition may be any formula and not just a trivial one. For the same reason, the corresponding formula that is generated by the new producer task may not be trivial, as well. It is also worth noting that in this context a single task may not suffice to satisfy the open condition completely, hence the last item of the definition.

Task insertion is trivially a sound plan modification generating function, because it produces refinement operators that address the flawed plan elements and have a positive elementary modification balance.

The other central modification establishes a causal link between two tasks that are both already present in the plan.

Definition 3.14 (Add Causal Link)**.** For a given partial plan $P = \langle TE, \prec, VC, CL \rangle$, flaw $\mathtt{f}$, and domain model $D = \langle \mathscr{M}, \Delta, \mathtt{T} \rangle$ over a language $\mathscr{L}$, the modification generating function $f^{mod}_{\mathtt{AddCLink}}$ proposes to add a new a new causal link to P documenting the causal structure on which the plan is hereby committed, and a set of appropriate variable constraints.

$\langle \{te_p \xrightarrow{\varphi} te_c, v_1 \doteq \tau_1, \ldots v_n \doteq \tau_n\}, \emptyset \rangle \in f^{mod}_{\mathtt{AddCLink}}(P, \mathtt{f}, D)$ with

1. $te_p \in TE$, $te_c \in TE \cap comp(\mathtt{f})$, and $\varphi' \in comp(\mathtt{f})$ being a well-formed formula over $\mathscr{L}$.
2. $te_p \xrightarrow{\varphi} te_c \notin CL$ being a new causal Link in P.
3. For all states s and all valuations β that are compatible with $VC \cup \{v_1 \doteq \tau_1, \ldots v_n \doteq \tau_n\}$ ($v_i \in \mathscr{V}$ and τ_i terms over $\mathscr{L}$ for $1 \leq i \leq n$) the following conditions hold:
   - te_p generates a formula φ'' such that $s \models_{\mathscr{M},\beta} (\varphi'' \Rightarrow \varphi)$. If $te_p = te_{init}$ it may as well not generate $\neg\varphi''$.
   - $s \models_{\mathscr{M},\beta} (\mathrm{prec}(te_c) \Rightarrow \varphi)$
   - $s \models_{\mathscr{M},\beta} (\varphi' \Rightarrow \varphi)$.

•

The main aspects correspond to the previous plan modification. The first item in the last condition is extended to handle the initial state encoding in the null plan representation: Any action schema is described in terms of which changes it induces on the environment; any fact that is not contradicting this change, that means, any fact that is not "affected" by the change, persists. The semantics of the task that represents the initial state is slightly different such that everything that cannot be deduced positively is assumed to hold negatively, which is also known as the *closed world assumption*.

Regarding the soundness property, it holds for this modification generating function for the same reasons as it did for the task insertion above.

Triggering Function $\alpha_{\mathtt{POCLP}}$

When deploying the flaw detection and modification generating function above, the following class relationships unfold:

$$\alpha_{\mathrm{POCLP}}(\mathbb{F}_x) = \begin{cases} \mathbb{M}_{\mathtt{InsertTask}} \cup \mathbb{M}_{\mathtt{AddCLink}} & \text{for } x = \mathtt{OpenPrec} \\ \mathbb{M}_{\mathtt{AddVarConstr}} \cup \mathbb{M}_{\mathtt{AddOrdConstr}} & \text{for } x = \mathtt{Threat} \\ \alpha_{\mathrm{TAP}}(\mathbb{F}_x) & \text{otherwise} \end{cases}$$

The handling of open preconditions is an obvious correlation, because the plan step and causal insertions directly address the problem of an unsatisfied causal demand. The same holds for the third case, because the embedded sub-problem of preserving constraint consistency can be adequately dealt with the TAP configuration components.

Note that the treatment of causal threats introduces cross-configuration relations in the POCLP extension. Concerning the other combinations of POCLP and TAP functions, it is important to point out that there exist *possible* triggers, but not *necessary* ones: the treatment of open preconditions can be addressed by adding a variable constraint, thereby narrowing down the choices, but the given modification generators can insert the necessary variable constraints for themselves. There is no gain in terms of an increased applicability range by indirect flaw resolution in this case. The same holds for the POCLP modification generators, which could be used for co-designating variables and introducing orderings, but they do not provide additional support to the TAP functions.

Inference Functions – $\mathfrak{Inf}_{\mathtt{POCLP}}$

All operators that are causally linked need to be executed in an order that is consistent with the linking. As we have motivated in the section about the POCLP domain model specifics, we provide the ordering constraints that are necessary to make the causal link effective via the following inference:

Definition 3.15 (Ordering Constraint Inference)**.** For a given partial plan $P = \langle TE, \prec, VC, CL\rangle$ and domain model $D = \langle \mathscr{M}, \Delta, \mathtt{T}\rangle$, the inference function $f^{inf}_{\mathtt{OrdConstraint}}$ asks to add an ordering constraint to P if the plan contains causally linked operators that are not adequately ordered.

$\langle te_i \prec te_j, \emptyset\rangle \in f^{inf}_{\mathtt{OrdConstraint}}(P, \pi)$ if and only if for te_i and $te_j \in TE$:

1. For $te_i = l_i{:}t_i(\bar{v}_i)$, $te_j = l_j{:}t_j(\bar{v}_j)$: t_i or $t_j \in \mathscr{T}_p$
2. There exists a causal link te_{init} for some formula φ over $\mathscr{L}$
3. $te_i \prec te_j \notin \prec$

•

This inference function is trivially sound, because it merely adds (properly constructed) ordering constraints. We note that it is also essential for the configuration to terminate, because it manifests the commitments on the causal structure in the ordering relation of the plan.

Summary of the POCLP Configuration

With the above definition, the system configuration for performing partial-order planning can be summarized as the following tuples:

$$\begin{aligned}\mathfrak{C}_{\text{POCLP}} = \langle \{ & \overbrace{f^{det}_{\texttt{OrdIncons}}, f^{det}_{\texttt{VarIncons}}, f^{det}_{\texttt{OpenVarBind}}, f^{det}_{\texttt{UnordTask}}}^{\mathfrak{Det}_{\text{TAP}}}, f^{det}_{\texttt{OpenPrec}}, f^{det}_{\texttt{Threat}} \}, \\ & \{ \overbrace{f^{mod}_{\texttt{AddOrdConstr}}, f^{mod}_{\texttt{AddVarConstr}}}^{\mathfrak{Mod}_{\text{TAP}}}, f^{mod}_{\texttt{InsertTask}}, f^{mod}_{\texttt{AddCLink}} \}, \\ & \{ f^{inf}_{\texttt{OrdConstraint}} \}, \\ & \mathfrak{Str} \rangle\end{aligned}$$

Theorem 3.3 (Properness of $\mathfrak{C}_{\text{POCLP}}$)**.** $\mathfrak{C}_{\text{POCLP}}$ *is a proper system configuration in the sense of Def. 3.2.*

Proof. The soundness of the detection functions has been shown in the above definitions as well as in the respective section for TAP planning. $\mathfrak{Det}_{\text{POCLP}}$ is a complete set of sound detection functions, because for any plan that is no solution to a problem, at least one flaw is issued. This can be proven by contradition: Let P be a plan that is no solution to π and assume that no detection function in the POCLP configuration returns a flaw. When we go through the semantic explanations why P fails to satisfy the solution criteria we can easily identify that either the TAP sub-problem is not solved or the preconditions of at least one operator do not hold. The former is handled properly by the respective configuration subset, while the latter can be divided into two cases: either the precondition does not hold in any state during the ground linearizations of P or the condition does hold, is however undone before the critical operator occurs. Both cases are addressed by the open precondition and threat flaw detection functions, thus contradicting the initial assumption that no flaw is found in P.
$\mathfrak{Mod}_{\text{POCLP}}$ is a semi-complete set of sound modification generating functions. The property has been proven for the task assignment planning subset before. While soundness of the respective POCLP functions has been shown with the definitions, individual completeness follows directly from the functions addressing every possible flaw. The conditions under which no modification for inserting a task or a causal link is published are obviously characterizing non-refineable plans.
$\mathfrak{Inf}_{\text{POCLP}}$ is a set of sound inference functions.
According to the definition of the triggering function α_{POCLP}, the set of modification generating functions $\mathfrak{Mod}_{\text{POCLP}}$ corresponds to the set of detection functions $\mathfrak{Det}_{\text{POCLP}}$. □

Theorem 3.4 ($\mathfrak{C}_{\text{POCLP}}$ is Modification-Complete)**.** $\mathfrak{C}_{\text{POCLP}}$ *is modification-complete according to Def. 3.3.*

Proof. α_{POCLP} provides a modification generated from a function in $\mathfrak{Mod}_{\text{POCLP}}$ for any non-critical flaw that is found by a function in $\mathfrak{Det}_{\text{POCLP}}$. □

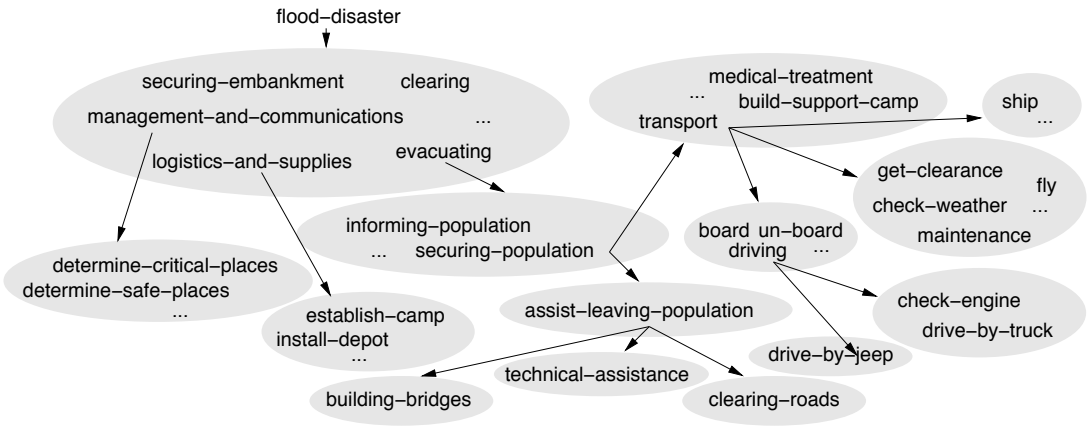


Figure 3.8: An typical decomposition hierarchy in HTN planning, taken from a disaster relief scenario [26].

3.2.3 Hierarchical Task Network Planning – $\mathfrak{C}_{\mathbf{HTNP}}$

This system configuration realizes hierarchical task network planning as it is described in the introductory section 1.1.3. The central idea is that an HTN planning-system does not try to synthesize sequences of primitive actions that achieve a specified goal state but instead takes a set of abstract *tasks* that are to be performed and tries to concretize them in a consistent way. The plan generation process consists of recursively decomposing abstract tasks into networks of sub-tasks (which are basically partial plans) until all steps in the plan have become primitive.[6] Such decomposition alternatives are explicitly declared per abstract task via so-called *methods*. A method is a user-specified implementation of an abstract action (see complex task semantics on p. 41) and a collection of such methods defines what is regarded to be a legal task implementation. Once a plan becomes primitive, it has to be examined whether its operators are executable or not. To this end, some of the causal reasoning from non-hierarchical partial order planning can be applied, that means, causal linking and threat resolution is used to establish and verify that the decomposition is executable. Note that this causal reasoning does not include the insertion of operators, therefore the only way to introduce plan steps is to do so via expansion networks. It is this context-dependency of operators which turns tasks into goal-like planning objectives and it is one of the key arguments for the greater expressivity of HTN planning over POCL planning [84].

Fig. 3.8 sketches a task hierarchy as it is induced by a set of decomposition methods within the domain model of a disaster relief scenario. A complete mission is specified by the root element of the decomposition hierarchy, the `flood-disaster` task. Solving the problem means finding an executable decomposition of this mission task. The shaded ellipses thereby denote the contexts of the respective task networks: the `flood-disaster` task is concretized into a plan that consists of securing the embankment of the nearby river, setting up a logistics centre, evacuating the population, and the like. The evacuation procedure includes informing the people and then securing them. The latter, the task `securing-population` in the center of the figure, can be decomposed in two ways, namely one network that represents assisting the population to leave the area by their own means and alternatively one for setting up a local infrastructure and transporting the population in groups to a secure location.

Domain Model Specifics

The HTN planning system configuration $\mathfrak{C}_{\text{HTNP}}$ extends domain models by the concept of user-defined task implementations or *method declarations*. It introduces the notion of *decomposition domain models*, which are structures of the form $D_{\text{HTNP}} = \langle \mathscr{M}, \emptyset, \mathtt{T}, \mathtt{M} \rangle$. The logical model $\mathscr{M}$ is thereby given as usual, while the

[6] The most important reference for HTN planning is certainly the work of Erol [83]. He discriminates primitive tasks and operators (the former without, the latter with effects), whereas our approach does not draw this technical distinction and allows for a direct decomposition into operators. Furthermore, Erol's approach delegates the concept of preconditions to constraints in the surrounding task networks.

state abstraction axioms are not used in this configurations and therefore $\Delta = \emptyset$. The task schemata $\mathtt{T}$ and the new method declarations $\mathtt{M}$ are described below.

For the operator representations in $\mathtt{T}$ we draw on the appropriate general formal definition 2.4 and the previously introduced $\mathfrak{C}_{\text{POCLP}}$ configuration, respectively. Concerning the abstract actions, an HTN domain model contains task schemata for complex task symbols; these schemata do however carry only trivial preconditions and effects. That means, all complex task schemata are of form $t(\bar{v}) = \langle \top, \top \rangle$.

In the component $\mathtt{M}$ of a decomposition domain model, the HTNP system configuration finds the method declarations, which are defined as follows: $m = \langle t(\bar{v}), \langle TE, \prec, VC, CL \rangle \rangle$. The complex task schema $t(\bar{v})$ is the task that is to be decomposed and $\langle TE, \prec, VC, CL \rangle$ is the appropriate task network into which an instance of $t(\bar{v})$ is to be decomposed. A task network's structure is thereby identical to that of a POCLP plan, except for methods being restricted to $CL = \emptyset$, because we want to reuse some techniques that we introduced for partial-order planning. Two more aspects are worth noting in this context: First, the variable constraint set VC may reference parameters of $t(\bar{v})$ in order to "pass parameter values" from the abstract task to subtasks in the expansion network. Second, methods do not possess condition-statements for controlling their applicability; expansion thereby relies solely on the semantics of complex tasks and plan refinements and is hence independent from the current plan generation situation. Such a declarative method definition is apparently a contrast to the highly sophisticated method concepts that are realized in the most famous HTN planning systems (see introduction to HTN planning on p. 13): O-PLAN and SIPE-2 incorporate a number of specialized linking variants for specifying how to establish inter-decomposition dependencies and the SHOP system uses a decomposition programming-language. For a more detailed discussion on the pros and cons of blending search control issues into the domain model see discussion in Sec. 1.1.3 (p. 14) and Sec. 2.8.1 (p. 72). In short, we prefer a declarative modelling style in order to be able to describe an essential quality of a domain model: the notion of consistency.

HTN domain-model consistency has to take into account method declarations, but since the intended use of a method is basically to substitute an instance of the specified task schema, the method-aware notion of consistency can be reduced to the respective definition of model consistency in our formal framework.

Definition 3.16 (Consistency of Decomposition Domain Models)**.** A decomposition domain model $D_{\text{HTNP}} = \langle \mathscr{M}, \emptyset, \mathtt{T}, \mathtt{M} \rangle$ over a given language $\mathscr{L}$ is called *consistent* if and only if the following conditions hold:

1. The included domain model $\langle \mathscr{M}, \Delta, \mathtt{T} \rangle$ is consistent (see Def. 2.8).
2. For every method $m \in \mathtt{M}$ with $m = \langle t(\bar{v}), \langle TE, \prec, VC, CL \rangle \rangle$, $t(\bar{v})$ is a complex task schema in $\mathtt{T}$ and the partial plan $\langle TE, \prec, VC, CL \rangle$ is consistent given the included domain model $\langle \mathscr{M}, \Delta, \mathtt{T} \rangle$ (see Def. 2.12).
3. For every complex task schema $t(\bar{v}) \in \mathtt{T}$ there exists at least one method $m \in \mathtt{M}$ such that $m = \langle t(\bar{v}), \langle TE, \prec, VC, CL \rangle \rangle$.
4. Recursive method definitions must allow for termination: Given a complex task schema $t(\bar{v}) \in \mathtt{T}$, let $m_1, \ldots, m_n$ be the appropriate methods in $\mathtt{M}$ and $\{\mathtt{T}_1, \ldots, \mathtt{T}_n\}$ the sets of those task schemata $\mathtt{T}_i \subseteq \mathtt{T}$ that occur in the respective decomposition network of method m_i. Every set $\mathtt{T}_i$ can be extended in an analogous way, that means, by adding all task schemata that occur in the networks of the methods of the respective complex task schemata in $\mathtt{T}_i$, resulting in sets $\{\mathtt{T}_{i_1}, \ldots, \mathtt{T}_{i_{n_i}}\}$. Let $\mathtt{T}_t^*$ denote the transitive closure of this process, in other words, the sets of all task schemata that are referenced in the decomposition hierarchy. $\mathtt{M}$ allows for termination if and only if the following holds:

$$\forall t(\bar{v}) \in \mathtt{T}\ \exists \mathtt{T}' \in \mathtt{T}_t^*\ t(\bar{v}) \notin \mathtt{T}'$$

•

The termination requirement (together with the finiteness of labeling symbols) is equivalent to the property that all expansions will eventually produce task networks that contain only primitive task expressions.

Please note that for the sake of simplicity, we gave a recursive definition of consistency: a domain model is consistent if all complex tasks are expanded by their methods into consistent plans, which in turn require

however consistent domain models. Let us sketch a formally more precise, inductive definition: a decomposition domain model without complex tasks and method definitions is consistent if it constitutes a consistent domain model for partial order causal link planning (see p. 92). A primitive decomposition plan is consistent to a (consistent) HTNP domain model if it is a consistent POCLP plan with respect to that domain model. Starting from these two base cases, we can extend our consistent "primitive" domain model by methods that add implementations for complex tasks such that the contained plan is consistent to the domain model before the adding. This basically corresponds to a strict "bottom-up" model construction. In view of this Noetherian induction (the extension of the domain model defines a well-founded relation) consistency corresponds to the fixed point of this model extension.

Problems and Solutions

A problem specification for HTNP is given by a domain model, an initial state, and an initial task network: $\pi_{\text{HTNP}} = \langle D_{\text{HTNP}}, s_{init}, \top, P_{init} \rangle$. The initial task network, which is given by $P_{init} = \langle TE_{init}, \prec_{init}, VC_{init}, \emptyset \rangle$, typically consists of a number of abstract plan steps, which are the tasks that have to be achieved by executing the solution plan. The trivial goal state specification reflects the fact that the only kind of "goals" in hierarchical task network planning are the complex tasks in P_{init}.

A plan P is a solution to an HTN problem specification π_{HTNP} if it is a solution in terms of the general definition 2.17 and P does not include a plan step that is an instance of an abstract task schema.

Since the primitive plan is essentially treated like in partial-order planning, we opt for the *null plan* (Def. 2.18) in order to maintain compatibility with the results from the $\mathfrak{C}_{\text{POCLP}}$ configuration. It can also be applied in the HTN context, because the null-plan representation preserves the initial task network. Analogously, we use the *solution plan* metaphor (see Def. 2.19) for verifying the general solution criteria in the view of causally annotated (primitive) plans.

Detection Functions – $\mathfrak{Det}_{\text{HTNP}}$

HTN planning is concerned about implementing an abstract plan. In the view of the HTNP configuration, the presence of complex task schema instances is therefore producing a flaw for each occurrence.

Definition 3.17 (Abstract Task)**.** For a given partial plan $P = \langle TE, \prec, VC, CL \rangle$ and HTNP planning problem π, the flaw detection function $f^{det}_{\texttt{AbstrTask}}$ indicates that a task expression $te \in TE$ is an instance of a non-primitive action schema.

$\{te\} \in f^{det}_{\texttt{AbstrTask}}(P, \pi)$ with $te = l : t(\bar{v})$, $te \in TE$, if and only if $t \in \mathscr{T}_c$. •

The detection function is obviously sound, because a solution plan does not contain abstract plan steps and is therefore not flawed.

For reasoning about the primitive plans' executability, the HTNP configuration relies on components from the system configuration $\mathfrak{C}_{\text{POCLP}}$ with its detection functions $f^{det}_{\texttt{OpenPrec}}$ and $f^{det}_{\texttt{Threat}}$. It furthermore inherits the respective set $\mathfrak{Det}_{\text{TAP}}$ from task assignment planning.

Modification Generating Functions – $\mathfrak{Mod}_{\text{HTNP}}$

Task expansion is the central modification in HTN planning and this is reflected in the HTNP configuration as well. An abstract task is decomposed by replacing the respective plan step by the expansion network that is deposited in an appropriate method as specified by the following definition.

Definition 3.18 (Expand Task)**.** Given a decomposition domain model $D = \langle \mathscr{M}, \Delta, \mathtt{T}, \mathtt{M} \rangle$, a partial plan $P = \langle TE, \prec, VC, CL \rangle$, and a flaw `f`, the modification generating function $f^{mod}_{\texttt{ExpandTask}}$ proposes for *every* occurrence of a complex task schema instance in `f` a decomposition according to *all* the appropriate method definitions in `M`.

$\langle TE_x \cup \prec_x \cup VC_x \cup \prec', \{te\} \cup \prec'' \rangle \in f^{mod}_{\texttt{ExpandTask}}(P, \mathtt{f}, D)$ with

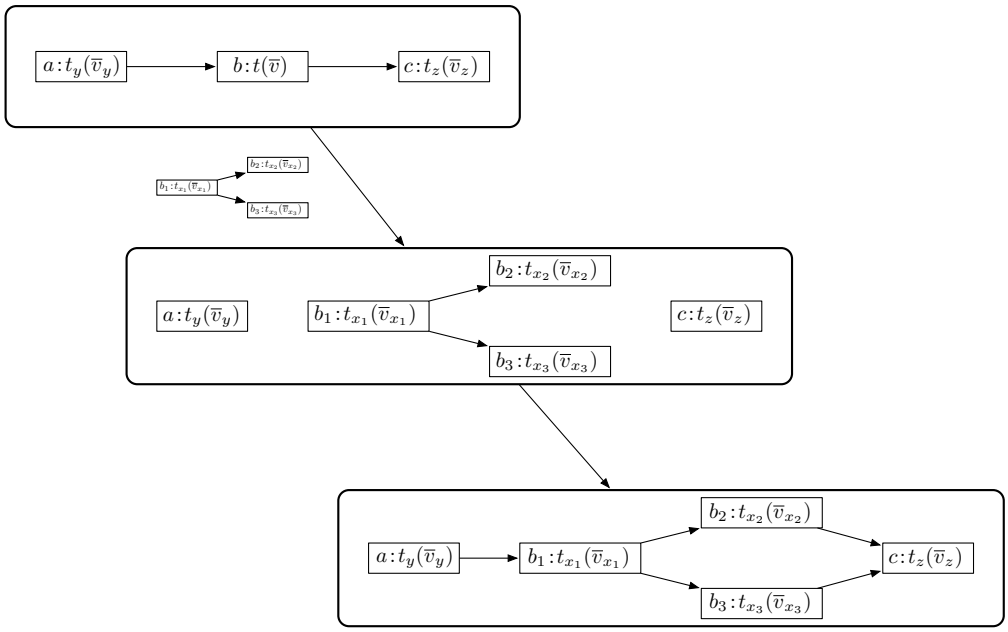


Figure 3.9: An example expansion in the HTNP configuration.

1. $te \in TE \cap comp(\mathtt{f})$ and $te = l{:}t(\overline{v})$ with $t \in \mathscr{T}_c$
2. $\prec''$ is the complete subset of ordering constraints $\prec$ in which te occurs, that means any constraint $te \prec te_P$ or $te_P \prec te$ for which $te_P \neq te$ in P
3. $m_x = \langle t(\overline{v}), \langle TE_x, \prec_x, VC_x, CL_x \rangle \rangle \in \mathtt{M}$
4. $TE_x \cap TE = \emptyset$, $\prec_x \cap \prec = \emptyset$, and $VC_x \cap VC = \emptyset$
5. $\prec'$ is the set of ordering constraints $te_x \prec te_P$ for which $te \prec te_P \in \prec$ and $te_x \in TE_x$, respectively ordering constraints $te_P \prec te_x$ for which $te_P \prec te \in \prec$ and $te_x \in TE_x$

•

The definition makes "replacing" more concrete: Items 1 and 2 define those plan elements that are to be removed, namely, the complex task expression and all related ordering constraints. In addition to the introduction of the components of the method's network (items 3 and 4), the $\mathbb{M}_{\mathtt{ExpandTask}}$ plan modifications adjust the ordering relation of the replaced task's context. That means, all steps that had been before or after the complex plan step are now before or after all plan steps in the expansion network, respectively (item 5). Fig. 3.9 illustrates the replacement.

Task expansion is issued by a sound plan modification generating function: First, it produces refinement operators for addressing the respective flaws, as the decomposition domain models' consistency provides the generating function with appropriate task implementations. Second, the expansion has a positive elementary modification balance, because it always adds more than it removes. Even in the case of a one-to-one expansion, that means, one abstract task is decomposed into a single primitive operator, the newly introduced co-designation constraints between the parameters yield soundness. The co-designation becomes necessary because on the one hand, the parameter binding is needed to ensure that the restrictions on the intended behaviour are maintained, and on the other hand, the parameters are necessarily syntactically disjoint for the task expressions in a partial plan.

We would like to point out that, due to the consistency of the decomposition domain model, an implementation of the method-based task expansion can always answer any flaw that references an abstract plan step. However, there is also the option to analyze the abstract task's context, to discover inconsistencies that will be induced by introducing the respective plan modification, and finally to decide to restrain the plan modification. Such an "optimization" is, of course, not affecting soundness or semi-completeness of the generating function (set).

The HTNP configuration furthermore employs $f^{mod}_{\texttt{AddCLink}}$ (Def. 3.14) of system configuration $\mathfrak{C}_{\text{POCLP}}$ for documenting causal interactions. It finally also inherits the entire set of modification generating functions from task assignment planning $\mathfrak{C}_{\text{TAP}}$ (Defs. 3.9 and 3.10).

Triggering Function α_{HTNP}

The triggering function for hierarchical task-network planning employs α_{POCLP} from the configuration of partial-order planning for dealing with the causal interactions of primitive operators. The newly introduced generator for task-expansion modifications is primarily triggered by abstract task flaws, but there is another application for it: it can answer open-precondition flaws. Causal support for a plan step may be enclosed in the implementation of an abstract task and causal link insertion therefore has "to wait" until appropriate producing plan steps appear in the partial plan – a typical indirect flaw resolution.

$$\alpha_{\text{HTNP}}(\mathbb{F}_x) = \begin{cases} \mathbb{M}_{\texttt{ExpandTask}} & \text{for } x = \texttt{AbstrTask} \\ \mathbb{M}_{\texttt{ExpandTask}} \cup \mathbb{M}_{\texttt{AddCLink}} & \text{for } x = \texttt{OpenPrec} \\ \alpha_{\text{POCLP}}(\mathbb{F}_x) & \text{otherwise} \end{cases}$$

Regarding a causal threat situation, the expansion of an abstract plan step cannot contribute to resolving the threat because the involved parties are primitive operators.

Inference Functions – $\mathfrak{Inf}_{\text{HTNP}}$

There are no additional inferences necessary to support HTN planning beyond those of partial-order causal-link planning, that means, $\mathfrak{Inf}_{\text{HTNP}} = \mathfrak{Inf}_{\text{POCLP}}$.

Summary of the HTNP Configuration

With the above function set definitions the following components constitute the HTNP configuration for performing hierarchical task-network planning:

$$\begin{aligned} \mathfrak{C}_{\text{HTNP}} = \langle \{ & \overbrace{f^{det}_{\texttt{OrdIncons}}, f^{det}_{\texttt{VarIncons}}, f^{det}_{\texttt{OpenVarBind}}, f^{det}_{\texttt{UnordTask}}}^{\mathfrak{Det}_{\text{TAP}}}, \overbrace{f^{det}_{\texttt{OpenPrec}}, f^{det}_{\texttt{Threat}}, f^{det}_{\texttt{AbstrTask}}}^{\mathfrak{Det}_{\text{POCLP}}} \}, \\ & \{ \overbrace{f^{mod}_{\texttt{AddOrdConstr}}, f^{mod}_{\texttt{AddVarConstr}}}^{\mathfrak{Mod}_{\text{TAP}}}, \overbrace{f^{mod}_{\texttt{AddCLink}}}^{\subset \mathfrak{Mod}_{\text{POCLP}}}, f^{mod}_{\texttt{ExpandTask}} \}, \\ & \{ \overbrace{f^{inf}_{\texttt{OrdConstraint}}}^{\mathfrak{Inf}_{\text{POCLP}}} \}, \\ & \mathfrak{Str} \rangle \end{aligned}$$

Note that although this configuration reuses many functions of the partial order causal link configuration, $\mathfrak{C}_{\text{HTNP}}$ is not an extension of $\mathfrak{C}_{\text{POCLP}}$ because of the missing task insertion modification.

Theorem 3.5 (Properness of $\mathfrak{C}_{\text{HTNP}}$)**.** *$\mathfrak{C}_{\text{HTNP}}$ is a proper system configuration in the sense of Def. 3.2.*

Proof. $\mathfrak{Det}_{\text{HTNP}}$ is a complete set of sound detection functions, because all included functions are sound and do not leave a non-solution plan unflawed. The latter property has been proven for the solution criteria violations with respect to the executability of operators; the remaining source of failure for a plan is the fact that not all tasks have been decomposed, which is verified by the HTNP detector for abstract tasks. The question whether or not compatible methods have been chosen for implementing the complex tasks is finally answered via executability analysis and constraint set coherence.

$\mathfrak{Mod}_{\text{HTNP}}$ is a semi-complete set of sound modification generating functions. The soundness of all included modification generators has been shown individually. Semi-completeness for the set is given, because it has already been shown that the property holds for the sub-configurations (that do not interact with the newly introduced functions), and the task expansion function guarantees by its definition that any flaw is addressed properly (Def. 3.18 and discussion).

Soundness of the inherited inference function sets has already been shown, $\mathfrak{Inf}_{\text{HTNP}}$ is therefore sound as well.

The set of modification generating functions $\mathfrak{Mod}_{\text{HTNP}}$ corresponds to the set of detection functions $\mathfrak{Det}_{\text{HTNP}}$ because the configuration provides suitable flaw classes for every generated modification class. □

Theorem 3.6 ($\mathfrak{C}_{\text{HTNP}}$ is Modification-Complete)**.** *$\mathfrak{C}_{\text{HTNP}}$ is modification-complete according to Def. 3.3.*

Proof. All included configuration components do answer any non-critical flaw. □

Now that we have specified the action-based partial-order configuration as well as the action abstracting hierarchical planning configuration, we are ready for the first enhanced configuration, namely that for *hybrid* planning.

3.3 Enhanced System Configurations

3.3.1 Hybrid Planning – $\mathfrak{C}_{\text{HYBP}}$

In this first integrating system configuration we are extending the partial-order and HTN configurations such that the extension implements the hybrid planning paradigm like we have discussed in the introductory Sec. 1.1.4. We would like to repeat at this occasion that for many real-world domains, approaches that integrate hierarchical task decomposition with action-based planning prove to be most appropriate (cf. Sec. 1.1.4). On the one hand, human-expert knowledge can be represented and exploited by means of tasks and methods, which describe how abstract tasks can be decomposed into predefined plans that accomplish them.[7] On the other hand, the flexibility to come up with non-standard solutions (synthesizing plans from scratch) or to overcome incompleteness of the explicitly defined solution space (completing under-specified procedural knowledge) results from the option to insert tasks and primitive actions like in POCL planning. Fig. 3.10 illustrates this flexibility: the abstract transport action can be decomposed into a task network that implements the basic way of transportation, loading the good into a vehicle, moving that vehicle to the destination, and unloading the good there. Non-standard tasks that have to be performed on demand include refuelling the vehicle or bringing the vehicle to the loading location.

Hybrid planning enables furthermore reasoning about causal interactions on any level of abstraction. That means, threat handling and condition establishment can be initiated and performed on more abstract and hence easier-to-manage plans. The precise abstraction can be chosen opportunistically, because all decisions and implicit commitments are propagated into the refinements.

Domain Model Specifics

Hybrid planning makes use of all domain model components that have been introduced for partial-order and hierarchical task network planning above. The $\mathfrak{C}_{\text{HYBP}}$ system configuration thus works on hybrid domain models $D_{\text{HYBP}} = \langle \mathscr{M}, \Delta, \mathtt{T}, \mathtt{M} \rangle$. The logical model $\mathscr{M}$ is given as usual and Δ is a set of state abstraction axioms as described in Def. 2.1. The task schemata $\mathtt{T}$ comprise primitive operators and complex tasks that both carry preconditions and effects, respectively postconditions. As we employ the operator representation of $\mathfrak{C}_{\text{POCLP}}$, all classes of actions are now built over the same syntactic features.[8] The method declarations

[7] Note that we continue advocating the declarative modelling paradigm. The captured knowledge is that of a *domain expert* and not of a *planning-for-that-domain expert*. That means, the modelled decomposition is a natural one in terms of the application domain.

[8] Remember that, technically speaking, complex tasks' postconditions are arbitrary formulae while an operator postcondition is restricted to a conjunction of literals. In most practical cases however, task postconditions adhere to the same conjunctive structure.

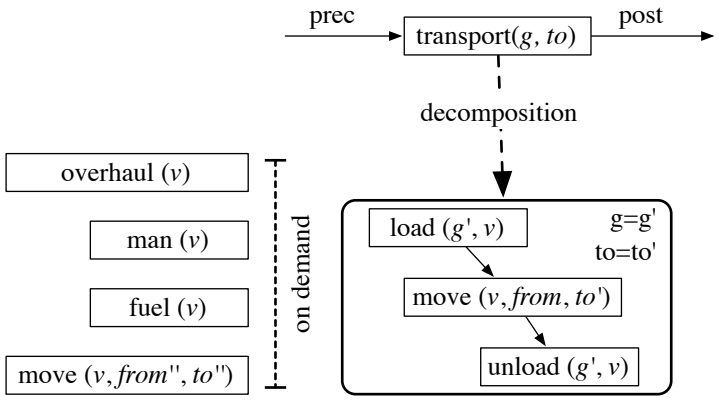


Figure 3.10: The completion of user-defined procedural knowledge in hybrid planning. After the decomposition of the complex transportation task, any of the four support tasks may be added on demand.

M are the same kind of user-defined task implementations as presented in the HTNP-configuration model above. The only difference is that task networks in hybrid planning are not restricted to empty causal link sets.

The previously introduced concept of decomposition domain model consistency has been defined in the context of $\mathfrak{C}_{\text{HTNP}}$, in which there is no semantic relationship between complex tasks and their implementing networks. This "deficiency" of HTN planning is addressed in hybrid planning: The user-defined implementation has to comply with the abstract state transition that is induced by the complex task.

Definition 3.19 (Consistency of Hybrid Domain Models)**.** A hybrid domain model specification $D_{\text{HYBP}} = \langle \mathscr{M}, \Delta, \mathtt{T}, \mathtt{M} \rangle$ over a given language $\mathscr{L}$ is called *consistent* if and only if the following conditions hold:

1. The included decomposition domain model is consistent (see Def. 3.16).
2. For every method $m \in \mathtt{M}$ with $m = \langle t(\overline{v}), \langle TE_1, \prec_1, VC_1, CL_1 \rangle \rangle$, the included decomposition network $\langle TE_1, \prec_1, VC_1, CL_1 \rangle$ is an implementation of $t(\overline{v})$ (see page 41).

•

Although verifying the consistency of a given hybrid domain model is considerably more complex than validating the same property for a decomposition or POCLP domain model, it is a worthwhile expenditure of (offline) computation time. The procedure becomes feasible by "unfolding" state abstraction axioms, thereby breaking down the causal structure of the tasks to the most primitive level, at which classical causal analysis can be conducted (see also Sec. 2.8.4 of the formal framework discussion).

The hybrid plan data structure is the same as for the general partial plan (Def. 2.9) with the consistency criteria given in Def. 2.12.

Problems and Solutions

The hybrid planning problem specification is simply a combination of HTN- and POCL-like descriptions: $\pi_{\text{HYBP}} = \langle D_{\text{HYBP}}, s_{init}, \mathfrak{s}_{goal}, P_{init} \rangle$. Like in hierarchical task network planning, all (complex) tasks in the initial plan P_{init} have to be accomplished in the given initial situation and, like in partial-order planning, the plan has to satisfy the goal state specification.

That means, a plan P is a solution to the hybrid planning problem, if and only if P is a solution to the included POCLP and HTNP problem, respectively, and that means if it is a solution to the general problem

$\langle D, s_{init}, \mathfrak{s}_{goal}, P_{init} \rangle$ as we have defined it in Def. 2.17. Note that the reduction on the general domain model D is done by "removing" the user-defined methods from the hybrid domain model D_{HYBP}.

Like we did for the extended configurations, the hybrid planning configuration represents its problems and solutions as *null plans* (Def. 2.18) and solutions plans (Def. 2.19), respectively.

Detection Functions – $\mathfrak{Det}_{\text{HYBP}}$

Setting up detection functions for hybrid planning is very simple: We can just employ the complete set $\mathfrak{Det}_{\text{HTNP}}$ of the hierarchical task network configuration. For triggering the expansion of complex tasks in the plan the detection function $f^{det}_{\texttt{AbstrTask}}$ is used. The treatment of causal interactions is postponed in HTN planning indirectly by the fact that no complex task can participate in precondition and effect calculations. By extending the representation as described above, the abstract tasks are automatically examined properly by the open precondition and threat detection functions that have been introduced by the partial-order configuration. The task assignment planning flaws are also directly transferable without change.

Modification Generating Functions – $\mathfrak{Mod}_{\text{HYBP}}$

Hybrid planning has a particular focus on the task expansion technique, because the causal interactions that have been established on the abstract level have to be carried properly into the refinement. Note that in principle, a loss of causal links during a task decomposition would not endanger the plan generation as such, basically because causal links are not carrying information that is relevant for the plans' semantics. They do however represent a considerable amount of commitment information that is too valuable to be re-established after every expansion of a task.

In order to guarantee a proper re-establishment of information about commitment to a causal structure, our hybrid planning configuration employs the following update of the task expansion modification generator.

Definition 3.20 (Expand Task – Redefined)**.** Given a hybrid domain model $D = \langle \mathscr{M}, \Delta, \mathtt{T}, \mathtt{M} \rangle$, a partial plan $P = \langle TE, \prec, VC, CL \rangle$, and a flaw $\mathtt{f}$, the redefined modification generating function $f^{mod}_{\texttt{ExpandTask}}$ proposes for *every* occurrence of a complex task schema instance in $\mathtt{f}$ a decomposition according to *every* appropriate method definition in $\mathtt{M}$ as follows.

$$\langle \underbrace{TE_x \cup \prec_x \cup VC_x \cup CL_x}_{\text{expansion network}} \cup \overbrace{\prec' \cup VC' \cup CL'}^{\text{new context}}, \quad \underbrace{\{te\}}_{\text{complex task}} \quad \cup \overbrace{\prec'' \cup CL''}^{\text{old context}} \rangle \in f^{mod}_{\texttt{ExpandTask}}(P, \mathtt{f}, D)$$

The expansion network is thereby substituting the complex task in consideration of its temporal and causal context.

More formally, the plan modification consists of the following components:

1. $te \in TE \cap comp(\mathtt{f})$ and $te = l : t(\overline{v})$ with $t \in \mathscr{T}_c$
2. $\prec''$ is the complete subset of ordering constraints $\prec$ in which te occurs, that means any constraint $te \prec te_P$ or $te_P \prec te$ for which $te_P \neq te$ in P
3. CL'' is the complete subset of causal links CL in which te occurs, that means an link $te \xrightarrow{\varphi} te_P$ or $te_P \xrightarrow{\varphi} te$ for which $te_P \neq te$ in P
4. $m_x = \langle t(\overline{v}), \langle TE_1, \prec_1, VC_1, CL_1 \rangle \rangle \in \mathtt{M}$
5. $TE_x \cap TE = \emptyset$, $\prec_x \cap \prec = \emptyset$, $VC_x \cap VC = \emptyset$, and $CL_x \cap CL = \emptyset$
6. $\prec'$ is a set of new ordering constraints $te_x \prec te_P$ for every $te \prec te_P$ in P and $te_x \in TE_x$, respectively $te_P \prec te_x$ for every $te_P \prec te$ in P and $te_x \in TE_x$
7. VC' and CL' are sets of new variable constraints and causal links such that:

a) For each incoming causal link $te_P \xrightarrow{\varphi} te \in CL''$, let $TE'_x \subseteq TE_x$ be a set of task expressions such that for any state s and valuation β that is compatible with $VC \cup VC_x \cup VC'$:

$$s \models_{\mathscr{M},\beta} \left(\bigwedge_{te_x \in TE'_x} \text{prec}(te_x) \right) \Rightarrow \varphi$$

TE'_x is thereby a minimal set in the sense that if any task expression is removed, the equation does not hold. Let furthermore φ_i be a formula for each te_i in TE'_x such that

$$s \models_{\mathscr{M},\beta} (\text{prec}(te_i) \Rightarrow \varphi_i) \wedge (\varphi \Rightarrow \varphi_i)$$

For each pair of plan steps te_{x_i} and formulae φ_i, the set CL'' includes a causal link $te_P \xrightarrow{\varphi_i} te_{x_i}$.

b) For each outgoing causal link $te \xrightarrow{\varphi} te_P \in CL''$, let $TE''_x \subseteq TE_x$ be a set of task expressions such that for any state s and valuation β that is compatible with $VC \cup VC_x \cup VC'$:

$$s \models_{\mathscr{M},\beta} \left(\bigwedge_{te_x \in TE''_x} \text{post}(te_x) \right) \Rightarrow \varphi$$

TE''_x is thereby a minimal set in the sense that if any task expression is removed, the equation does not hold.[9] Let furthermore φ_i be a formula for each te_i in TE''_x such that

$$s \models_{\mathscr{M},\beta} (\text{post}(te_i) \Rightarrow \varphi_i) \wedge (\varphi \Rightarrow \varphi_i)$$

For each such pair of plan steps te_{x_i} and formulae φ_i, the set CL'' includes a causal link $te_{x_i} \xrightarrow{\varphi_i} te_P$.

•

The annotated plan modification structure at the beginning of the above definition is reflected by the formal criteria in the following way: Item 1 addresses the complex task that is referenced by the flaw and that is therefore to be expanded. Since the plan step is going to be removed, its temporal context (item 2) and causal dependencies (item 3) are removed as well. The parameters of the task remain in the variable constraints in order to preserve binding commitments. Item 4 relates the complex task with one of the user-defined methods. The introduction of the method's new task network components into the partial plan is described in item 5. "Applying" the method implies some management with respect to task-expression labels and variable names: the new components have to be provided with appropriate identifiers that are unique in the respective partial plan. Items 6 and 7 represent the re-establishing of the temporal, respectively causal context for the task expressions in the expansion network. The ordering constraints that are imposed on the abstract plan step oblige its implementation to respect the same temporal restrictions and therefore all sub-tasks are ordered according to the complex task. The rationale behind the causal link manipulations is to "redistribute" causality among appropriate sub-tasks. That means, first, that incoming causal links (item 7a) are re-established on those steps in the expansion network that carry a precondition of which the complex task's precondition is an abstraction. Second, those concrete effects in the expansion network that are sufficient for the abstract effects of the complex tasks on which the plan has been committed are the new origins for the corresponding outgoing causal links (item 7b).

Fig. 3.11 illustrates the re-establishment of the causal structure in an example situation. The (complex or primitive) tasks te_p and te_c are producing two precondition atoms $P_1(y)$ and $P_2(z)$ for the complex plan step te, respectively consuming its effects in terms of an atom $Q(z)$. The right hand side of the figure depicts a task network for expanding the complex task, consisting of three sub-tasks that are temporally and causally linked in a way such that te_1 and te_2 are to be executed before te_3.

The shaded area in the figure shows the application of an appropriate plan modification after definition items 1–6 have been put into effect; the depicted atomic formulae represent some precondition and effect

[9] If the respective plan step is obtained from a primitive task schema, we may assume that post(te_x) denotes a formula that is generated by te_x.

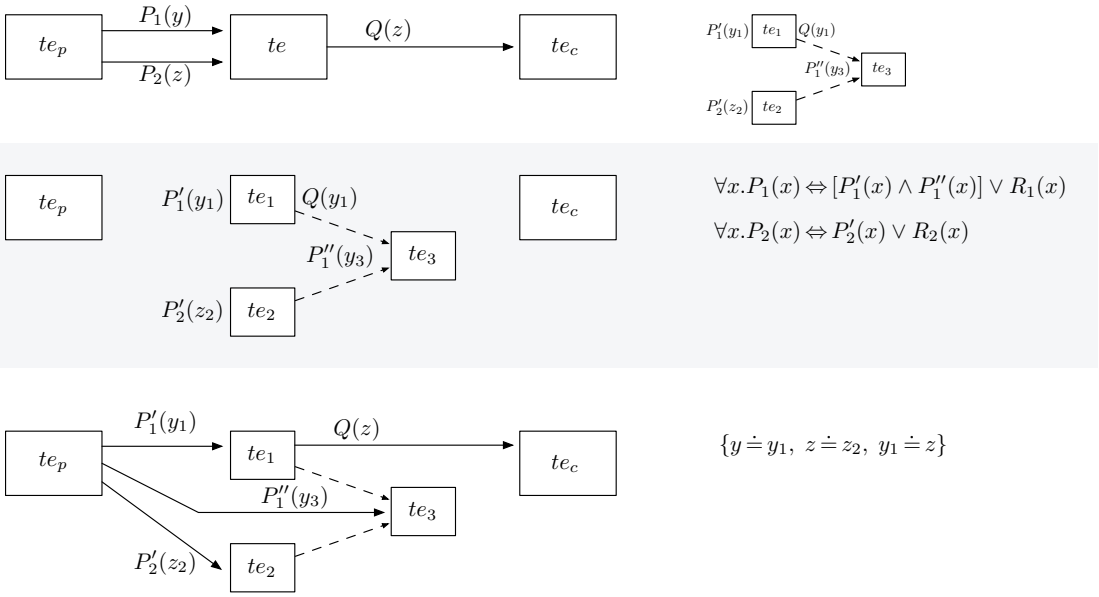


Figure 3.11: Task expansion in the hybrid planning configuration – an example for the re-establishment of causality.

literals of the respective plan step, and on the right side two state abstraction axioms are given. Atoms over relation Q are not refined and task te_1 carries a suitable atom in its postcondition: the commitment to the more abstract causal structure as it is represented by the removed causal link $te - Q(z) \rightarrow te_c$ is therefore re-established (cf. definition item 7b) by introducing the "new" causal link $te_1 - Q(z) \rightarrow te_c$ and the corresponding variable co-designation constraint $y_1 \doteq z$.

Regarding the incoming causal links (cf. definition item 7a), let us begin with examining the inheritor of the abstract causal link $te_p - P_2(z) \rightarrow te$. According to the second state-abstraction axiom, an atom over P_2 represents an abstraction of an atom over P'_2 or R_2 – the former is the symbol that is used in the precondition of te_2. The causal commitment is therefore refined into the link $te_p - P'_2(z_2) \rightarrow te_2$ and the co-designation constraint $z \doteq z_2$. Note that making the causal commitment more specific imposes further constraints on future developments of the current plan. After the depicted expansion, te_p is intended to produce an effect that subsumes the more concrete atom $P'_2(z_2)$, and if te_p is abstract, its future expansion will have to be *compatible* in this sense with the expansion of te.

For the second causal link, that is $te_p - P_1(y) \rightarrow te$, note the first state-abstraction axiom, which relates the abstract atom $P_1(x)$ with the conjunction $P'_1(x) \wedge P''_1(x)$ or alternatively with the atomic formula $R_1(x)$. The latter is apparently not supported by the sub-tasks in the expansion network but the former is: te_1 carries a positive literal over the P'_1 relation symbol in its precondition, and te_3 does so for P''_1. In this situation, the incoming abstract causal link is *split* into two more concrete links in the partial plan, the depicted $te_p - P'_1(y_1) \rightarrow te_1$ and $te_p - P''_1(y_3) \rightarrow te_3$ (together with the appropriate co-designation constraints). The third segment of Fig. 3.11 shows the plan after the described application of the plan modification is completed.

The example shows that the redistribution of causality does not necessarily result in plans in which the incoming links are connected to the leading sub-tasks and outgoing links to the terminal steps. Moreover, it is this way of causality handling that provides hybrid planning with the flexibility to produce *overlapping* implementations of complex tasks. It is also a benefit of the causal link introduction in the partial-order configuration $\mathfrak{C}_{\text{POCLP}}$ that causal links do not automatically induce an ordering constraint on the involved plan steps: We first commit to the causal structure and maintain flexibility (for overlapping parts of the plan) until temporal commitment becomes inevitable.

Note that according to Def. 3.20 the modification generating function $f^{mod}_{\texttt{ExpandTask}}$ has to account two combinatorial factors when computing the expansion of a specific flawed complex task: First, it has to consider

all available method specifications. It is thereby not relevant whether or not the generator filters out those methods that will produce inconsistent plans as such flawed plans will be discarded later. Second, if the causal structure cannot be re-established unambiguously then one plan modification has to be generated *for each permutation*. This is the case if either two tasks in the expansion network carry the appropriate preconditions or effects, respectively, or if the decomposition axioms allow for an ambiguous interpretation of the network; in both scenarios two minimal sets for re-linking exist (TE'_x in item 7). For each complex task te, the hybrid expansion generator therefore issues |methods for te| · |linking permutations| many plan modifications.

Regarding soundness of the redefined task expansion, the arguments concerning the initial modification generator in Def. 3.18 remain valid and therefore $f^{mod}_{\texttt{ExpandTask}}$ is sound.

Triggering Function α_{HYBP}

The purely hierarchical and action-based aspects of hybrid planning are perfectly covered by the triggering functions of the HTNP and POCLP configurations, which are therefore reused in $\mathfrak{C}_{\text{HYBP}}$. In particular, it has to be emphasized that causal reasoning in the POCL configuration components is based on the semantics of states and hence takes into account the state-abstraction axioms. It thereby automatically mediates between producing and consuming plan steps at different levels of abstraction.

$$\alpha_{\text{HYBP}}(\mathbb{F}_x) = \begin{cases} \mathbb{M}_{\texttt{ExpandTask}} \cup \alpha_{\text{POCLP}}(\mathbb{F}_x) & \text{for } x = \texttt{OpenPrec} \\ \mathbb{M}_{\texttt{ExpandTask}} \cup \alpha_{\text{POCLP}}(\mathbb{F}_x) & \text{for } x = \texttt{Threat} \\ \alpha_{\text{HTNP}}(\mathbb{F}_x) & \text{otherwise} \end{cases}$$

The updated expand task modification substitutes the previous HTN version in the ordinary task expansion situation as well as in the search for a suitable precondition establisher. Since in hybrid planning the causal interactions also concern abstract tasks, α_{HYBP} has to consider task expansion as an additional means for resolving causal threats. The rationale for this relationship is twofold: First, the conflict may be resolved immediately if the variable constraints that are included in the expansion network ruled out the respective co-designations (*separation by expansion*). Second, refining the abstract causality into a structure of finer granularity provides the configuration with additional ordering options (promotion and demotion on atomic complex tasks versus *overlapping implementations*).

Inference Functions – $\mathfrak{Inf}_{\text{HYBP}}$

There are no additional inferences necessary to support hybrid planning beyond those of hierarchical task-network planning. We therefore have $\mathfrak{Inf}_{\text{HYBP}} = \mathfrak{Inf}_{\text{HTNP}}$.

Summary of the HYBP Configuration

$\mathfrak{C}_{\text{HYBP}}$, the system configuration for hybrid planning, is composed of the following function sets:

$$\begin{aligned} \mathfrak{C}_{\text{HYBP}} = \langle \{ & \overbrace{f^{det}_{\texttt{OrdIncons}}, f^{det}_{\texttt{VarIncons}}, f^{det}_{\texttt{OpenVarBind}}, f^{det}_{\texttt{UnordTask}}}^{\mathfrak{Det}_{\text{TAP}}}, \overbrace{f^{det}_{\texttt{OpenPrec}}, f^{det}_{\texttt{Threat}}}^{\mathfrak{Det}_{\text{POCLP}}}, \overbrace{f^{det}_{\texttt{AbstrTask}}}^{\mathfrak{Det}_{\text{HTNP}}} \}, \\ & \{ \overbrace{f^{mod}_{\texttt{AddOrdConstr}}, f^{mod}_{\texttt{AddVarConstr}}}^{\mathfrak{Mod}_{\text{TAP}}}, \overbrace{f^{mod}_{\texttt{InsertTask}}, f^{mod}_{\texttt{AddCLink}}}^{\mathfrak{Mod}_{\text{POCLP}}}, f^{mod}_{\texttt{ExpandTask}} \}, \\ & \{ \overbrace{f^{inf}_{\texttt{OrdConstraint}}}^{\mathfrak{Inf}_{\text{POCLP}}} \}, \\ & \mathfrak{Str} \rangle \end{aligned}$$

Theorem 3.7 (Properness of $\mathfrak{C}_{\text{HYBP}}$)**.** $\mathfrak{C}_{\text{HYBP}}$ *is a proper system configuration in the sense of Def. 3.2.*

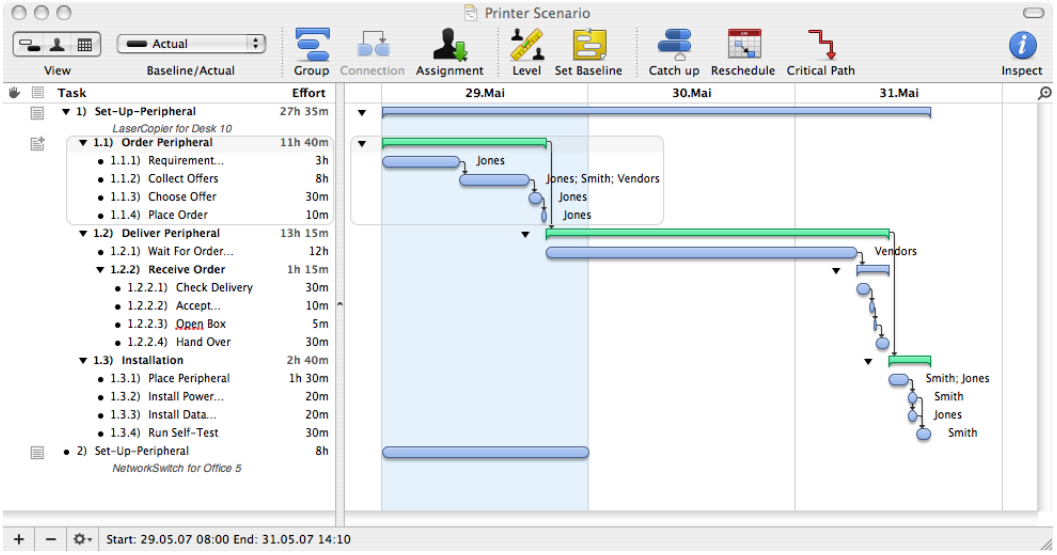


Figure 3.12: The peripheral-delivery example with additional temporal information. The screenshot is taken from OmniPlan, ©2006 The Omni Group.

Proof. As $\mathfrak{C}_{\text{HYBP}}$ extends $\mathfrak{C}_{\text{POCLP}}$ as well as $\mathfrak{C}_{\text{HTNP}}$, the respective properness results of theorems 3.3 and 3.5 are sufficient for this configuration. The redefinition of the modification generator for task expansion does thereby not affect semi-completeness of the $\mathfrak{Mod}_{\text{HYBP}}$, respectively invalidate its correspondence property. □

Theorem 3.8 ($\mathfrak{C}_{\text{HYBP}}$ is Modification-Complete)**.** $\mathfrak{C}_{\text{HYBP}}$ *is modification-complete according to Def. 3.3.*

Proof. The property follows directly from theorems 3.4 and 3.6. □

3.3.2 Temporal Planning – $\mathfrak{C}_{\text{TTAP}}$

The ability to reason about temporal phenomena and their relationships is an important issue for many applications of artificial intelligence and so it is also for the area of planning. The previous configurations suggested that actions are atomic and hence instantaneous events that induce change on the otherwise timeless states. Since plans produce sequences of successive states, our conceptualization of time can so far be regarded as an implicit and qualitative one. However, most real-world application domains require an explicit representation of quantitative temporal knowledge. This includes knowledge about at what time plan execution is intended to start, what the defined deadlines are for achieving the planned goals, and how long an action takes to be executed. As a first step towards implementing a system with real-world-suitable scheduling capabilities, we introduce in the following sections an extension of task assignment planning with a simple quantitative temporal model. The rationale behind $\mathfrak{C}_{\text{TTAP}}$ is to present a system configuration that encapsulates the functionality for temporal reasoning at least premises with respect to the domain model. Since we extend the TAP configuration, temporal planning can be easily integrated with any complex system configuration. In particular, if a more sophisticated temporal model is needed, this configuration can be safely replaced.

An example for a task assignment plan with additional temporal information is given in Fig. 3.12. In the depicted project definition, starting time for the first task is Tuesday, May 29th, 2007, at eight o'clock. Employee Jones obtains the requirements for a laser copier, which is to be purchased for "desk 10", within three hours (task 1.1.1), before he is picking up the phone to collect appropriate offers within the following working day, and so on. As we have noted for the TAP configuration, commercial project planning tools do neither reason about parameter dependencies (they require parameter bindings or, according to their terminology, resource assignments) nor perform any search. They do however support a set of temporal

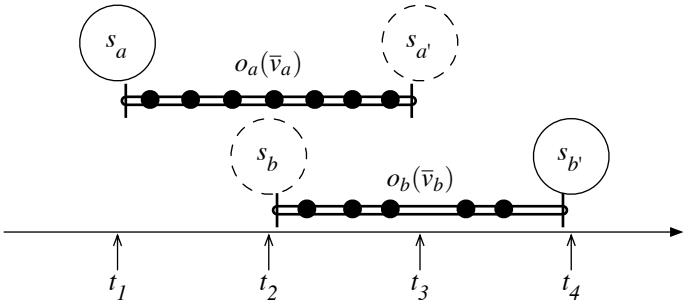


Figure 3.13: The state-transition view on temporally overlapping operators.

constraint types like precedence of tasks, task start times and deadlines, follow-up tasks, etc., which goes beyond the representation capabilities of the current task assignment planning domain model. This chapter's discussion section will address enhanced temporal representations.

In order to provide a coherent picture, the temporal dimension has to be added carefully to our formal framework. On the one hand, it appears natural to assume that time is an integral part of a state description, for example, as a flexible constant that represents a global clock. This is consistent with our view of states as arbitrarily small temporal entities, "snapshots of the world", or "moments". As a consequence, time passes *between* states, and the clock proceeds explicitly via corresponding term updates that increase the time constant accordingly. On the other hand, introducing parallelism in this way becomes a problematic issue, since the underlying semantics of our approach is inherently connected to *sequences* of state transitions. As we have argued in Sec. 2.8.3, the formal framework does not exclude pseudo-parallel operators, because they represent atomic transitions that may occur arbitrarily close to each other and therefore appear as simultaneous events to an observer. Temporally extended state manipulations, however, introduce the problem that is depicted in Fig. 3.13: Let us assume two operators $o_a(\overline{v}_a)$ and $o_b(\overline{v}_b)$ with $o_a(\overline{v}_a)$ being executed at time point t_1 and ending at t_3 and with $o_b(\overline{v}_b)$ lasting from t_2 to t_4, respectively. The black dots on the operator time lines represent the corresponding elementary operations, that means, updates of the flexible symbol interpretations[10]. It is important to note that although we know in which order these elementary operations occur, we cannot deduce much at which point in time because the "intermediate" interpretation may not have updated its clock yet (not to speak of other undefined interpretations); the elementary operations are below our modelling horizon and are therefore only virtual time points. While states s_a and $s_{b'}$ are covered by our operator semantics, states s_b and $s_{a'}$ are not because it cannot be deduced from the model which "portion" of $o_a(\overline{v}_a)$'s elementary operations is already implemented at the exact time point at which $o_b(\overline{v}_b)$ starts. For the same reason, it is specified by the temporal relation that $o_a(\overline{v}_a)$'s effects hold in $s_{a'}$ but we do not have any information about the sub-atomic state changes that have been induced by $o_b(\overline{v}_b)$ up to this point. Last, but not least, the update of the clock is neither necessarily the last operation in the chain nor can it be synchronized such that the time update of the earlier operator occurs before that of the later operator. These problems seem to have been addressed in temporal extensions to PDDL by interpreting actions as triples, which consist of an initial, intermediate, and conclusive virtual sub-operator [99]. However, since these action fragments are causally atomic entities by themselves, we feel that our problem of finding an adequate representation has been solved only superficially.

What appears as a dilemma turns out to be a valuable insight into the nature of temporal planning. First, it does not necessarily make sense to incorporate the progression of time via operator updates. If we, instead, let time "pass" in the sense of an external and uncontrollable event and if we understand the temporal extension of actions as rather "watching" than setting the clock at the beginning and at the end of an operator execution, then we do not have to address the problem of synchronizing term updates or waiting actively for time to pass. As a positive side effect, the time points do not multiply: For computing the update of the clock, its initial value is assessed in s_b. With State $s_{a'}$ in Fig. 3.13 happening potentially before the actual update operation, a recalculation of that value is implied and a new clock value has to be created and

[10] Please note that although we are not going to use the state changes in this configuration, we do however have to deal with it later.

introduced. The second insight is that an explicit treatment of parallel actions does not contribute to our framework, because we cannot infer any coherent state description for the (semi-) parallel threads on this level of abstraction. We therefore suggest to retain the atomicity of operators.

The conclusion is simply the following: abstract causal reasoning should be done by means of the framework while the more detailed temporal reasoning is performed in a meta-level calculus that is synchronized with the framework via a symbolic clock representation. Let us briefly motivate why this is a conceptually clean and semantics preserving way of integrating time into the system configuration.

Assume that we deploy a temporal reasoning mechanism for dealing with the time points and durations like those shown in Fig. 3.13. If this mechanism infers that t_3 has to happen strictly before t_2 then the underlying plan has to add an ordering constraint between operator $o_a(\overline{v}_a)$ and $o_b(\overline{v}_b)$ and vice versa. Plan executability is determined in the usual way according to Def. 2.11. In these simple situations, it is easy to see that the temporal subsystem is a mere book-keeping entity that performs all necessary computations and that communicates with the formal framework interpretation of the partial plan technically via the shared symbols for the four time points t_1 to t_4.

The intervals $[t_1,t_3]$ and $[t_2,t_4]$ are allowed to overlap in any way (which ways are admissible is depending on the underlying temporal model) if and only if this is consistent with the ordering constraints and if two operators do not interfere on the shared time slice, say, on the interval $[t_2,t_3]$. This is apparently an exact correspondence with respect to the causal threat situation in partial-order planning (Sec. 3.2.2), because the potentially interfering elementary modifications may appear at any point in time and in particular in the worst case, that is, during the shared section. Hence, the time interval that has to be protected is in every case that of the complete operator duration. In other words: two actions may overlap if and only if their preconditions and effects do not interfere negatively, otherwise they have to be *strictly ordered*. As a consequence, a partial plan with overlapping durative actions can only constitute a solution to a given planning problem if the non-temporal representation is a solution in the usual sense of Def. 2.17.

Concerning causal dependencies between plan steps, the above principle has to be applied as well, that means, every state feature may be set at the latest possible point in time by the effect producer, while it may be required at the earliest possible point in time by the condition consumer. An action is consequently only executable if its precondition is satisfied by the initial state and the effects of the *strictly preceding* plan steps.

We would like to note that this domain model design may appear to employ a rather strict notion of temporally extended actions (cf. PDDL [99], temporal relation algebra [5], etc.). It is however reasonably concise, that means, its semantics are clear and there are no highly specialized construction features to be considered [188], and the puristic operator concept will be augmented by our abstraction mechanisms for the hierarchical planning configurations below.

These considerations apply, of course, to all kinds of sharable, strictly consumable resources. Time is however the only practically relevant resource of that kind and therefore deserves a specific treatment.

Domain Model Specifics

The domain model representation for the TTAP system configuration is based on the task assignment planning model, that means, $D_{\mathrm{TTAP}} = \langle \mathscr{M}, \emptyset, \mathtt{T} \rangle$. While $\mathscr{M}$ is our usual logical model, $\mathtt{T}$ denotes a set of *temporally extended operator schemata*: $o(\overline{v}) = \langle \top, \varepsilon, d_o^{\min}, d_o^{\max} \rangle$. The schema extensions $d_o^{\min}$ and $d_o^{\max}$ represent the minimal and maximal duration of the operator, that means, the amount of time the execution of that action is going to take. We may assume that all duration annotations in D_{TTAP} are specified as natural number symbols that refer to the same time base, for instance, minutes.[11]

Definition 3.21 (Consistency of Temporal Domain Models)**.** A temporal domain model $D_{\mathrm{TTAP}} = \langle \mathscr{M}, \emptyset, \mathtt{T} \rangle$ over a given language $\mathscr{L}$ is called *consistent* if and only if the following conditions hold:

1. The included domain model $\langle \mathscr{M}, \Delta, \mathtt{T} \rangle$ is consistent (see Def. 2.8).

[11] We refer the reader to the discussion about representing natural numbers in our framework in Sec. 2.8.3.

2. For every temporally extended operator schema in T with $o(\bar{v}) = \langle \top, \varepsilon, d_o^{\min}, d_o^{\max} \rangle$ the following inequation holds: $0 \leq d_o^{\min} \leq d_o^{\max} < \infty$

•

A TTAP plan extends the task assignment plan data structure as follows: $P = \langle TE, \prec, VC, TC \rangle$. It consists of a partially ordered set of parametrized tasks (given as usual by the sets *TE* and $\prec$), a set of variable constraints *VC*, and a set of *temporal constraints TC*. The constraints in *TC* represent the temporal information as a *simple temporal problem* (STP) [71], a representation formalism that allows to implement a point algebra efficiently [53] and that has been successfully deployed in the planning context (see discussion section). *TC* is a constraint system (z, d, c) with z being a set of variables that represent time points and $d : z \rightarrow \mathbb{R}_0^+$ a function for assigning sets of real numbers (including the symbol ∞ for representing an infinite amount of time) to each variable in z. The set of real numbers d_{x_i} that is assigned by d to a variable $x_i \in z$ is called the *domain* of that variable. The set c is a set of unary and binary constraints. A binary constraint represents the temporal distance between two time point variables x_i and x_j by an interval $[min, max]$ such that the equation $min \leq x_j - x_i \leq max$ holds. A unary temporal constraint specifies a time point x by an interval $[early, late]$, which means that $early \leq x \leq late$.

The temporal network specifies for each task expression $te \in TE$ over an operator schema $o(\bar{v})$ two time points that denote the beginning and the end of the task: *start* in $[0, \infty)$ and *end* in $[start + d_o^{\min}, start + d_o^{\max}]$. The symbols for these time points are the clock markers that will be used for synchronizing the temporal and the non-temporal representations. Since we need two of these markers per task expression, corresponding to the states that are involved in the task's transition, $\mathscr{M}$ does only need to contain a finite number of additional constants as distinguishable state marker symbols. For every two task expressions $te_i, te_j \in TE$ that are instances of operator schemata $o_i(\bar{v}_i)$ and $o_j(\bar{v}_j)$, respectively, and for which $te_i \prec te_j$ holds in the transitive closure of the ordering relation, consequently the temporal relation $end_{t_i}^{\max} \leq start_{t_j}$ holds in *TC*, that means, their temporal distance is given by the interval $[0, \infty)$. Conversely, for every two tasks for which $end_{t_i}^{\max} \leq start_{t_j}$ holds in *TC*, the transitive closure of the plan's ordering relation has to contain $te_i \prec te_j$.

Definition 3.22 (Consistency of Temporal Plans)**.** A temporal plan $P = \langle TE, \prec, VC, TC \rangle$ over a given language $\mathscr{L}$ and domain model D_{TTAP} is called *consistent* if and only if the following conditions hold:

1. The included partial plan $P = \langle TE, \prec, VC, CL \rangle$ with $CL = \emptyset$ is consistent over the included non-temporal domain model (see Def. 2.12).
2. *TC* is a consistent temporal constraint network, that means, there exists an assignment of domain values to temporal variables that satisfies all equations induced by *TC*.

•

Problems and Solutions

A TTAP problem for a given consistent temporal domain is specified by the following structure: $\pi_{\text{TTAP}} = \langle D_{\text{TTAP}}, s_\varepsilon, \top, P_{init} \rangle$. The initial plan P_{init} is thereby a (consistent) temporal plan, while the other components, the empty initial default-state and the trivially satisfiable goal state description, are defined analogously to task assignment planning problems. Please note that we do adopt the notion of a *null plan* regarding the problem definition (Def. 2.18). This representation encodes the initial and goal states as artificial actions in the plan, which makes them referable in temporal constraints. The corresponding artefacts in the temporal constraint set TC_{init} allows for specifying the starting time of the plan, the maximal makespan, or the deadline, etc.; another common usage of the temporal constraints is the explicit definition of clips and struts [100]. Please note that the regular temporal activity specifications, for example their duration, are given in the task schemata and not in the initial plan's constraint set, which is used for problem-specific temporal restrictions.

Given a temporal planning problem $\pi_{\text{TTAP}} = \langle D_{\text{TTAP}}, s_\varepsilon, \top, \langle TE_{init}, \prec_{init}, VC_{init}, TC_{init} \rangle \rangle$, the temporal plan $P = \langle TE, \prec, VC, TC \rangle$ is a solution to π_{TTAP} if and only if the following conditions hold:

1. The included task assignment plan $P' = \langle TE, \prec, VC \rangle$ is a solution to the included TAP problem $\pi'_{\text{TAP}} = \langle D_{\text{TAP}}, s_\varepsilon, \top, \langle TE_{init}, \prec_{init}, VC_{init} \rangle \rangle$. Domain model D_{TAP} is thereby the corresponding task assignment model with non-temporal versions of the operator schema definitions.
2. *TC* is consistent.
3. *TC* is a refinement of TC_{init}, that means, all solutions of *TC* are also solutions of TC_{init}.
4. *TC* is complete with respect to *TE* and $\prec$:
   a) For every $te = l : t(\bar{v}) \in TE$, *TC* contains a constraint $start_{te}$ in $[a,b]$ for some $a \geq 0$ and $b < \infty$ or $[a,\infty)$, respectively, as well as end_{te} in $[start_{te} + d_t^{\min}, start_{te} + d_t^{\max}]$.
   b) For every ordering relation tuple $te_i \prec te_j$ in the transitive closure of $\prec$, every solution to *TC* satisfies the in-equation $end_{o_i}^{\max} \leq start_{o_j}$.

* No temporal variable $start_o$ or end_o in *TC* is constrained by an infinite upper bound.

The solution criteria reflect the two reasoning layers: the first criterion grounds TAP plan generation in the underlying task assignment plan, the second and third guarantee that our plan refinement methodology is adopted, and the fourth twin criterion realizes the synchronization of the temporal and the non-temporal layer. Please note that the solution does not necessarily finalize the temporal assignments, that means, the solution may contain time intervals for the start of actions, etc. (cf. Def. 2.19 for *solution plan*). Analogously to action serialization and variable binding in task assignment planning, we intend by the last, optional criterion that all time intervals have to be "defined" in the sense that the start and end of every task are not unspecific intervals with infinite endpoints.

We are now ready to operationalize the solution criteria in the detection functions.

Detection Functions – $\mathfrak{Det}_{\text{TTAP}}$

The most prominent detection function that is specific for temporal planning is the following one; it detects occurrences of intervals for time variables that collapse and temporal distances that become negative. The flaw structure is analogous to the inconsistency detections on the variable and ordering constraint sets: we collect all variables in *TC* for which we can verify a constraint violation and map them onto the involved plan steps.

Definition 3.23 (Temporal Inconsistency)**.** For a given temporal plan $P = \langle TE, \prec, VC, TC \rangle$ and planning problem π, the flaw detection function $f^{det}_{\texttt{TempInconsistency}}$ indicates that the temporal constraints in *TC* have become inconsistent.

Given that *TC* is inconsistent, let $\{x_1, \ldots, x_n\}$ be those temporal variables in *TC* for which no value assignment can be found, in other words, for which a unary constraint implies an empty interval or which are involved in a temporal distance constraint that evaluates to an empty interval. Then $f^{det}_{\texttt{TempInconsistency}}(P, \pi)$ is the set of task expressions $\{te_1, \ldots, te_m\} \subseteq TE$ such that for any $1 \leq i \leq n$ there is a $1 \leq j \leq m$ with x_i being $start_{te_j}$ or end_{te_j}. If no such association can be found, the set includes te_{init} and te_{goal} instead. •

Since the flaw directly implements the beforehand given solution criterion of constraint consistency, it is trivially sound. Please also note that his flaw class is a critical one according to Def. 2.35.

The above flaw definition assumes that the deployed constraint reasoning mechanism may have some information about specific temporal constraints that are violated. Pleas note that this knowledge is in general an artefact of the reasoning system, because after the first occurrence of a range collapse, the propagation algorithms typically stops as there is nothing to deduce from an inconsistent constraint set. For reasoning about the STP, we currently deploy a dynamic AC-3 based constraint reasoning approach [47] that maintains the temporal constraint network arc-B-consistent (cf. [168]). Therefore, we mainly use the flaws' plan step references as an explanatory suggestion in cases where specific temporal modifications evoke the inconsistency locally, for example, in an optimization effort. If the algorithm has no such insight into the inconsistency or if an affected variable cannot be associated to a plan step for some other reason (the hierarchical extension, for example, will introduce such auxiliary variables) then the detection function generalizes that the whole temporal chain became invalid from the initial state up to the goal state.

Exceeding a maximum makespan or passing a deadline is also covered by the above detection function: the corresponding interval for $start_{te_{goal}}$ will induce a network inconsistency if the starting time cannot be realized.

Definition 3.24 (Open Time Variable Binding)**.** For a given temporal planning problem π and consistent temporal plan $P = \langle TE, \prec, VC, TC \rangle$, the flaw detection function $f^{det}_{\texttt{OpenTmpBind}}$ indicates that the temporal constraints in TC allow for a more precise scheduling of the tasks.

Let $\{x_1, \ldots, x_n\}$ be those temporal variables in TC for which the constraints support an infinite value, that means, for every x_i with $1 \leq i \leq n$ there is a solution to TC that includes the assignment $x_i = \infty$. Then $\{te\} \in f^{det}_{\texttt{OpenTmpBind}}(P, \pi)$ with $te \in TE$ being the task expression for which $x_i \in \{start_{te_j}, end_{te_j}\}$. •

This definition covers the optional solution criterion and is only included in the configuration if the application requires finite time bounds in any circumstance; unbound time variables can only occur in problems with unspecified deadline, because otherwise an upper bound would be propagated. We would also like to point out that the open time variable binding flaw class is a prototypical example for understanding a temporal quality as a plan deficiency. As we will address in the discussion of this chapter, the notion of opportunity – in this case the opportunity to reduce the slack time or the makespan – will be introduced in future temporal planning configurations via detection functions of that kind.

For reasoning about the consistency of partial plans, this configuration inherits the detection function set $\mathfrak{Det}_{\text{TAP}} = \{f^{det}_{\texttt{OrdIncons}}, f^{det}_{\texttt{VarIncons}}, f^{det}_{\texttt{OpenVarBind}}, f^{det}_{\texttt{UnordTask}}\}$ from the task assignment planning system configuration.

Modification Generating Functions – $\mathfrak{Mod}_{\text{TTAP}}$

The refinement options for temporal planning are first of all rooted in the modification generating functions from task assignment planning. In addition, we provide in a first step of "active" temporal planning (see Sec. 3.5.5) the configuration with an explorative way of narrowing time intervals for flawed plan steps.

Definition 3.25 (Add Temporal Constraint)**.** For a given temporal plan $P = \langle TE, \prec, VC, TC \rangle$, flaw $\texttt{f}$, and temporal domain model D, the modification generating function $f^{mod}_{\texttt{AddTempConstr}}$ proposes to include an appropriate temporal constraint as follows: Let $te \in TE \cap comp(\texttt{f})$ be a flawed plan step and let all assignments to $start_{te}$ that are consistent with the constraints in TC lie in the interval $[a,b]$. Let us furthermore assume that the generating function has defined an arbitrary real number $0 < \delta < \infty$. The function then returns

$$f^{mod}_{\texttt{AddTempConstr}}(P, \texttt{f}, D) \supseteq \begin{cases} \{\langle a \leq start_{te} \leq a+\delta, \emptyset \rangle, \langle a+\delta \leq start_{te} < \infty, \emptyset \rangle\} & \text{if } b = \infty \\ \{\langle a \leq start_{te} \leq \frac{b-a}{2}, \emptyset \rangle, \langle \frac{b-a}{2} \leq start_{te} \leq b, \emptyset \rangle\} & \text{else} \end{cases}$$

If the start time point is assigned a single value, that means $start_{te} \in [a,a]$, $f^{mod}_{\texttt{AddTempConstr}}$ returns no plan modification for $\texttt{f}$. •

The strategy that is followed by this modification generator is apparently that of systematically reducing the assignment range for the starting time point of plan steps (given that the configuration provides the means for propagating this decision, start and end point are constrained by the task duration). If a task with definite time bounds is flawed, the function induces a binary decision in terms of probing the lower and upper half of the interval, respectively. The second case, a missing upper bound, leads to a guess (in this version of the configuration a completely uninformed one) for a suitable split point. The rationale for publishing an unbound alternative modification is that the completeness of refinement options has to be maintained: if the remaining plan generation process does not support the corresponding operator to start within δ time, the configuration still has refinement options available.

The temporal constraints adding function is a sound plan modification generator, because the flaw is addressed implicitly and the component balance is positive. Furthermore, it is also obviously a refinement, because of the convex relation that is imposed by the STP constraint intervals. Please also note that we may assume the modification application function (see app operator in Def. 2.20) to treat the temporal constraints

transparently. Concerning future developments, in correspondence to our comments on the temporal flaw detectors, the modification generating functions will deploy more sophisticated temporal calculations and informed heuristics in order to balance the time slack, reduce the makespan, and the like. In particular, the proper choice of the δ parameter, which cuts finite sub-intervals from unrestricted ones, has a large implications on the efficiency of the procedure.

Triggering Function $\alpha_{\mathbf{TTAP}}$

The triggering function for temporal planning employs α_{TAP} of the configuration of task assignment planning for dealing with the task orderings and parameter assignments. The newly introduced generator for adding temporal constraints is solely triggered by the temporal variable binding deficiency.

$$\alpha_{\mathrm{TTAP}}(\mathbb{F}_x) = \begin{cases} \mathbb{M}_{\texttt{AddTempConstr}} & \text{for } x = \texttt{OpenTmpBind} \\ \emptyset & \text{for } x = \texttt{TempInconsistency} \\ \alpha_{\mathrm{TAP}}(\mathbb{F}_x) & \text{otherwise} \end{cases}$$

A temporal inconsistency cannot be resolved by a plan modification and is therefore not addressed by any modification generating function. It is a critical flaw and induces a plan discard (cf. Def. 2.35).

Inference Functions – $\mathfrak{Inf}_{\mathbf{TTAP}}$

A system configuration implementation places similar requirements on temporal constraints as it does on the variable constraints: one constraint reasoning subsystem should make all deduced relationships explicitly available in the constraint sets. Since this section presents temporal planning from a conceptual perspective, we consider such an inference function as optional and omit it for this presentation. We note, however, that it would be a sound inference function in any case, because it returns proper plan modifications with a positive balance of elementary additions.

Furthermore, inference functions are an integral part of the temporal planning system configuration, because they are the semantic connection between the formal framework core and the temporal reasoning layer. The following inference functions realize this by deducing temporal constraints that are implied by the ordering relation.

Definition 3.26 (Temporal Variable Introduction)**.** For a given temporal plan $P = \langle TE, \prec, VC, TC \rangle$ and planning problem π, the inference function $f^{inf}_{\texttt{IntroTempVar}}$ adds a unary temporal constraint to plan P for each plan step that is apparently not yet represented in TC.

$\langle \{0 \leq start_{te} < \infty,\ start_{te} + d_t^{\min} \leq end_{te} \leq start_{te} + d_t^{\max}\}, \emptyset \rangle \in f^{inf}_{\texttt{IntroTempVar}}(P, \pi)$ if and only if $te \in TE$ is an instance of task schema t and $start_{te}$ and end_{te} do not occur in TC. •

The rationale for the first inference is to introduce newly added plan steps to the temporal reasoning system. It allows to deploy time-unaware configuration components to safely manage the TE set.

Definition 3.27 (Temporal Distance Introduction)**.** For a given temporal plan $P = \langle TE, \prec, VC, TC \rangle$ and planning problem π, the inference function $f^{inf}_{\texttt{IntroTempDist}}$ adds a binary temporal constraint to plan P if the ordering constraints contain a precedence that is not implied by TC.

Given two task expressions $te_i = l_i : t_i(\bar{v}_i)$ and $te_j = l_j : t_j(\bar{v}_j) \in TE$ with $te_i \prec te_j$ in the transitive closure of $\prec$, then $\langle \{end_{t_i}^{\max} \leq start_{t_j}\}, \emptyset \rangle \in f^{inf}_{\texttt{IntroTempDist}}(P, \pi)$ if and only if $end_{t_i}^{\max} \leq start_{t_j}$ is not satisfied by TC. •

The introduction of distance constraints corresponds to the previous inference such that components may focus on the qualitative abstract temporal model and manipulate the ordering constraints accordingly, while at the same time these manipulations are synchronized with the quantitative temporal representation. The following inference provides the inverse functionality, that means, transfers quantitative results to the qualitative level.

Definition 3.28 (Ordering Constraint Introduction)**.** For a given temporal plan $P = \langle TE, \prec, VC, TC \rangle$ and planning problem π, the inference function $f^{inf}_{\texttt{IntroOrdering}}$ adds an ordering constraint to plan P if the temporal constraints support a precedence that is not implied by $\prec$.

Given two task expressions $te_i = l_i : t_i(\overline{v}_i)$ and $te_j = l_j : t_j(\overline{v}_j) \in TE$ for which TC is consistent with the inequation $end^{\max}_{t_i} \leq start_{t_j}$, then $\langle \{te_i \prec te_j\}, \emptyset \rangle \in f^{inf}_{\texttt{IntroOrdering}}(P, \pi)$ if and only if $te_i \prec te_j \notin \prec^*$, the transitive closure of $\prec$. •

When we say that the inference mechanism "synchronizes" the qualitative and quantitative temporal view, we are taking into account that all inference functions are called repeatedly until the inferential closure is reached. That means that after the first phase of the generic refinement planning algorithm (Alg. 2.2, lines 4-8) *all* implicit interdependencies between $\prec$ and TC are made explicit (cf. management modules in O-PLAN that are dedicated to the management of temporal information and feedback into corresponding plan data structures [76]).

All three inference functions are trivially sound, because they merely add constraints to the respective sets, thereby conservatively reducing the solution candidates.

Other inferences may realize the notion of *precision* by co-designating temporal variables for which the distance drops below a certain threshold $\varepsilon > 0$ that characterizes the minimal temporal quantity that is observable in the execution environment. As an example, consider our hardware installation project above for which it would not make sense to distinguish durations that are shorter than five minutes.

Summary of the TTAP Configuration

With the above function definitions, the temporal planning configuration specified as follows:

$$\begin{aligned}\mathfrak{C}_{\text{TTAP}} = \langle \{ & \overbrace{f^{det}_{\texttt{OrdIncons}}, f^{det}_{\texttt{VarIncons}}, f^{det}_{\texttt{OpenVarBind}}, f^{det}_{\texttt{UnordTask}}}^{\mathfrak{Det}_{\text{TAP}}}, f^{det}_{\texttt{TempInconsistency}}, f^{det}_{\texttt{OpenTmpBind}} \} \\ & \overbrace{\{f^{mod}_{\texttt{AddOrdConstr}}, f^{mod}_{\texttt{AddVarConstr}}}^{\mathfrak{Mod}_{\text{TAP}}}, f^{mod}_{\texttt{AddTempConstr}}\}, \\ & \{f^{inf}_{\texttt{IntroTempVar}}, f^{inf}_{\texttt{IntroTempDist}}, f^{inf}_{\texttt{IntroOrdering}}\}, \\ & \mathfrak{Str} \rangle\end{aligned}$$

Theorem 3.9 (Properness of $\mathfrak{C}_{\text{TTAP}}$)**.** $\mathfrak{C}_{\text{TTAP}}$ *is a proper system configuration in the sense of Def. 3.2.*

Proof. Given that $\mathfrak{C}_{\text{TAP}}$ is proper according to Theorem 3.1, the following arguments hold.

$\mathfrak{Det}_{\text{TTAP}}$ is a complete set of sound detection functions, because the included set $\mathfrak{Det}_{\text{TAP}}$ is proven to be complete and the additional detection functions directly correspond to the solution criteria concerning the temporal meta-constraint system.

$\mathfrak{Mod}_{\text{TTAP}}$ is a semi-complete set of sound modification generating functions, because the addition of temporal constraints is a sound generator that satisfies any proposed flaw if the corresponding refinement is applicable and because it extends $\mathfrak{Mod}_{\text{TAP}}$, which is semi-complete as well.

$\mathfrak{Inf}_{\text{TTAP}}$ has been shown above to be a set of sound inference functions.

It can also be easily seen that $\mathfrak{Mod}_{\text{TTAP}}$ corresponds to $\mathfrak{Det}_{\text{TTAP}}$: every modification generator is assigned a detection function in α_{TTAP}. □

Theorem 3.10 ($\mathfrak{C}_{\text{TTAP}}$ is Modification-Complete)**.** $\mathfrak{C}_{\text{TTAP}}$ *is modification-complete according to Def. 3.3.*

Proof. Given that the included $\mathfrak{C}_{\text{TAP}}$ configuration is modification complete (Theorem 3.2) and that the only un-processed flaw is a critical one (temporal inconsistency), the proposition holds. □

3.3.3 Resource Planning – $\mathfrak{C}_{\mathbf{RTAP}}$

The system configuration for resource planning is an extension of task assignment planning that allows for reasoning about objects or substances that induce constraints on the actions that use them because of their cost or available quantity; we call these entities *resources*. As we have argued in the introductory chapter (Sec. 1.2), the notion of consumable substances like energy or money as well as the concept of anonymous objects like tools or machines is an integral part of real-world planning applications. Reasoning about resources has a long tradition and hence a large variety of representation and reasoning mechanisms is available today. This framework focuses on creating a unifying basis for resource reasoning and causal inference and therefore formulates a *constructive* approach to profile management of *discrete* resources.

We build resource reasoning directly on the TAP configuration in order to design it as flexible as possible. In particular, our intention is to realize the resource treatment in a way that is analogous to the integration of temporal information in the previous configuration: a layer on top of the causal plan that uses it as an abstract view on the resource plan. The production and consumption of resources are therefore understood as processes that cannot be analyzed below the modelling horizon as it is defined by the primitive operators. Our arguments are basically the same as for temporal planning before (Sec. 3.3.2): In situations of parallel action execution, the precise interleavement of elementary operations is intentionally not deducible and therefore two actions that operate on the same resource necessarily have to be ordered sequentially (cf. Fig. 3.13). We also want to keep the modelling of access primitives as simple as possible: Instead of providing sets of directives for explicitly consuming, producing, and allocating resources, we will employ a natural representation that uses term updates and our sort system.

Our rationale for defining resource planning as a more "passive" configuration is to provide a basic configuration that is able to verify the resource demands of a plan, to indicate inconsistent allocations accordingly, and finally to complete any partial assignment in a systematic way. In this view, the resource reasoning subsystem is keeping track of the worst case estimate, that is to say, it is profiling the upper bound regarding resource utilization. It is thereby guaranteeing that any solution is feasible with respect to the given resource availability. Please note that the actual interpretation of allocation information, its propagation and computation techniques, is not in the focus of this section. Our system design allows for the deployment of standard reasoning techniques and algorithms and we will give reference to them where applicable. $\mathfrak{C}_{\text{RTAP}}$ is thereby the foundation for future developments that will implement resource management more actively in the canon of configurations, and in particular in a more optimization-oriented way; the discussion section details some of them.

Domain Model Specifics

Let us begin with the basic definitions of the resources per se. We support two kinds of resources, namely symbolic and numerical ones. The latter correspond to the usual notion of *quantity*, for example, the numerical resources of available construction material in tons, fuel in gallons, etc. A numerical resource is typically associated with the concept of a *level*, the currently available quantity, and the *capacity*, its maximal level. An example is the electric energy provided by a battery: its capacity is the maximal battery charge and its level drops according to the electric power consumption. We may assume that the lower bound for every resource level is 0, that means, the resource becomes unavailable, negative levels are therefore not permitted. For practical considerations we furthermore distinguish *consumable* and *non-consumable* numerical resources, with which we refer to resources that are consumed or "borrowed" by the processes that use them, respectively. A numerical resource is accessed via querying and updating a specific flexible term; increasing and decreasing its value corresponds to consuming and producing quantities, respectively. Since the formal framework does not perform numeric calculations but rather deals with symbolic (in-) equalities, the restriction to finite symbol sets for the representation of a finite number of "tokens" in the continuous space of real numbers is feasible.

The difference between symbolic resources and regular planning objects is a subtle one: From the point of view of the formal framework, both concepts coincide, but from the conceptual knowledge representation perspective, the identity of a resource entity is explicitly not of interest. This allows for efficient reasoning

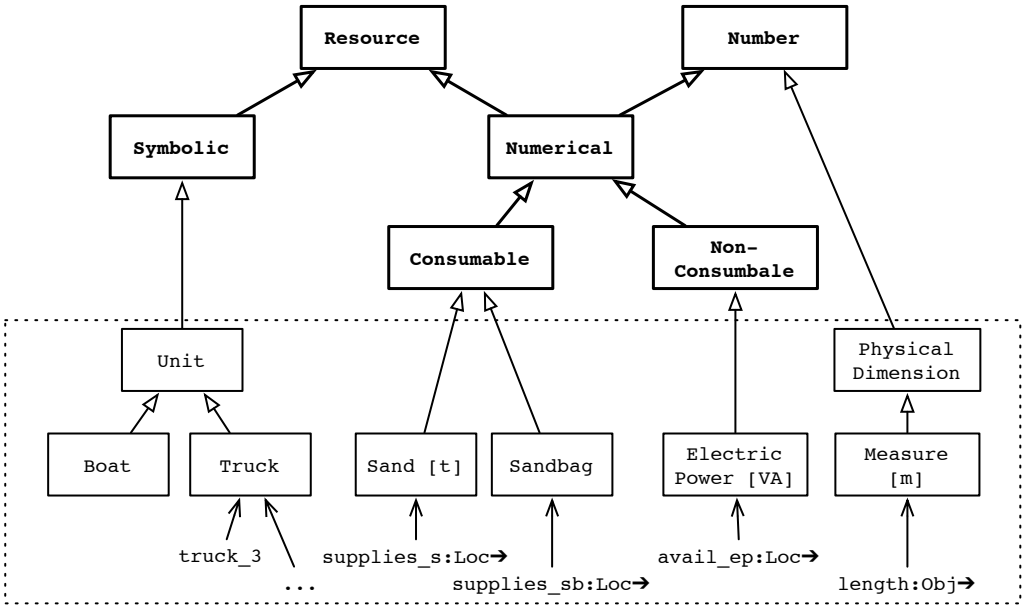


Figure 3.14: The sort hierarchy for resource planning: The relationship between symbolic, consumable numerical, and non-consumable numerical resources. The dotted area represents an example for resource definitions in the disaster relief mission domain.

mechanisms that analyze allocation profiles and identify bottlenecks, potential and definite over-allocations, etc., rather than dealing with equations and in-equations in constraint sets. The allocation of symbolic resources is represented implicitly by task and operator parameters, which signal the usage of objects of the given sort. Note that we cannot "create" or "destroy" objects, therefore symbolic resources are necessarily non-consumable.

The logical language $\mathscr{L} = \langle \mathscr{Z}, \leq, \mathscr{R}_r, \mathscr{R}_f, \mathscr{F}_r, \mathscr{F}_f, \mathscr{V}, \mathscr{T}_p, \mathscr{T}_c, \mathscr{E} \rangle$ for a resource planning configuration $\mathfrak{C}_{\text{RTAP}}$ reflects the resource conceptualization in its sort system and function definitions. $\mathscr{Z}$ contains six designated sort symbols that are in the following sub-sort relationships (Fig. 3.14): $\texttt{Symbolic} \leq \texttt{Resource}$, $\texttt{Numerical} \leq \texttt{Resource}$, $\texttt{Numerical} \leq \texttt{Number}$, $\texttt{Consumable} \leq \texttt{Numerical}$, and $\texttt{NonConsumable} \leq \texttt{Numerical}$. We may assume that $\mathscr{C}_r$ provides a suitable set of natural or real number constants for populating the numerical sorts `Number` and `Numerical`. The consumable numerical resources are represented by the flexible functions that are of the corresponding sort `Consumable`, that means $\mathscr{F}_{consumable} = \{f | f \in \mathscr{F}_f \cap \mathscr{F}_{Z_1 \ldots Z_n, Z'}$ with $Z' \leq \texttt{Consumable}\}$. The figure shows a function $\texttt{supplies\_s} : \texttt{Loc} \rightarrow \texttt{Sand}$ which is intended to model the sand supplies that are available at a given location. `Sand` is an example for a consumable numerical resource, which is a flexible function that describes the amount of sand in tons. $\texttt{supplies\_sb} : \texttt{Loc} \rightarrow \texttt{Sandbag}$ is the corresponding function for modelling sandbags that can be deployed for dyke fortifications. The non-consumable numerical resources $\mathscr{F}_{nonConsumable}$ are defined analogously via the rigid function symbols. The presented example shows the function $\texttt{avail\_ep} : \texttt{Loc} \rightarrow \texttt{ElectricPower}$, which associates an electric power level in volt-ampere with a given location. The model uses a non-consumable resource, because the electric energy is on a certain level of abstraction an energy source that is shared by consumer devices, which do however not drain the generators like they would do batteries. The implied constraint is that at every moment in time a certain amount of electric power is available that must not be exceeded.

Although the distinction in rigid and flexible functions subsumes in some sense the resource definitions in the sort hierarchy, we prefer this rather verbose modelling style for its clarity. In this way, all resources are explicitly declared in the concept taxonomy and at the same time defined in a semantically safe way. The domain model furthermore includes rigid functions for numerical calculations like addition, subtraction, and multiplication, as well as some basic functions like maximum and minimum. Fig. 3.14 furthermore shows numerical data types that are not regarded as resources, in the example this is a rigid function that defines

the length of objects in meters.

Please note that we model inequations in operator schemata in a not very obvious way, namely $x \equiv \max(x,y)$ instead of $x \geq y$, and the inequality relation symbols are consequently not included in the resource planning language. The rationale is simply to maintain a view on resource calculations that directly matches the causal view of the formal framework: an equation query is naturally causally linked to an update and can therefore be directly interpreted as the symbolic outcome of the update (no numerical computations), which is recorded by equations in the variable constraint set, if appropriate. It is the duty of a (possibly external) term reasoning facility to verify under which assumptions the term interpretations are consistent; if they are not, it has to announce a flaw. Introducing inequations however implied that causal reasoning had to deal with threat scenarios involving $\geq$ and $\leq$ statements and, consequently, the corresponding numerical comparisons had to be integrated into the variable constraints. As a consequence of the equality usage, term queries and corresponding updates can be examined in a causality-aware configuration such that resource production and consumption activities are linked on the causal level as well. As a result, the planning system has a coherent view on some of the scheduling aspects of the problem.

Given a resource-extended logical language $\mathscr{L}$, the domain model representation for the RTAP system configuration is based on a task assignment planning model, that means, $D_{\text{RTAP}} = \langle \mathscr{M}, \emptyset, \mathtt{T} \rangle$. The operator schema definitions in `T` basically have the structure that we introduced in the formal framework section (Def. 2.4), that means $o(\bar{v}) = \langle \text{prec}(o(\bar{v})), \text{eff}(o(\bar{v})) \rangle$. The resource planning specifics are the following:

- $\bar{v} = v_1, \ldots, v_n$ is the list of operator parameters. Any parameter v_i with $1 \leq i \leq n$ of sort Z_i with $Z_i \leq \mathtt{Symbolic}$ denotes an allocation of resource type Z_i, that means, an exclusive employment of a domain object $c \in \mathbb{D}_{Z_i}$.
- $\text{prec}(o(\bar{v}))$ contains queries to numerical resource levels. Every occurrence of a non-consumable term is interpreted as an allocation of the term's sort with the quantity given by the term's value in the state in which operator o is to be executed.
- $\text{eff}(o(\bar{v})) = e_1 \ldots e_m$ is a non-empty, finite sequence of elementary term-update operations. It contains monotonic manipulations of consumable numerical resources, that means, term updates that are either necessarily increasing or decreasing the value for all operator instances.

The last structural restriction is not essential for this configuration, it is however a necessary simplification for our prototypical resource reasoning approach to efficiently deduce resource bounds from the operator specification. It allows to classify operators into producing and consuming activities for a given resource. Please note that according to operator consistency requirements (Def. 2.6) the effects are restricted to one update per term and that the outcome of the elementary operations does not depend on their ordering. The latter implies that operators cannot define interdependent resource manipulation effects.

Regarding the temporal dimension, as it has been stated above, no allocation or update can be located at a specific point in time that is below the operator resolution. That means, that the complete state transition has to be protected against concurrent resource access and, consequently, two "parallel" operators have to share their resources during their complete execution interval. The temporal model used in this section is the qualitative ordering relation, the consequences will however propagate into the temporal extensions of the $\mathfrak{C}_{\text{RTAP}}$ system configuration.

We show an example for a resource planning domain model in the following operator definitions of the disaster relief mission domain. Let us assume that the corresponding language introduced the variables u, l, and d of sort `Unit`, `Loc`, and `Dyke`, respectively.

$$\begin{aligned}
\mathtt{MobilizeUnite}(u) &= \langle \top, \emptyset \rangle \\
\mathtt{ProvideElectricity}(u,l) &= \langle \top, \emptyset \rangle \\
\mathtt{InstallIllumination}(u,l) &= \langle \equiv (\mathtt{avail_ep}(l), \max(\mathtt{avail_ep}(l), 20)), \emptyset \rangle \\
\mathtt{FillSandbags}(u,l,n) &= \langle \top, := \mathtt{supplies_s}(\mathtt{supplies_s}(l) - n) \\
&\qquad := \mathtt{supplies_sb}(\mathtt{supplies_sb} + (n \cdot 60)) \rangle \\
\mathtt{DistributeSandbags}(u,d) &= \langle \top, := \mathtt{supplies_sb}(\mathtt{supplies_sb}(d) - (\mathtt{length}(d) \cdot 100)) \rangle
\end{aligned}$$

The intended meaning of these operator definitions is that all activities require the presence of a relief organization unit, which implies the allocation of the corresponding vehicle and personnel. `ProvideElectricity` is performed by a unit that is equipped with a 20kVA generator; the electric power is thereby not produced in order to be consumed (a non-consumable resource) but rather relocated in order to be shared by `InstallIllumination` actions. The latter require 20kVA for providing daylight-like conditions for a working area; the implicitly allocated unit operates the illumination system. Concerning the dyke fortifications, we have to distribute about 100 sandbags per meter length of the damaged wall segment. The corresponding supplies are made by filling sand into fabric bags; the model specifies that it takes about 1t of sand for producing 60 sandbags.

We want to emphasize two particularities of our modelling approach: First, allocating a non-consumable resource is usually represented by a combination of a resource consumption and a subsequent production. It is however impossible to access the intermediate (sub-atomic) states below the operator level and therefore the allocation cannot be represented in our framework in this way. Our solution is shown in the example operator `InstallIllumination`, that means, the term query implements a "lock" on the required electric energy. The allocation is of course only interpretable by the reasoning system of the resource planning configuration, because a purely causal view on the plan will only perceive a requirement of 20kVA that does not imply any state change. The resource reasoning system can however deduce from the sort hierarchy that electric power is a non-consumable resource and therefore every identified term evaluation has to be treated as concurrently accessed quantities, analogously to symbolic resources.

The second unique aspect of our approach is the use of the sort hierarchy for resource reasoning. Allocating a certain quantity of a given resource implicitly allocates the same quantity of the super-sort resource, for example, deploying a truck implies deploying a unit (cf. Fig. 3.14). The more interesting inference, however, is to deduce *potential allocations* of concrete resources from the allocation of abstract ones. If the `FillSandbags` activity uses one unit, it potentially uses one truck or one boat, unless variable typing constraints rule out one option. We will see this mechanism in an example below.

Given the above components for the specification of resources and corresponding manipulation methods, consistency of such a resource-aware model has to be considered carefully according to the following definition:

Definition 3.29 (Consistency of Resource Planning Domain Models)**.** A resource planning domain model $D_{\mathrm{RTAP}} = \langle \mathscr{M}, \emptyset, \mathtt{T} \rangle$ over a given (resource) language $\mathscr{L}$ is called *consistent* if and only if the following conditions hold:

1. The included domain model $\langle \mathscr{M}, \Delta, \mathtt{T} \rangle$ is consistent (see Def. 2.8).
2. Neither $\mathtt{Consumable} \leq \mathtt{NonConsumable}$ nor $\mathtt{Consumable} \leq \mathtt{NonConsumable}$ and neither $\mathtt{Symbolic} \leq \mathtt{Numerical}$ nor $\mathtt{Consumable} \leq \mathtt{Symbolic}$ hold in the transitive closure of the sub-sort relation $\leq$.
3. All operators that access consumable resources have satisfiable preconditions and effects such that productions do not necessarily over-produce the resource and consumptions do not over-consume, respectively.

•

For representing the dynamics of consumable resources in a plan, we deploy a constraint-based approach that parallels our temporal constraint handling and therefore we discuss it only briefly in this section (see p. 112). A resource plan is the structure $P = \langle TE, \prec, VC, RC \rangle$, which extends the task assignment plan specification by a set RC of resource constraints. The underlying constraint system (z, d, c) consists of variables z that represent the resource levels in the states before and after the operator transitions, namely $in^{\mathtt{r}}_{te}$ for the input level of resource `r` at the plan step te and $out^{\mathtt{r}}_{te}$ for the corresponding output level. The set z also contains a third type of variables, the allocation levels for non-consumable resources $alloc^{\mathtt{r}}_{te}$. The domain assignment function for the variables is given by $d : z \to \mathbb{R}^{+}_{0}$, which also includes the symbol ∞ for denoting an unlimited amount. The set c contains unary and binary constraints that describe the possible intervals for the value assignments to the level variables. For each resource `r` of sort Z, $Z \leq \mathtt{Consumable}$, that occurs in a task expression te in TE, c contains two constraints such that the equations $i_{\min} \leq in^{\mathtt{r}}_{te} \leq i_{\max}$ and $o_{\min} \leq out^{\mathtt{r}}_{te} \leq o_{\max}$ hold. Since the set of basic mathematical functions is known in advance and the resource manipulating terms are restricted to a monotonic composition, it can be directly deduced from the

operator schema's effect structure whether it consumes or produces a resource. If te utilizes $\mathtt{r}$ at a rate of $n^{\mathtt{r}}_{te} \in [\min^{\mathtt{r}}_{te}, \max^{\mathtt{r}}_{te}]$ then the binary constraints relate input and output variables as follows: $\min^{\mathtt{r}}_{te} \leq out^{\mathtt{r}}_{te} - in^{\mathtt{r}}_{te} \leq \max^{\mathtt{r}}_{te}$. Although the RTAP configuration does not directly support rate ranges in its domain models, that means, $\min^{\mathtt{r}}_{te} = \max^{\mathtt{r}}_{te}$ for all operators, we still use these constructs in the resource constraint set for later configurations. We will see in the hierarchical extension of this configuration that ranges of consumption and production may occur in effect abstraction. We have to note, however, that a complete inference on these presented resource constraints in the sense of optimization is significantly more complex than in the case of temporal information because the propagation has to take into account the possibility of adding resource production steps. For an efficient solution to this problem, we refer to a technique called *reservoir balance propagation*, which has been developed in [162] and which we will adopt for future resource planning configurations. Our resource representation corresponds to the therein given Resource Temporal Networks (RTN), and in particular we can include our temporal model that we have introduced before (an STN is the time model of an RTN). However, for the time being, our resource constraints are only used for documenting the resource commitments of plan steps and are thereby the basis for deducing upper and lower bounds of resource utilization in the fashion of optimistic and pessimistic *resource profiles* that will be described in the following section.

A resource profile for a resource $\mathtt{r}$ in a resource plan $P = \langle TE, \prec, VC, RC \rangle$ is given by a set $RC' = RC$ of resource constraints and a set $\prec' = \prec$ of ordering constraints. The *optimistic profile* for $\mathtt{r}$ over P is obtained from the neutral profile as follows: Let $te_1 \ldots te_n$ be a sequence of all task expressions in TE that utilize $\mathtt{r}$ such that for increasing sequence position the associated utilization maxima decrease, that means, for $1 \leq i < j \leq n$ the equation $\max^{\mathtt{r}}_{te_i} \geq \max^{\mathtt{r}}_{te_j}$ holds. Starting from $i = 1$, we add $te_i \prec te_j$ to $\prec'$ for every $1 \leq i < j \leq n$ if $te_j \prec te_i$ is not in $\prec$. The result is an ordering relation $\prec'$ in which the most producing plan steps are moved to the front, if possible. We note that it is actually only relevant to prefer producing to consuming steps. After that, we build a second task sequence $te'_1 \ldots te'_n$ that is consistent with $\prec'$ and that contains all task expressions in TE that utilize $\mathtt{r}$. For $1 \leq i < n$ we add to RC' the resource constraint $out^{r}_{te'_i} = in^{r}_{te'_{i+1}}$ and for every $1 \leq k \leq n$ $n^{\mathtt{r}}_{te'_k} = \max^{\mathtt{r}}_{te_{k'}}$. Now we are able to propagate the production and consumption of the resource along the respective plan steps, which produces the upper and lower bounds of the total resource manipulation.

The *pessimistic profile* is calculated in exactly the same way, except that the first sequence of tasks is ordered according to an increasing minimal utilization, that means for $1 \leq i < j \leq n$ we have $\min^{\mathtt{r}}_{te_i} \leq \min^{\mathtt{r}}_{te_j}$, and with the second sequence we associate for $1 \leq k \leq n$ the minimal rates $n^{\mathtt{r}}_{te'_k} = \min^{\mathtt{r}}_{te_{k'}}$. In this profile, the most consuming tasks are put in front, followed by the production steps.

Both profiles over-consume or over-produce a resource if the updated constraint sets RC' become inconsistent. We may however want to be more specific in order to identify the first point of failure for bottleneck analysis. To this end, we can iterate the propagation procedure for the profiles along the production and consumption episodes. The plan necessarily over-consumes the resource if the first inconsistency occurs in a consumption episode of the optimistic profile and it necessarily over-produces it, if the first failure occurs in a production episode of the pessimistic profile. Please take into account that a plan may both over-consume and -produce the same resource multiple times.

The non-consumable and symbolic resources are managed in a very simple way: Symbolic resources of sort $Z_{\mathtt{r}}$ are implicitly represented via the variable constraints VC and the ordering relation $\prec$. We deduce an optimistic lower bound that is equal to the number of necessarily distinct instances of the given sort and a pessimistic upper bound that is given by the total number of objects in the domain of $Z_{\mathtt{r}}$ minus the co-designated identities in VC. Regarding the numerical non-consumable resources, we introduce variables $alloc^{\mathtt{r}}_{te}$ in RC with which we model profiles that allocate at rates that depend on the range of other resources in RC. We then infer from the partial ordering relation all possible time slices and with that, we can determine the upper and lower bounds for parallel allocations. These reasoning tasks are considerably easier than those for consumable resources and rely on standard algorithms, for instance, the calculation of minimal critical sets [52, 164].

Concerning the consistency of resource plans, the RTAP configuration completely relies on the included TAP plan.

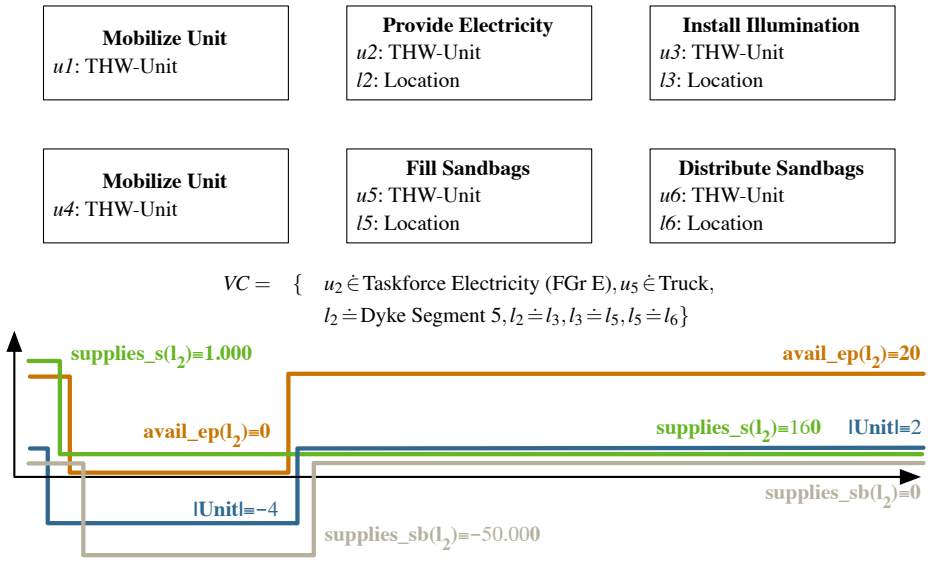


Figure 3.15: An initial plan of a resource planning problem in the disaster relief mission domain.

Definition 3.30 (Consistency of Resource Plans)**.** A resource plan $P = \langle TE, \prec, VC, RC \rangle$ over a given language $\mathscr{L}$ and domain model D_{RTAP} is called *consistent* if and only if the following conditions hold:

1. The included partial plan $P = \langle TE, \prec, VC, CL \rangle$ with $CL = \emptyset$ is consistent over D_{RTAP} (see Def. 2.12).
2. RC is consistent, that means, there exists an assignment of domain values to resource variables that satisfies all equations induced by RC.

•

Regarding the consistency condition for the resource constraint set, please recall that we do not automatically introduce binary constraints between consecutive plan steps because later extensions of this configuration will be able to add production steps. This non-monotonic constraint manipulations would make an over-consuming and hence inconsistent resource demand feasible.

Problems and Solutions

RTAP problems are defined as task assignment planning problems that have in addition an initial state for specifying the available resource quantities and a goal state description for specifying designated resource levels: $\pi_{\text{RTAP}} = \langle D_{\text{RTAP}}, s_{init}, \mathfrak{s}_{goal}, \langle TE_{init}, \prec_{init}, VC_{init}, RC_{init} \rangle \rangle$. In terms of the *null plan*, we employ a set-up task for s_{init} that updates all resource terms, including the non-consumable resources (but is otherwise consistent, cf. discussion in Sec. 2.6.1), and a goal state task with the appropriate term queries in its precondition according to $\mathfrak{s}_{goal}$. In addition to the regular resource specifications in the action schemata, the resource constraints that are given in the initial resource plan (RC_{init}) may impose further restrictions that are problem-specific. An example for such a restriction is a balanced production of two activities, in which an equation between the output levels guarantees that both produce the same amount of a given substance.

Fig. 3.15 shows an example specification for a resource planning problem, which we adapted from the task assignment example (Fig. 3.4). The corresponding domain model has been sketched above. The six activities are not ordered, all parameters for the relief organization units are unbound, and the operation area is set to a specific damaged segment of a dyke. Initially, there are 1.000t of sand available at the dyke, no sandbags, and a mobile 20kVA generator; the relief organization has deployed two units in the area. The lower part of the figure depicts the pessimistic usage profiles for the four resources over an imaginary time line. Recall

that this worst-case scenario assumes for every resource that any consumption occurs as early and extensive as possible while any production occurs as late and sparse possible [77]. The resource reasoning system in our prototypical configuration implementation employs this profile calculation in order to identify *potential* resource over-consumptions and -allocations. The blue `Unit` profile, for example, assumes that all six units to which the plan refers are allocated at the beginning of plan execution. In this case, the plan also represents courses of action that require six different units at the same time; this resource demand exceeds the available quantity by four. The profile for the other sorts is not depicted in the figure, but we note that via the sort hierarchy, the `Boat` resource becomes potentially over-allocated as well, because four units may turn out to be co-typed into this sort for which no objects are defined. Regarding the sandbag supplies, the distribution action consumes 50.000 sandbags for stabilizing the dyke, while a corresponding production step may occur afterwards. Please note that the profiles do not necessarily represent a valid state of the world because we decouple resource manipulations from the actual action instance by computing them in isolation. In this view, they are rather a collection of disjunctive, negative events. If the pessimistic profiles do however not indicate a resource over-consumption, then every ground serialization the plan will necessarily comply with the resource restrictions. It is easy to see that this is a correct criterion that does not produce false positive classifications.

During search, the system keeps also track of the antagonistic profiles, the optimistic views on resource utilizations. These views hypothesize, for example, that all deployed units are the same instances (the special force unit is at the same time a truck vehicle), that the sandbags are provided at the beginning of the execution, and that the sandbags are distributed at the very end. Whenever a resource becomes unavailable according to the optimistic profile, over-consumption or over-allocation is inevitable and search has to backtrack if there is no way to produce the resource. While the literature typically focuses only on this best-case view for determining necessary cuts in the search space, we use both optimistic and pessimistic profiles as heuristics for describing the lower and upper bound of resource usage in order to increase the predictive power of the estimate (cf. discussion in [68]) .

Given a resource planning problem $\pi_{\text{RTAP}} = \langle D_{\text{RTAP}}, s_{init}, \mathfrak{s}_{goal}, \langle TE_{init}, \prec_{init}, VC_{init}, RC_{init} \rangle \rangle$, the resource plan $P = \langle TE, \prec, VC, RC \rangle$ is a solution to π_{RTAP} if and only if the following conditions hold:

1. The included task assignment plan $P' = \langle TE, \prec, VC \rangle$ is a solution to the included TAP problem $\pi'_{\text{TAP}} = \langle D_{\text{TAP}}, s_\varepsilon, \top, \langle TE_{init}, \prec_{init}, VC_{init} \rangle \rangle$. Domain model D_{TAP} is thereby the corresponding task assignment model without resource queries and updates in the operator schema definitions.
2. All pessimistic profiles for the non-consumable resources over P are consistent. This in particular implies that RC is a consistent resource constraint network, that means, there exists an assignment of domain values to resource variables that satisfies all equations induced by RC.
3. RC is a refinement of RC_{init}, that means, for all unary constraints $a \leq x \leq b$ that occur in RC_{init} there is a constraint $a' \leq x \leq b'$ in RC with $a \leq a' \leq b' \leq b$.
4. RC is consistent with VC such that every constant assignment in VC is consistent with the equations in RC.
5. RC is consistent with $\prec$ such that the constraints correspond with the partial order.
6. RC is complete in the sense that for every $te \in TE$ that accesses a resource `r`, RC contains the constraints $a \leq in^{\mathtt{r}}_{te} \leq b$ and $c \leq out^{\mathtt{r}}_{te} \leq d$ for some $a, c \geq 0$ and $b, d < \infty$.

The first two arguments refer the symbolic and non-consumable resources: The VC handling of task assignment planning configuration will eventually assign a constant value to each task parameter and therefore it will necessarily comply with the capacity of symbolic resources. The problem with non-consumable resources is the fact that the underlying causal view does not reflect parallel action execution, the pessimistic resource utilization will therefore verify whether the worst-case scenario, that is to say, a maximal parallel execution over-allocates the resource or not.

Conditions two and three confirm that the resource reasoning layer has come to a positive conclusion and that the resulting constraint network has been obtained from the initial one.

The last three conditions ensure that our refinement mechanism has been properly synchronized with the resource reasoning system. The parameter assignments have to be supported by the resource restrictions and vice versa, the parallel allocation of non-consumable, respectively symbolic resources has to be supported

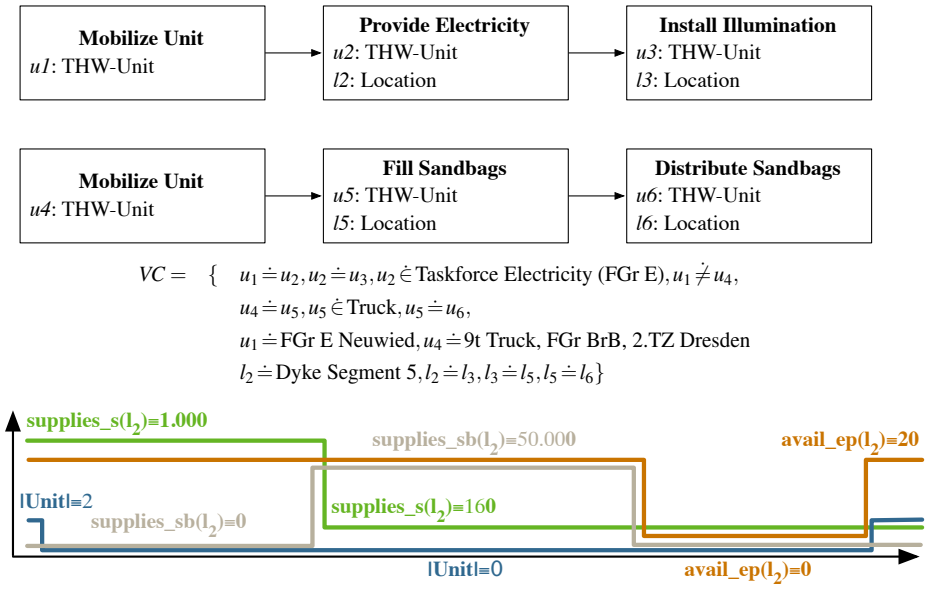


Figure 3.16: A solution to the resource planning example problem of Fig. 3.15.

by the partial order, and finally the resource constraints have to reflect all resources and involved manipulation plan steps. Regarding the last condition, we also want the solution to be definite in the sense that all upper bounds are at least finite (assuming that the constraint propagation will reduce the upper bound to a reasonable limit).

There is a subtle difference between these solution criteria and that of temporal planning: While both configurations are organized such that the task assignment plan (and eventually a causally structured plan) is one "layer" for representing and reasoning about actions and the constraint systems operate on a second, more specific layer, resource manipulations are however explicitly represented in the operator schemata. As a consequence, we have to show that the above definition agrees with the solution criteria in Def. 2.17, in particular with respect to the numerical resources. However, as we have argued above, the computation of resource updates and term evaluations can be performed on a purely symbolic level. That means, when we update a numerical term we do not calculate a numerical value on the semantic level but delegate the actual calculation to an external "computation oracle" that decides on later term queries. In this way, all numbers are represented by the terms that create them (for example, "the term that is the result of adding constants a and b") except for the few constants that we need for defining operator schemata and problem instances. The consequence of this approach is that the resource constraints act as an oracle witness for the underlying causal view of updates and queries and therefore our solution criteria agree.

Before we define the refinement planning function sets, let us examine a solution to the previous problem as it is given in Fig. 3.16. The depicted plan realizes two threads of activity, namely one concerned with installing the illumination and on dealing with fortifying the dyke structure. By assigning co-designations and non-codesignations to the parameters, the plan allocates only two instances of the symbolic resource `Unit` in parallel, which matches its availability. The additional ordering constraints, and in particular `FillSandbags` $\prec$ `DistributeSandbags`, ensure that the required sandbags have been prepared before they are build into the dyke. Please note that the causal dependency between providing an electricity generator and using that electricity at the same place is not reflected in the resource model; the same holds for mobilizing the appropriate unit with the required competences (in the example problem, this knowledge is reflected only implicitly in the co-typing constraints).

Detection Functions – $\mathfrak{Det}_{\mathbf{RTAP}}$

The above solution criteria for the resource planning configuration can be divided into three categories: First, the included task assignment plan must be a solution to the corresponding sub-problem, second, the solution must be a refinement of the problem, and third, the resource reasoning extensions must constitute a solution to the "attached" resource problem. Concerning the first category, $\mathfrak{Det}_{\mathrm{RTAP}}$ will contain all detection functions from the TAP configuration. The second category is implicitly addressed by our refinement generators being sound; these will be defined below. Finally, for the third category we propose the following two flaw detection functions.

The most important flaw is also a critical one; it announces plans for which the resource constraint network has become faulty. It plays its major role in situations in which task parameters occur in resource terms and a variable constraint implies a restriction on the associated resource profile.

Definition 3.31 (Resource Inconsistency)**.** For a given resource plan $P = \langle TE, \prec, VC, RC \rangle$ and problem π, the flaw detection function $f^{det}_{\texttt{ResInconsistency}}$ indicates that the resource constraints in RC have become inconsistent.

Given that RC is inconsistent, let $\{x_1, \ldots, x_n\}$ be those resource variables in RC for which no value assignment can be found, that is to say, for which a unary constraint implies an empty interval or which are involved in a binary constraint that evaluates to an empty interval. Then $f^{det}_{\texttt{ResInconsistency}}(P, \pi)$ is the set of task expressions $\{te_1, \ldots, te_m\} \subseteq TE$ such that for any $1 \leq i \leq n$ there is a $1 \leq j \leq m$ with x_i being $in^{\mathtt{r}}_{te_j}$ or $out^{\mathtt{r}}_{te_j}$ for a resource $\mathtt{r}$. If no such association can be found, the set includes te_{init} and te_{goal} instead. •

As no solution can be obtained from a collapsed constraint set, this detection function is sound.

The second flaw corresponds to the causal interactions, namely a threat between resource utilizations. We basically distinguish two classes of threats, one that emerges due to a potential concurrent access to the same resource and one that indicates if resource bounds are exceeded. The latter may be either over-consumptions or over-productions.

Definition 3.32 (Resource Threat)**.** For a given resource plan $P = \langle TE, \prec, VC, RC \rangle$ and problem π, the flaw detection function $f^{det}_{\texttt{ResThreat}}$ announces critical sets of plan steps that over-allocate a non-consumable or symbolic resource, plan steps on critical paths that over-consume or over-produce a consumable resource, and the set of plan steps that access the same resource concurrently.

$f^{det}_{\texttt{ResThreat}}(P, \pi)$ contains the set of task expressions $\widehat{te} = \{te_1, \ldots, te_n\} \subseteq TE$ if one of the following conditions holds:

1. For every $1 \leq i, j \leq n$ the transitive closure of the plan step ordering does not contain $te_i \prec te_j$. Furthermore, there is a non-consumable resource $\mathtt{r}$ such that for every $1 \leq i \leq n$ RC contains a constraint $lb_i \leq alloc^{\mathtt{r}}_{te_i} \leq ub_i$ and for this resource $\sum_{1 \leq i \leq n} ub_i > out^{\mathtt{r}}_{te_{init}}$ holds. This is the maximal set of plan steps for this resource, so the detection function returns no set $\widehat{te}' \cap \widehat{te} \neq \emptyset$. $f^{det}_{\texttt{ResThreat}}$ issues one flaw for each resource and segment in the partial order in which the over-allocation occurs.
2. For every $1 \leq i, j \leq n$ the transitive closure of the plan step ordering does not contain $te_i \prec te_j$. Let $\widehat{\mathtt{r}}$ be the set of parameters of all plan steps in $\widehat{te}$ that are of sort $\mathtt{r} \leq \texttt{Symbolic}$. Let furthermore $\widehat{\mathtt{r}}^*$ be obtained from $\widehat{\mathtt{r}}$ by iteratively removing parameters r for which $\widehat{\mathtt{r}}$ contains a second parameter $r' \neq r$ such that $r \doteq r'$ holds in the inferential closure of VC. We assume that the detection function returns the maximal set of plan steps for this resource and no other $\widehat{te'} \cap \widehat{te} \neq \emptyset$. $f^{det}_{\texttt{ResThreat}}$ issues one flaw for each potentially over-allocated resource.
3. $\widehat{te}$ is the sequence of plan steps of a profile that over-consumes or -produces a resource. $f^{det}_{\texttt{ResThreat}}$ issues one flaw for each such resource.

•

Please note that regarding the first and second condition, the detection function issues one flaw per over-allocation *episode* and the return sets for different resources may of course overlap. It is also easy to see that the detection function directly implements solution criterion 2 and hence is sound.

Modification Generating Functions – $\mathfrak{Mod}_{\text{RTAP}}$

The refinement options for resource planning are very similar to temporal planning. While the management of the underlying causal structures is under the regime of the modification generating functions from task assignment planning we provide the configuration now with manipulations of the resource constraint set.

Definition 3.33 (Add Resource Constraint)**.** For a given resource plan $P = \langle TE, \prec, VC, RC \rangle$, flaw $\mathtt{f}$, and resource domain model D, the modification generating function $f^{mod}_{\mathtt{AddResConstr}}$ proposes to include a resource constraint as follows: Let $te \in TE \cap comp(\mathtt{f})$ be a flawed plan step and let $lb \leq x \leq ub$ be a resource variable in RC that is associated with te.[12] If the flaw does not contain plan steps we may alternatively check for a variable $v \in \mathcal{V}$ that is a component of the flaw and which is represented by the resource variable x.

Let us assume that the generating function has defined an arbitrary real number $0 < \delta < \infty$. The function then returns

$$f^{mod}_{\mathtt{AddResConstr}}(P, \mathtt{f}, D) \supseteq \begin{cases} \{\quad\} & \text{if } lb = ub \\ \{\langle lb \leq x \leq lb + \delta, \emptyset\rangle, \langle x = lb + \delta, \emptyset\rangle, \langle lb + \delta \leq x < \infty, \emptyset\rangle\} & \text{if } ub = \infty \\ \{\langle lb \leq x \leq \frac{ub-lb}{2}, \emptyset\rangle, \langle x = \frac{ub-lb}{2}, \emptyset\rangle, \langle \frac{ub-lb}{2} \leq x \leq ub, \emptyset\rangle\} & \text{else} \end{cases}$$

•

The systematic reduction of the resource assignments parallels the corresponding modification generating function of the TTAP configuration, given that the app operator (Def. 2.20) treats the resource constraints transparently. It is also a sound plan modification function for exactly the same reasons: we conservatively extend the constraint sets while we address the argument flaw. Please note that when a parameter is asked to be assigned a resource value, we probe a *finite* number of different values.

Future developments will have to focus on more informed ways of narrowing down resource intervals, but also on more sophisticated flaw detections with more predictive power. Like for the temporal constraints, the δ-probing of parameter values is essential for the efficiency of the procedure.

Triggering Function α_{RTAP}

The triggering function for resource planning presents the resource deficiencies not only to the specific modification generator but also to ordering and variable binding refinements. Bottlenecks of over-allocations may be resolved in this way by an appropriate ordering of the steps or an equality constraint. An over-consumption and over-production of a resource can be addressed by adding ordering constraints (in order to improve the pessimistic estimator) or by introducing appropriate range restrictions of the resource variables. Finally, a parallel access has to be avoided and therefore definitely needs a step ordering. Otherwise, this configuration refers to α_{TAP} of the task assignment planning configuration.

$$\alpha_{\text{RTAP}}(\mathbb{F}_x) = \begin{cases} \mathbb{M}_{\mathtt{AddResConstr}} \cup \mathbb{M}_{\mathtt{AddOrdConstr}} \cup \mathbb{M}_{\mathtt{AddVarConstr}} & \text{for } x = \mathtt{ResThreat} \\ \mathbb{M}_{\mathtt{AddResConstr}} \cup \alpha_{\text{TAP}}(\mathbb{F}_x) & \text{for } x = \mathtt{OpenVarBind} \\ \emptyset & \text{for } x = \mathtt{ResInconsistency} \\ \alpha_{\text{TAP}}(\mathbb{F}_x) & \text{otherwise} \end{cases}$$

A resource constraint inconsistency cannot be resolved by a plan modification and is therefore not addressed by any modification generating function. It is a critical flaw and induces a plan discard (cf. Def. 2.35).

[12] We may assume that the function is able to access the resource profiles and can therefore deduce which resources are over-consumed, respectively over-produced.

Inference Functions – $\mathfrak{Inf}_{\mathbf{RTAP}}$

The inference functions in the resource planning configuration are responsible for maintaining the completeness property in the solution criteria: every task expression in the plan that utilizes a resource has to be reflected in the constraint set and the information about any term that occurs in the resource constraints as well as in the variable constraints has to be kept consistent in both sets. This leads us to the following two inference functions:

Definition 3.34 (Resource Variable Introduction)**.** For a given resource plan $P = \langle TE, \prec, VC, RC \rangle$ and planning problem π, the inference function $f^{inf}_{\texttt{IntroResVar}}$ adds appropriate resource constraints to plan P for every plan step that is apparently not yet represented in RC.

For each plan step te in TE that includes in its precondition the query of a term τ of sort Z with $Z \leq$ `NonConsumable` and for which $alloc^{\tau}_{te}$ does not occur in RC:

$$f^{inf}_{\texttt{IntroTempVar}}(P,\pi) \ni \begin{cases} \langle \{0 \leq alloc^{\tau}_{te} \leq \tau'\}, \emptyset \rangle & \text{for query } \equiv (\tau, \min(\tau,\tau')) \\ \langle \{\tau' \leq alloc^{\tau}_{te} < \infty\}, \emptyset \rangle & \text{for query } \equiv (\tau, \max(\tau,\tau')) \\ \langle \{alloc^{\tau}_{te} = \tau'\}, \emptyset \rangle & \text{for query } \equiv (\tau,\tau') \end{cases}$$

If the sort of τ is $Z \leq$ `Consumable`, then the above inference function references resource variable in^{τ}_{te} instead.

For each plan step te in TE that includes in its effects the update of a term τ of sort Z with $Z \leq$ `Numerical` and for which neither in^{τ}_{te} nor out^{τ}_{te} do occur in RC:

$$\langle \{0 \leq in^{\tau}_{te} < \infty, 0 \leq out^{\tau}_{te} < \infty, \min_{te}^{\tau} \leq out^{\tau}_{te} - in^{\tau}_{te} \leq \max_{te}^{\tau}\}, \emptyset \rangle \in f^{inf}_{\texttt{IntroTempVar}}(P,\pi)$$

with te producing τ at rate $n^{\tau}_{te} \in [\min^{\tau}_{te}, \max^{\tau}_{te}]$. •

The main benefit of the above inference function is that it allows to deploy resource-unaware configuration components that can now safely manage the TE set. The same holds for the following inference function that parallels the temporal constraint set synchronization by adding explicit constraints where implicit dependencies exist between the resource and the variable constraint sets.

Definition 3.35 (Resource and Variable Constraint Synchronization)**.** For a given problem π and resource plan $P = \langle TE, \prec, VC, RC \rangle$, the inference function $f^{inf}_{\texttt{IntroVarConstraint}}$ adds variable constraints if they do not reflect a definite result in the resource constraint set and vice versa.

For each variable $v \in \mathcal{V}$ that occurs in VC and for which $\tau \leq v \leq \tau$ holds in RC, the inference function application $f^{inf}_{\texttt{IntroVarConstraint}}(P,\pi)$ contains the modification $\langle \{v \doteq \tau\}, \emptyset \rangle$ if this co-designation constraint does not hold yet in VC. Please note that τ is necessarily a rigid term.

For each variable $v \in \mathcal{V}$ that occurs in RC and for which $v \doteq \tau$ holds in VC, the application of the inference function $f^{inf}_{\texttt{IntroVarConstraint}}(P,\pi)$ contains the plan modification $\langle \{\tau \leq v \leq \tau\}, \emptyset \rangle$ if this resource constraint does not hold yet in RC. •

Both inference functions perform a conservative extension of the resource constraint set and are therefore sound.

Summary of the RTAP Configuration

With the above definitions for the resource planning function components, $\mathfrak{C}_{\text{RTAP}}$ can be summarized as follows:

$$\begin{aligned}\mathfrak{C}_{\text{RTAP}} = \langle \{ & \overbrace{f^{det}_{\texttt{OrdIncons}}, f^{det}_{\texttt{VarIncons}}, f^{det}_{\texttt{OpenVarBind}}, f^{det}_{\texttt{UnordTask}}}^{\mathfrak{Det}_{\text{TAP}}}, f^{det}_{\texttt{ResInconsistency}}, f^{det}_{\texttt{ResThreat}} \}, \\ & \{ \overbrace{f^{mod}_{\texttt{AddOrdConstr}}, f^{mod}_{\texttt{AddVarConstr}}}^{\mathfrak{Mod}_{\text{TAP}}}, f^{mod}_{\texttt{AddResConstr}} \}, \\ & \{ f^{inf}_{\texttt{IntroResVar}}, f^{inf}_{\texttt{IntroVarConstraint}} \}, \\ & \mathfrak{Str} \rangle\end{aligned}$$

Theorem 3.11 (Properness of $\mathfrak{C}_{\text{RTAP}}$)**.** *$\mathfrak{C}_{\text{RTAP}}$ is a proper system configuration in the sense of Def. 3.2.*

Proof. Given that $\mathfrak{C}_{\text{TAP}}$ is proper according to Theorem 3.1, the following arguments hold, analogously to the temporal planning configuration.

$\mathfrak{Det}_{\text{RTAP}}$ is a complete set of sound detection functions, because the included set $\mathfrak{Det}_{\text{TAP}}$ is proven to be complete and the additional detection functions directly correspond to the solution criteria concerning the resource meta-constraint system.

$\mathfrak{Mod}_{\text{RTAP}}$ is a semi-complete set of sound modification generating functions, because the addition of resource constraints is a sound generator that satisfies any proposed flaw if the corresponding refinement is applicable and because it extends $\mathfrak{Mod}_{\text{TAP}}$, which is semi-complete as well.

$\mathfrak{Inf}_{\text{RTAP}}$ has been shown above to be a set of sound inference functions.

It can also be easily seen that $\mathfrak{Mod}_{\text{RTAP}}$ corresponds to $\mathfrak{Det}_{\text{RTAP}}$: every modification generator is assigned a detection function in α_{RTAP}. □

Theorem 3.12 ($\mathfrak{C}_{\text{RTAP}}$ is Modification-Complete)**.** *$\mathfrak{C}_{\text{RTAP}}$ is modification-complete according to Def. 3.3.*

Proof. Given that the included $\mathfrak{C}_{\text{TAP}}$ configuration is modification complete (Theorem 3.2) and that the only un-processed flaw is a critical one (resource constraint inconsistency), the proposition holds. □

3.3.4 Scheduling – $\mathfrak{C}_{\text{SCHED}}$

According to our introduction to scheduling in Sec. 1.2, we understand scheduling as the process of finding a solution to the temporal aspects and resource restrictions of a plan that is given as an input specification. System configuration $\mathfrak{C}_{\text{SCHED}}$ is our corresponding incarnation of the refinement planning framework and it is intended to serve as a scheduling subsystem in all its extensions. We thereby generalize from classical scheduling in two ways: First, we do not focus on the optimization aspects of scheduling, for example, cost minimization, but instead on verifying the feasibility of the resulting schedules. Please note that this does not prevent our approach from employing optimizing methods. Secondly, we assume that the input specification is not necessarily finalized, in particular if our scheduling configuration is extended with respect to activity inserting refinements. Both generalizations are, of course, mutually dependent.

Fig. 3.17 gives an example for this kind of scheduling applications. Project planning software has been used for illustrating several of our configurations, in this case, we address practically all aspects that are depicted in the screenshot. Given the disaster relief mission scenario from resource planning (Sec. 3.3.3), the figure shows actions with absolute durations, for example in line 11 of the left column. Each action is associated with specific resources (lines 12 and 13) that are allocated by the given amount. The time line view gives the user an impression of the plan's progression and the resource utilization. This system corresponds to our view on scheduling because it verifies the availability of the resources at the given times, notifies the user of resource over-consumptions, and also indicates where resource usage is declared but not instantiated (here,

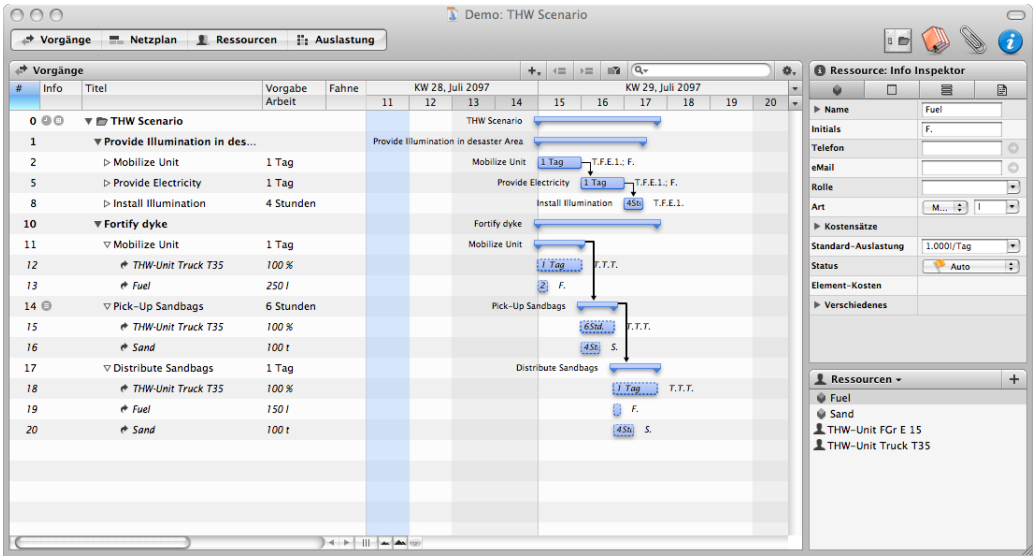


Figure 3.17: Scheduling example, a simplified scenario modelled in and processed by the commercial project planning software Merlin (©2002-2004 by ProjectWizards).

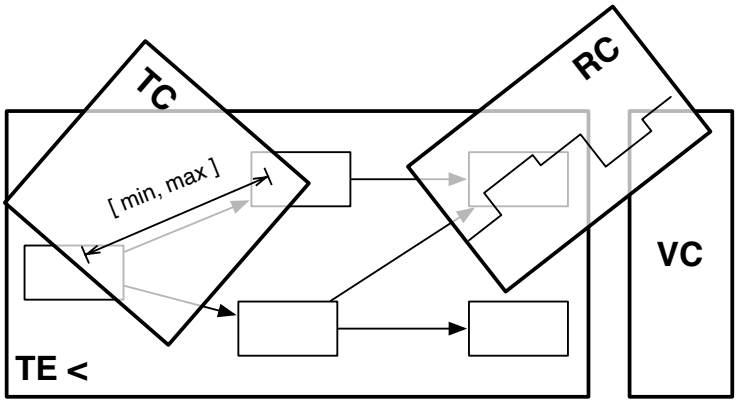


Figure 3.18: The conceptual view on the scheduling system configuration.

this means un-specified staff). The displayed tool's strengths are clearly the visualization of the scenario and less the representational capabilities or automated problem solving competence. The reader may think of the scheduling configuration being a back-end engine to such a tool that deals with more expressive temporal and resource representations and that provides two essential services: It verifies the user's actual specification (the pictured tool does so for significantly limited expressiveness though) and is furthermore able to predict potential bottlenecks and to suggest corresponding corrective measures.

We factorize the scheduling functionality as shown in Fig. 3.18: The first two compartments contribute with reasoning about partially ordered plan steps and parameter relations, denoted by the data structures TE and VC, respectively. Based on this rudimentary plan skeleton we employ two independently working competences for dealing with temporal and resource knowledge (TC and RC). They are coordinated indirectly via the plan skeleton: if and only if the temporal position implies a change on the qualitative plan step ordering, the causal structure of the plan changes, certain execution scenarios are not taken into consideration any more, and hence a re-assessment of the resource situation becomes necessary. On the contrary, resource restrictions may require that certain resource consumption and production options are eliminated from the solution candidates. If and only if this can be achieved by manipulating the ordering relation between those steps, resource reasoning induces a change in the temporal dimension of the plan.

This constellation perfectly fits our planning system configurations for task assignment planning $\mathfrak{C}_{\text{TAP}}$, temporal planning $\mathfrak{C}_{\text{TTAP}}$, and resource planning $\mathfrak{C}_{\text{RTAP}}$. The scheduling configuration will therefore extend both $\mathfrak{C}_{\text{TTAP}}$ and $\mathfrak{C}_{\text{RTAP}}$ in a joint framework instance that is composed of the union of the respective function sets. Our conceptual separation of resource and temporal reasoning thereby resembles the O-OSCAR scheduling approach [50] in which a temporal network is the basis for a meta-CSP that deals with resource allocation interactions [52].

Domain Model Specifics

The scheduling domain model specification is a merge of temporal and resource planning, that means $D_{\text{SCHED}} = \langle \mathscr{M}, \emptyset, \mathtt{T} \rangle$ with $\mathscr{M}$ including the resource-specific sort and function specifications and $\mathtt{T}$ consisting of temporally extended operator schemata that query and manipulate resources via their preconditions and effects, respectively. Concerning domain model consistency, the SCHED configuration benefits from the clean, orthogonal design of its base configurations:

Definition 3.36 (Consistency of Scheduling Domain Models)**.** A scheduling domain model $D_{\text{SCHED}} = \langle \mathscr{M}, \emptyset, \mathtt{T} \rangle$ over a given language $\mathscr{L}$ is called *consistent* if and only if the following conditions hold:

1. The included domain model for temporal planning is consistent (see Def. 3.21).
2. The included domain model for resource planning is consistent (see Def. 3.29).

•

A schedule is consequently the joint data structure of a temporal and resource plan, that means $P = \langle TE, \prec, VC, TC, RC \rangle$ with TC being the set of temporal constraints and RC the respective set of resource constraints. The notion of consistency that is associated with a schedule is therefore directly based on the corresponding consistency definitions for the included substructures.

Definition 3.37 (Consistency of Schedules)**.** A schedule $P = \langle TE, \prec, VC, TC, RC \rangle$ over a given language $\mathscr{L}$ and domain model D_{SCHED} is called *consistent* if and only if the following conditions hold:

1. The included temporal plan $P = \langle TE, \prec, VC, TC \rangle$ is consistent over the included temporal domain model D_{TTAP} (see Def. 3.21).
2. The included resource plan $P = \langle TE, \prec, VC, RC \rangle$ is consistent over the included resource domain model D_{RTAP} (see Def. 3.29).

•

Problems and Solutions

A scheduling problem is a structure that combines the problem specification features of a temporal and resource planning problem in equal shares. With $\pi_{\text{SCHED}} = \langle D_{\text{SCHED}}, s_{init}, \mathfrak{s}_{goal}, \langle TE_{init}, \prec_{init}, VC_{init}, TC_{init}, RC_{init} \rangle \rangle$ we specify the activities that are to be scheduled (TE_{init}), the precedence constraints on those activities ($\prec_{init}$), and the relationships that hold between activity parameters (VC_{init}). Please recall that the temporal constraints (TC_{init}) and the resource constraints (RC_{init}) represent *additional* information on the problem's restrictions that goes beyond the knowledge that is stored in the task schemata.

The global time frame, for example the deadline for finishing the plan, is modelled implicitly in the initial state and goal state description. Since we use the *null plan* representation for describing the problem, this information is stored in the temporal constraints of TC_{init} that refer to the initial and goal state task. In the same way, resource availability and capacities are introduced by resource constraints in RC_{init} on the initial and goal state task.

Given a scheduling problem $\pi_{\text{SCHED}} = \langle D_{\text{SCHED}}, s_{init}, \mathfrak{s}_{goal}, \langle TE_{init}, \prec_{init}, VC_{init}, TC_{init}, RC_{init} \rangle \rangle$, the schedule $P = \langle TE, \prec, VC, TC, RC \rangle$ is a solution to π_{SCHED} if and only if the following conditions hold:

1. The included temporal plan $P' = \langle TE, \prec, VC, TC \rangle$ is a solution to the included temporal planning problem $\pi'_{\text{TTAP}} = \langle D_{\text{TTAP}}, s_\varepsilon, \top, \langle TE_{init}, \prec_{init}, VC_{init}, TC_{init} \rangle \rangle$. Domain model D_{TTAP} is thereby the corresponding temporal planning model without resource queries and updates in the operator schema definitions.
2. The included resource plan $P'' = \langle TE, \prec, VC, RC \rangle$ is a solution to the included resource planning problem $\pi''_{\text{RTAP}} = \langle D_{\text{RTAP}}, s_{init}, \mathfrak{s}_{goal}, \langle TE_{init}, \prec_{init}, VC_{init}, RC_{init} \rangle \rangle$. Domain model D_{RTAP} is thereby the corresponding resource planning model without temporal information annotations in the operator schema definitions.

It is important to take into account that this configuration *is able* to describe and solve classical scheduling problems, including their optimization aspects (given suitable strategy functions). In classical scheduling, however, the solution is implicitly characterized by $TE = TE_{init}$, which is not directly supported by our refinement semantics. $\mathfrak{C}_{\text{SCHED}}$ will consequently act as a classical scheduling engine, but it will not do so in an extended system configuration that provides task inserting plan modification generators.

Detection Functions – $\mathfrak{Det}_{\text{SCHED}}$

The joint of the solution criteria of the TTAP and RTAP configurations is reflected in the structure of the detection functions for the scheduling configuration. Since the problem aspects can be mapped uniquely on each of the two sub-configurations, a union of the two corresponding detection function sets $\mathfrak{Det}_{\text{SCHED}} = \mathfrak{Det}_{\text{TTAP}} \cup \mathfrak{Det}_{\text{RTAP}}$ satisfies the above solution criteria.

Modification Generating Functions – $\mathfrak{Mod}_{\text{SCHED}}$

Regarding the available plan refinements, the scheduling functionality is instantly available in a union of the modification generating function sets $\mathfrak{Mod}_{\text{SCHED}} = \mathfrak{Mod}_{\text{TTAP}} \cup \mathfrak{Mod}_{\text{RTAP}}$. This is possible because our configuration design bases both reasoning layers on the shared, neutral task assignment plan representation. Hence, any additional cross-configuration synchronization mechanism is redundant.

Triggering Function α_{SCHED}

In joint system configuration extensions, one has to be careful about the inter-configuration relationships between the flaws of one configuration and the plan modifications of the other. However, as we have explained above, the included temporal and resource planning configurations do not share any knowledge than that about the underlying TAP plan. Their coexistence in one configuration extension does consequently not impose any modifications on the triggering relation.

$$\alpha_{\text{SCHED}}(\mathbb{F}_x) = \begin{cases} \alpha_{\text{TTAP}}(\mathbb{F}_x) & \text{for } x = \texttt{OpenTmpBind} \\ \alpha_{\text{RTAP}}(\mathbb{F}_x) & \text{otherwise} \end{cases}$$

The function basically mimics α_{TTAP} but instead of passing all unhandeled flaws to the task assignment trigger we employ the resource planning function α_{RTAP}, which in turn preserves the underlying α_{TAP} like in temporal planning.

Inference Functions – $\mathfrak{Inf}_{\text{SCHED}}$

Our arguments for simply joining the modification generating function sets apply as well to the refinement options that are provided by the inference functions. We therefore define $\mathfrak{Inf}_{\text{SCHED}}$ as $\mathfrak{Inf}_{\text{TTAP}} \cup \mathfrak{Inf}_{\text{RTAP}}$ without the need of further adjustments.

Summary of the SCHED Configuration

$\mathfrak{C}_{\text{SCHED}}$, the system configuration for classical scheduling, is composed of the following function sets:

$$\mathfrak{C}_{\text{SCHED}} = \langle \{ \overbrace{f^{det}_{\texttt{OrdIncons}}, f^{det}_{\texttt{VarIncons}}, f^{det}_{\texttt{OpenVarBind}}, f^{det}_{\texttt{UnordTask}}}^{\mathfrak{Det}_{\text{TAP}}}, \overbrace{f^{det}_{\texttt{TempInconsistency}}, f^{det}_{\texttt{OpenTmpBind}}}^{\mathfrak{Det}_{\text{TTAP}}}, \overbrace{f^{det}_{\texttt{ResInconsistency}}, f^{det}_{\texttt{ResThreat}}}^{\mathfrak{Det}_{\text{RTAP}}} \}$$
$$\{ \overbrace{f^{mod}_{\texttt{AddOrdConstr}}, f^{mod}_{\texttt{AddVarConstr}}}^{\mathfrak{Mod}_{\text{TAP}}}, \overbrace{f^{mod}_{\texttt{AddTempConstr}}}^{\mathfrak{Mod}_{\text{TTAP}}}, \overbrace{f^{mod}_{\texttt{AddResConstr}}}^{\mathfrak{Mod}_{\text{RTAP}}} \},$$
$$\{ \overbrace{f^{inf}_{\texttt{IntroTempVar}}, f^{inf}_{\texttt{IntroTempDist}}, f^{inf}_{\texttt{IntroOrdering}}}^{\mathfrak{Inf}_{\text{TTAP}}}, \overbrace{f^{inf}_{\texttt{IntroResVar}}, f^{inf}_{\texttt{IntroVarConstraint}}}^{\mathfrak{Inf}_{\text{RTAP}}} \},$$
$$\mathfrak{Str} \rangle$$

Theorem 3.13 (Properness of $\mathfrak{C}_{\text{SCHED}}$)**.** $\mathfrak{C}_{\text{SCHED}}$ *is a proper system configuration in the sense of Def. 3.2.*

Proof. The property holds because, according to Theorems 3.9 and 3.11, the included $\mathfrak{C}_{\text{TTAP}}$ and $\mathfrak{C}_{\text{RTAP}}$ are proper system configurations. □

Theorem 3.14 ($\mathfrak{C}_{\text{SCHED}}$ is Modification-Complete)**.** $\mathfrak{C}_{\text{SCHED}}$ *is modification-complete according to Def. 3.3.*

Proof. The proposition follows directly from Theorems 3.10 and 3.12. □

3.4 Hybrid Planning and Scheduling – $\mathfrak{C}_{\text{PANDA}}$

With the system configurations for hybrid planning $\mathfrak{C}_{\text{HYBP}}$ (Sec. 3.3.1) and for scheduling $\mathfrak{C}_{\text{SCHED}}$ (Sec. 3.3.4) we have created two cornerstones for realizing a complete coverage of planning functionality aspects as we have stipulated them in the introduction chapter. On the one hand, we have an implementation for integrating hierarchical planning and the concept of synthesizing plans from operators by causal reasoning, on the other hand we have the means for describing, analyzing, and manipulating activity networks that extend temporally and utilize resources. Bringing both together in this joint extension $\mathfrak{C}_{\text{PANDA}}$ will yield two major results: The first and most obvious is that we instantiate in our framework a hierarchical planning system that is able to reason about time and resource demands. Since both configurations will operate within the framework's refinement planning paradigm, the result of the fusion is a really integrated approach in which planning and scheduling completely interleave. Furthermore, if the strategy does not deliberately introduce a bias towards one sub-configuration, the system opportunistically switches between operating in the planning and in the scheduling methodology.

The second result lies in the way the joint extension is realized: If the corresponding function sets were simply joined without further adaptation, the integrated configuration would work as a scheduler not until the primitive operator level is reached by the refinements of the hybrid planner. Although this would be some sort of integration (and in fact the semantic basis of most hierarchical resource planning systems), we missed the opportunity to deal with temporal and resource restrictions on the abstract task level. We will therefore redefine some of the configurations' functions such that the hierarchical aspects of planning are reflected on the scheduling side as well, which will add a novel dimension to scheduling, namely *hierarchical resources*. This topic has been touched briefly in $\mathfrak{C}_{\text{SCHED}}$ when we discussed the role of the sort hierarchy in the context of what it means to allocate a resource of some abstract sort. In this section, we will address the issue of hierarchical resources by identifying four types of abstraction that can be derived from applying our action and state abstraction methods, in particular the state abstraction axioms, to resources. We thereby identify the following basic abstraction principles:

Subsumption: defines one resource to be a specialization of another resource;

Approximation: relaxes upper and lower bounds for numerical values;

Qualification: implements the transfer from symbolic to numerical values;

Aggregation: groups components to super-structures.

The hierarchies that are imposed on resources by these abstractions apparently fit nicely the decomposition hierarchies of actions, such that abstract tasks are not only characterized by abstract symbolic preconditions and effects but also by abstract resource utilization. Furthermore, hierarchical resources are a very natural way of structuring complex domain models. Many of our scenarios are taken from mission documentations of the German *Technisches Hilfswerk* (THW) – a governmental disaster relief organization – within the flood disaster at the river Oder in July 1997. In this domain of crisis management support a large number of resources can be identified that are only used at specific levels of abstraction, respectively resources that need to be differentiated on the concrete level. For example, when securing a dyke, thousands of sandbags have to be prepared and installed, several types of specialized vehicles fortify constructions or build new ones – assisted by workers on the dykes and divers in the water. Furthermore, various kinds of tools and materials have to be organized, including power supplies for the illumination of the working area at night, and the like. On the more abstract level, we find "units" (consisting of "personnel") that use "building materials" and consume "energy" in doing so. On the concrete level the human planner specifies that diesel fuel is needed or that electric power is required to run some machinery.

Integrating hierarchical knowledge in scheduling or resource reasoning is typically understood as propagating the bounds of primitive task networks into their abstractions [61], as structural advice for defining (and efficiently solving) sub-problems [295], or as deducing value assignment strategies from hierarchical resource dependencies [104]. The example scenarios that we have used above for motivating some of our application requirements have been realized in current commercially available project planning systems; their interpretation of hierarchical information is even weaker: the systems allow to define a simple form of aggregation of processes, which is solely used for the purpose of presentation to the user (grouping activities visually under the label of an abstract description) and for displaying cumulative time bounds and resource consumptions.

The fundamental difference between our integration effort and existing approaches is that we base resource utilization in the semantics of our refinement planning framework. In doing so, we achieve a correspondence between the abstraction levels of resources and other state features, which is not only advantageous for modelling purposes but also allows to address temporal and resource issues at any level of abstraction.

Domain Model Specifics

A domain model for hybrid planning and scheduling combines the specifications of the $\mathfrak{C}_{\text{HYBP}}$ and $\mathfrak{C}_{\text{SCHED}}$ configurations. That means, a PANDA domain model over a language $\mathscr{L}$ is given by the structure $D_{\text{PANDA}} = \langle \mathscr{M}, \Delta, \mathtt{T}, \mathtt{M} \rangle$. The logical model $\mathscr{M}$ contains all sort and function specifics from scheduling and Δ is a set of state abstraction axioms (cf. Sec. 3.3.1). The set of task schemata $\mathtt{T}$ consist of primitive operators and complex tasks that are both temporally extended schemata, carrying preconditions and effects (cf. representation of $\mathfrak{C}_{\text{POCLP}}$), and employing resource queries and manipulations.

Resource handling in tasks requires a more detailed description: Action schemata in this configuration, in particular those of complex tasks, are specified by the following structure

$$t(\overline{v}) = \langle \text{prec}(t(\overline{v})), \text{post}(t(\overline{v})), d^{\min}_{t(\overline{v})}, d^{\max}_{t(\overline{v})} \rangle$$

Since $\text{post}(t(\overline{v}))$ is a formula over $\mathscr{L}$(cf. Def. 2.5) and not built from elementary operations, the term update per se is not available. For that reason, the rigid query relation has been declared by the resource planning configuration as an exception that is allowed in effect formulae. The idea is as follows: Instead of explicitly updating the term in the effect formula, we can verify that an appropriate update has happened by comparing the term with the updated value. However, since this verification query is evaluated in the state *after* the transition(s) that the task represents, any reference to the term's value before the transition(s) has to be

preserved by a rigid symbol that is queried in the task's precondition. That means, the manipulation of a resource τ by some operation g is given by the structure $t(\overline{v}) = \langle \ldots \equiv (v, g(\tau)), \ldots \equiv (\tau, v), d^{\min}_{t(\overline{v})}, d^{\max}_{t(\overline{v})} \rangle$, where $v \in \mathscr{V}$ is a parameter of the task. Please note that whether this auxiliary query variable is added to the task parameter list or if it is declared locally to the task is of limited relevance. For practical considerations, we decided to introduce "non-essential" local parameters in task schemata for the sole purpose of resource binding, but this thesis will use the more formal task parameter solution. In any case we may assume that task introduction is always generating "new" variable symbols for these queries.

An important aspect of our abstract resource manipulation approach via queries is that we can specify *ranges* of resource production and consumption. A task specification

$$t(\overline{v}) = \langle \ldots \underbrace{\equiv (v_{lb}, g(\tau))}_{\text{set lower bound}} \wedge \overbrace{\equiv (v_{ub}, h(\tau))}^{\text{set upper bound}}, \ldots \underbrace{\equiv (\max(\tau, v_{lb}), \tau)}_{\text{verify lower bound}} \wedge \overbrace{\equiv (\min(\tau, v_{ub}), \tau)}^{\text{verify upper bound}}, d^{\min}_{t(\overline{v})}, d^{\max}_{t(\overline{v})} \rangle$$

defines resource level τ to be set to a value in the interval $[g(\tau), h(\tau)]$, which is however not necessarily a consumption or a production; in fact, the abstract action may be undecided between reducing and increasing a resource level. We will see the rationale for range usages below.

As an example for the PANDA task specification, we revisit an operator definition in the resource planning configuration:

$$\begin{aligned}
\texttt{DistributeSandbags}(u,d) = &\langle \top, := \texttt{supplies_sb}(\texttt{supplies_sb}(d) - (\texttt{length}(d) \cdot 100)) \rangle \\
&\text{is now defined as follows:} \\
\texttt{DistributeSandbags}(u,d,x) = &\langle \equiv (x, \texttt{supplies_sb}(d) - (\texttt{length}(d) \cdot 100)), \\
&\quad \equiv (\texttt{supplies_sb}(d), x), \\
&\quad d^{\min}_{\texttt{DistributeSandbags}}, d^{\max}_{\texttt{DistributeSandbags}} \rangle
\end{aligned}$$

Please note, that the $\equiv$ relation is not discriminating the position of the arguments and therefore must not be confused with the assignment operation in programming languages or PDDL resource manipulations [99]. In our example, the semantics of the task effect is clearly to verify that the (flexible) term $\texttt{supplies\_sb}(d)$ has decreased its value by the given amount, that is to say, it changed its interpretation to the rigid value of parameter x. However, if we bound the flexible term in the precondition to another flexible one, say, $\equiv (\texttt{supplies\_sb}(d'), \texttt{supplies\_sb}(d) - 1)$, and in the postcondition accordingly $\equiv (\texttt{supplies\_sb}(d'), \texttt{supplies\_sb}(d))$, the meaning of the task is not an implicit reduction of $\texttt{supplies\_sb}(d)$ by 1, because we did not store any information about the term interpretation between one state and its successor. We can only deduce is that the relative distance between the interpretations is reduced from 1 to 0, which can also be achieved by increasing the reference term (which *may* have been the intended meaning). Although these are all consequences of a coherent semantic basis that is very compact and precise, we recommend domain modelers that are used to more "practitioners oriented" specification languages to make themselves particularly aware of these aspects. In this context it is also worth noting that a supposed consumption effect $\equiv (\texttt{supplies\_sb}(d), \texttt{supplies\_sb}(d) - 1)$ is trivially inconsistent in all reasonable models, because a numerical term can never evaluate to its own decrement.

Before we can define decomposition methods, we have to introduce the plan structure for the PANDA configuration. Hybrid planning and scheduling thereby combines all the features of its sub-configurations: A plan $P = \langle TE, \prec, VC, CL, TC, RC \rangle$ consists of the usual set of plan steps TE, ordering constraints $\prec$, and variable constraints VC. CL denotes the set of causal links like they are defined in $\mathfrak{C}_{\text{HYBP}}$, and TC and RC are the temporal constraints, respectively resource constraints from $\mathfrak{C}_{\text{SCHED}}$. Consistency for such plans is derived from the respective configuration's consistency criteria.

Definition 3.38 (Consistency of Plans for Hybrid Planning and Scheduling)**.** A plan for hybrid planning and scheduling $P = \langle TE, \prec, VC, CL, TC, RC \rangle$ over a given language $\mathscr{L}$ and domain model $D_{\text{PANDA}} = \langle \mathscr{M}, \Delta, \mathtt{T}, \mathtt{M} \rangle$ is called *consistent* if and only if the following conditions hold:

1. The included hybrid plan $P' = \langle TE, \prec, VC, CL \rangle$ is consistent over $\mathscr{L}$ and the corresponding domain model fragment D_{HYBP} (see Def. 2.12).

2. The included schedule $P'' = \langle TE, \prec, VC, TC, RC \rangle$ is consistent over $\mathscr{L}$ and the corresponding domain model fragment D_{SCHED} (see Def. 3.37).

•

Since the hybrid planning and scheduling approach operates with complex task expressions, plan consistency becomes a criterion for the legality of the included decomposition methods.

Definition 3.39 (Consistency of Hybrid Planning and Scheduling Domain Models)**.** A hybrid planning and scheduling domain model $D_{\text{PANDA}} = \langle \mathscr{M}, \Delta, \mathtt{T}, \mathtt{M} \rangle$ over a given language $\mathscr{L}$ is called *consistent* if and only if the following conditions hold:

1. The included hybrid domain model is consistent (see Def. 3.19).
2. The included scheduling domain model is consistent (see Def. 3.36).
3. For all method specifications $m \in \mathtt{M}$ with $m = \langle t(\overline{v}), \langle TE, \prec, VC, CL, TC, RC \rangle \rangle$:
   a) For every resource $\mathtt{r}$ that is queried or manipulated by $t(\overline{v})$, let RC_{opt} be the optimistic profile of the resource plan that is included in the method's RTAP network $\langle TE, \prec, VC, RC \rangle$. The input level of the first, that is to say, the most producing plan step in the leading stratum (see p. 121) is given by the resource variable $in^{\mathtt{r}}_{opt}$; the output level of the last plan step in the profile, the least consuming one in the last stratum, is represented by $out^{\mathtt{r}}_{opt}$. Let furthermore $RC_{t(\overline{v})}$ be the set of resource constraints that define the input and output levels of the complex task schema, $in^{\mathtt{r}}_{t(\overline{v})}$ and $out^{\mathtt{r}}_{t(\overline{v})}$, as well as the appropriate change rates according to the complex task schema (see p. 120). In a consistent domain model, the resource constraint set
   $$RC_{opt} \cup \{in^{\mathtt{r}}_{t(\overline{v})} = in^{\mathtt{r}}_{opt}, out^{\mathtt{r}}_{t(\overline{v})} = out^{\mathtt{r}}_{opt}\} \cup RC_{t(\overline{v})}$$
   is a consistent resource constraint network, as well as the analogous constraint set
   $$RC_{pess} \cup \{in^{\mathtt{r}}_{t(\overline{v})} = in^{\mathtt{r}}_{pess}, out^{\mathtt{r}}_{t(\overline{v})} = out^{\mathtt{r}}_{pess}\} \cup RC_{t(\overline{v})}$$
   with the respective pessimistic profile RC_{pess} and levels $in^{\mathtt{r}}_{pess}$ and $out^{\mathtt{r}}_{pess}$.
   b) Let $end_{t(\overline{v})}$ be in the interval $[start_{t(\overline{v})} + d^{\min}_t, start_{t(\overline{v})} + d^{\max}_t]$, according to the complex task schema. Let furthermore TC' be a temporal constraint network that is obtained from TC by including all constraints that are implied by TE and the step ordering $\prec$ (see p. 112), as well as two additional constraints for each task Expression $te \in TE$, namely $start_t \leq start_{te}$ and $end^{\max}_{te} \leq end^{\max}_t$. In a consistent domain model, TC' is a consistent temporal network.

•

The above definition of domain model consistency transfers our established notion of abstraction into the scheduling configuration aspects of $\mathfrak{C}_{\text{PANDA}}$: a concrete solution has to be a solution on the abstract level as well. The first method criterion means that the bounds for each resource $\mathtt{r}$ that are induced by the complex task schema have to be valid bounds for the implementing task network. The same concept holds for the temporal constraints in the second criterion, that requires the time window of the implementation to be included in the time window of the abstract action. Since in a hybrid planning and scheduling domain model the complex tasks have multiple methods defined for their implementation, the consumption or production range of an abstract action becomes a reasonable modelling concept. As we have seen above, our complex task model is able to express such ranges.

Relying on the representation concepts for abstraction in planning with resources, which we have defined so far, we will now present the utilization of these abstraction hierarchies in our refinement planning framework.

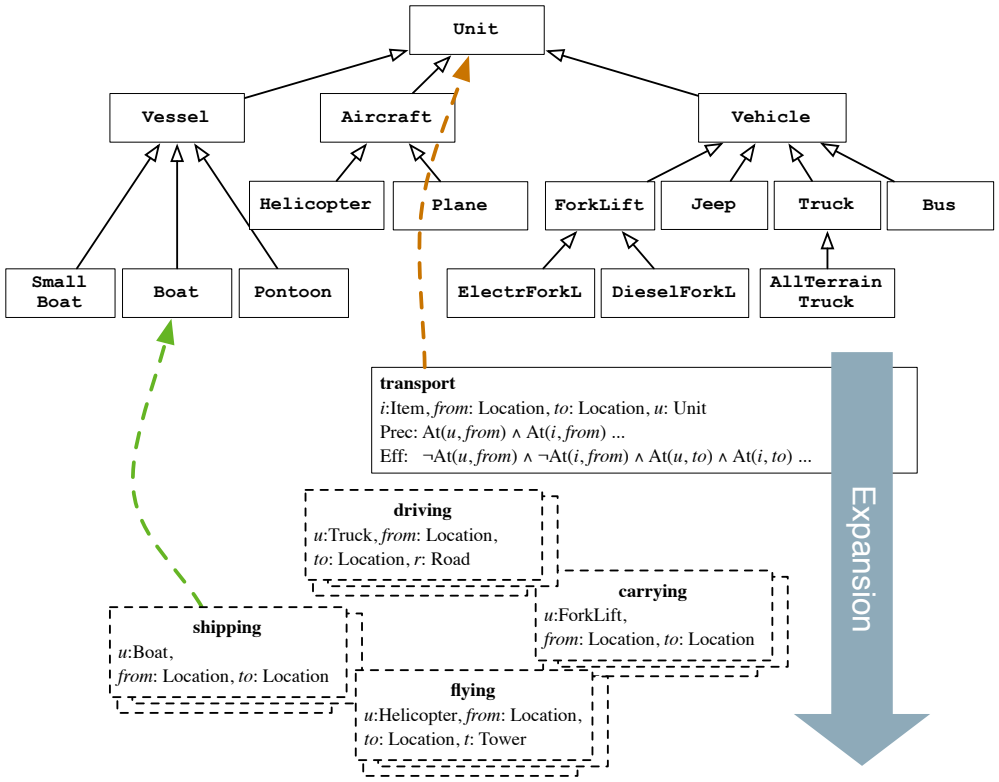


Figure 3.19: Abstraction of resources by building a subsumption hierarchy.

Subsumption

Subsumption is expressed by defining one type of resource to be a specialization of another via the sort hierarchy of the domain model. We have introduced this kind of resource abstraction briefly in the resource planning configuration. Fig. 3.19 shows an excerpt of a sort hierarchy on resources that is part of the example domain that we have shown in Fig. 3.14. Typical disaster relief missions involve several types of transportation units; we therefore define an abstract transportation task allocating one unit of the abstract resource THW `Unit` – indicated by the dashed orange arrow. After a task decomposition refinement step, the task is specialized into a more concrete way of transportation. For example, the configuration instance chooses a method that decomposes the abstract task into a network that implements a shipping transportation task. According to the complex task signature and a state abstraction axiom that is defined as

$$\begin{aligned} &\forall u_{\texttt{Unit}}, from_{\texttt{Location}} \quad \exists v_{\texttt{Vehicle}}, a_{\texttt{Aircraft}}, b_{\texttt{Boat}}, r_{\texttt{Road}}, h_{\texttt{Height}}, w_{\texttt{WaterStreet}} : \\ &\texttt{At}(u, from) \Leftrightarrow \quad \texttt{StandingAt}(v, from, r) \wedge \equiv (u, v) \quad \vee \\ &\qquad\qquad\quad \texttt{AircraftAt}(a, from, h) \wedge \equiv (u, a) \quad \vee \\ &\qquad\qquad\quad \texttt{BoatAt}(b, from, w) \wedge \equiv (u, b) \quad \vee \ldots \end{aligned}$$

one unit of the more concrete resource `Boat` has to be allocated (dashed green arrow), which is consistent with the subsumption defined above.

Reasoning within this hierarchy is quite similar to the mechanism involved in dealing with symbolic causal interactions in the hybrid planning configuration. Since every resource qualifies for being allocated instead of one of its super-sorts, usage profiles have to take into account that

1. every allocation of a resource implies an allocation of the respective super-sort, and
2. every allocation of a resource implies a *possible* allocation of every sub-sort.

As a consequence, every profile over a resource in *RC* has to include the profiles of subsumed resources.

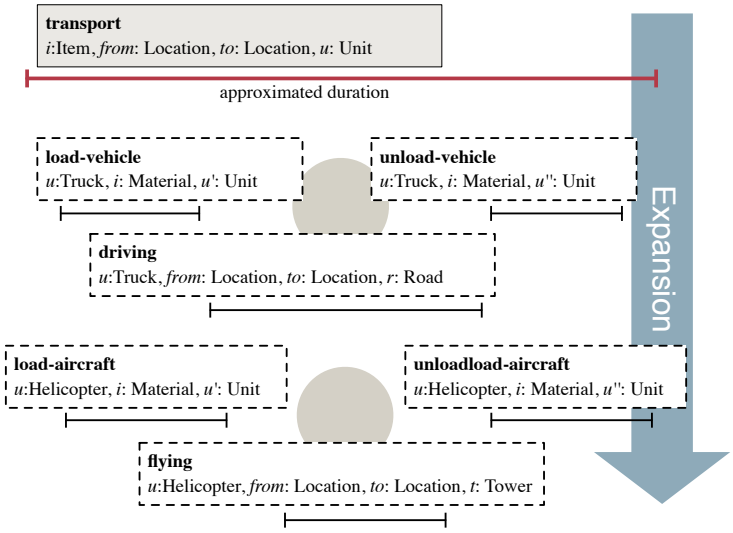


Figure 3.20: Abstraction of resources by approximation. The abstract task over-estimates the duration of every expansion.

Approximation

Numerical values for resources and their manipulation operations may be estimated on the more abstract level of a plan, which we call an *approximation*. Durations for tasks are often approximated, as their exact time consumption cannot be calculated precisely until the very concrete plan refinement level has been reached with all necessary tasks inserted in the plan. Fig. 3.20 shows possible refinements of the transportation task together with an informal account of the respective durations.

In this example, the overall time interval for the abstract transportation task over-estimates all of its possible refinements. The sum of the durations for the expansion that contains the transportation by helicopter and its loading operations is in particular smaller than that of the tasks involved in delivery by truck.

The estimation of duration parameters is modelled, as described in the temporal planning configuration, by duration intervals with lower and upper bounds ranging from zero to infinite. Infinite upper and zero lower bounds represent open intervals for the informal concepts “at least” and “at most”, respectively. Approximation facilitates a very natural way of changing the view on numerical values from one level of abstraction to the next. In a situation like that depicted in Fig. 3.20, we could like to model that the abstract transportation task takes at least four hours, the loading procedure at most ten minutes, and so on. We note however, that by this kind of (numerical) approximation we may sacrifice an important monotonicity property, that is to say, it does not guarantee to over-estimate increasing resource manipulations and under-estimate decreasing ones along the task refinements in a *predictable* way: the abstract task uses a resource “at least”, its sub-task uses it “at most”, and the third abstraction level uses again an “at least” formulation (given that the task specifications are consistent, the constraints will eventually narrow down all ranges). It may therefore not be usable as a heuristic for reducing the search space as efficiently as desirable: For example, a resource over-consumption may turn out harmless after another refinement step if production and consumption are not consistently over- and under-estimated, respectively. Another point here is the accuracy – or in some sense admissibility – of the approximating function. The more precise the estimations are, the better can resource information guide search.

On the other hand, we can use the method definitions to check for inconsistently over- and under-estimating approximation hierarchies off-line. Similar calculations can guide the scheduling process by suggesting to assign interval restrictions as soon as particular expansions are ruled out by the planning process: According to Fig. 3.20, after the system has unsuccessfully tried to use the transporting method using trucks, the abstract time estimation can be reduced.

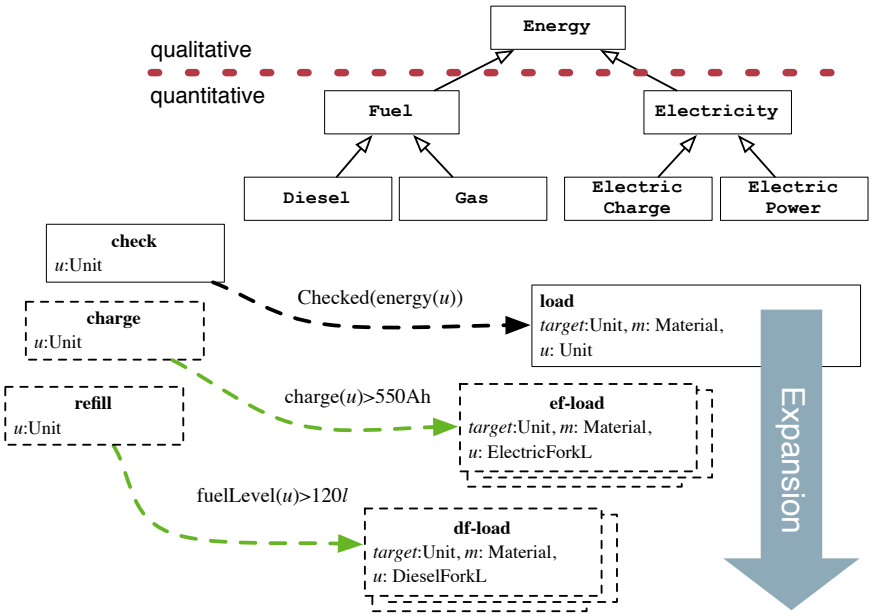


Figure 3.21: *Energy* as abstraction of resources by qualifying the numerical resources *Fuel* and *Electric Power*.

Qualification

In some cases, approximation may not make much sense when building a task hierarchy. For the abstraction of quantitative, numerical values, a modeler may want to specify a phase transition from quantitative descriptions to qualitative, symbolic terms. This transition is referred to as *qualification*. An example: There are many ways for loading material on a transportation unit, and all of them consume a certain (possibly approximated) numerical amount of energy. The two gadgets for performing the tasks are electrical fork lifts used at urban supply centers and all-terrain fork lifts driven by diesel fuel that operate on site. As their two energy sources do not reasonably correlate with respect to their consumption to allow for a good estimation, we abstract from concrete numbers and intervals and speak of energy in general that has to be checked or refilled before the loading task can be performed.

Fig. 3.21 shows such a situation: The resource hierarchy at the top implements qualification (see the dotted line) by defining `Energy` as the super-resource of `Fuel` and `Electricity` with `Fuel` subsuming diesel fuel and gas and with `Electricity` subsuming the electric charge of batteries and the electric power of generators. The abstract loading task `load` on the right hand side includes the resource allocating term over function $energy : \texttt{Unit} \rightarrow \texttt{Energy}$ in its precondition. We assume that (the non-numerical) energy has to be verified beforehand by an appropriate inspection task `check`, which carries a classical postcondition for contributing the appropriate state feature (denoted by the dashed arrow).

The example furthermore indicates two methods for implementing the loading procedure, namely one that is dedicated to dealing with an electric fork lift and another that describes operating a diesel driven model. The relevant precondition literals employ functions $charge : \texttt{Unit} \rightarrow \texttt{ElectricCharge}$ and $fuelLevel : \texttt{Unit} \rightarrow \texttt{Fuel}$; for the sake of readability, we make use of an abbreviated notation $a > b$ instead of the query literal $\equiv (a, \max(a,b))$. The abstract energy resource and its consumption are related to their concretization by the techniques that we call subsumption and qualification: With a sort hierarchy like that of Fig. 3.21 and an occurrence of resources in the state abstraction axioms like

$$\forall u_{\texttt{Unit}} : \texttt{Checked}(energy(u)) \quad \Leftrightarrow \quad charge(u) > 550 \vee fuelLevel(u) > 120 \vee \ldots$$

the abstract energy usage is semantically connected with numerical resource profiles. The above axiom implies that if the abstract resource energy is characterized as being “checked” and therefore available on

the abstract state level, this means for a refinement state that a numerical quantity of 550 ampere-hours of electric charge is accessible, respectively, 120 liters of diesel fuel. The figure depicts the refined causality by green dashed arrows between the decomposition alternatives for preparation and loading tasks (`ef_load` and `df_load`).

The hybrid planning and scheduling configuration addresses resource qualification in profile computations by exploiting the sort hierarchy and state abstraction axioms as follows:

1. For each plan step, for each occurrence of a literal that constitutes a qualified resource, the corresponding allocations are propagated into the resource constraint set. According to the subsumption procedure, this is to be done in all profiles concerning the quantitative sub-resources.
2. The formula that qualifies a resource holds in a state if the corresponding resource queries and conditions are satisfied.

This allocation rules imply, that in the above example the abstract loading task consumes (potentially) $550Ah$ of electric charge and at the same time $120l$ of diesel fuel until that task is decomposed and the resource consumption is specialized accordingly, for instance, into the electric variant (which releases the amount of $120l$ fuel in the `Diesel` profile). On the other hand, the availability of the given amount of electric charge is interpreted to be sufficient for satisfying the abstract need for energy. Although this is regular state abstraction axiom reasoning, it requires an "oracle", in this case the resource subsystem, to compute the numerical profile levels for the state in question.

The rationale behind this kind of hierarchical relation is to *determine* a symbolic causal structure on the abstract level and to refine it into numerical quantities as soon as the planner decided which expansion schemata are appropriate. Qualification is conceptually related to approximation and hence it may be a worthwhile effort in future work to integrate them into one abstraction principle. In particular temporal reasoning seems to benefit from a qualification step with very natural specifications such as task durations that are "long" or starting times being "early in the morning".

Aggregation

Aggregation is a structural abstraction, used for complex resources that are composed of a number of more or less independent components, which are in turn regarded as components as well. The disaster relief scenario includes many examples for aggregations. A set of several THW units, for instance, is typically organized in a platoon such that the units' capabilities make the platoon a multi-purpose aggregate that is deployable in various operation scenarios. The basic configuration of the platoon aggregate consists of a small bus for the transportation of persons, two heavy trucks for carrying various gadgets, one mission-specific vehicle, and the personnel for operating the platoon. Specific services are realized by extending the basic setup, as shown in Fig. 3.22. For example, if supporting measures for broken down energy supplies or drinking water are needed, mission control does not compile the appropriate equipment from their inventory but deploys a so-called infrastructure platoon, which is pre-configured with all the required tools and devices mounted on its trucks. On site, the infrastructure platoon can be split up so that the power lines are repaired independently from handing out additional equipment to the relievers, etc.

Fig. 3.23 shows an abstract task for organizing power supplies, which allocates one of the infrastructure platoons (see dashed arrow). The components of the platoon are distributed among the tasks in the expansion network like it is depicted with the small arrows: For example, checking the feeders in the area needs one of the all terrain trucks (`ATTruck`) and ten people working. We assume that all relevant access functions are declared in the domain model, including mtw : `Platoon` $\rightarrow$ `Bus` for associating a platoon with its crew car ("Mannschaftstransportwagen"), $mlw1$: `Platoon` $\rightarrow$ `ATTruck` for defining the respective all-terrain multi-purpose truck, rs : `Platoon` $\rightarrow$ `RadioSet` for specifying the the radio-communication facility, and finally

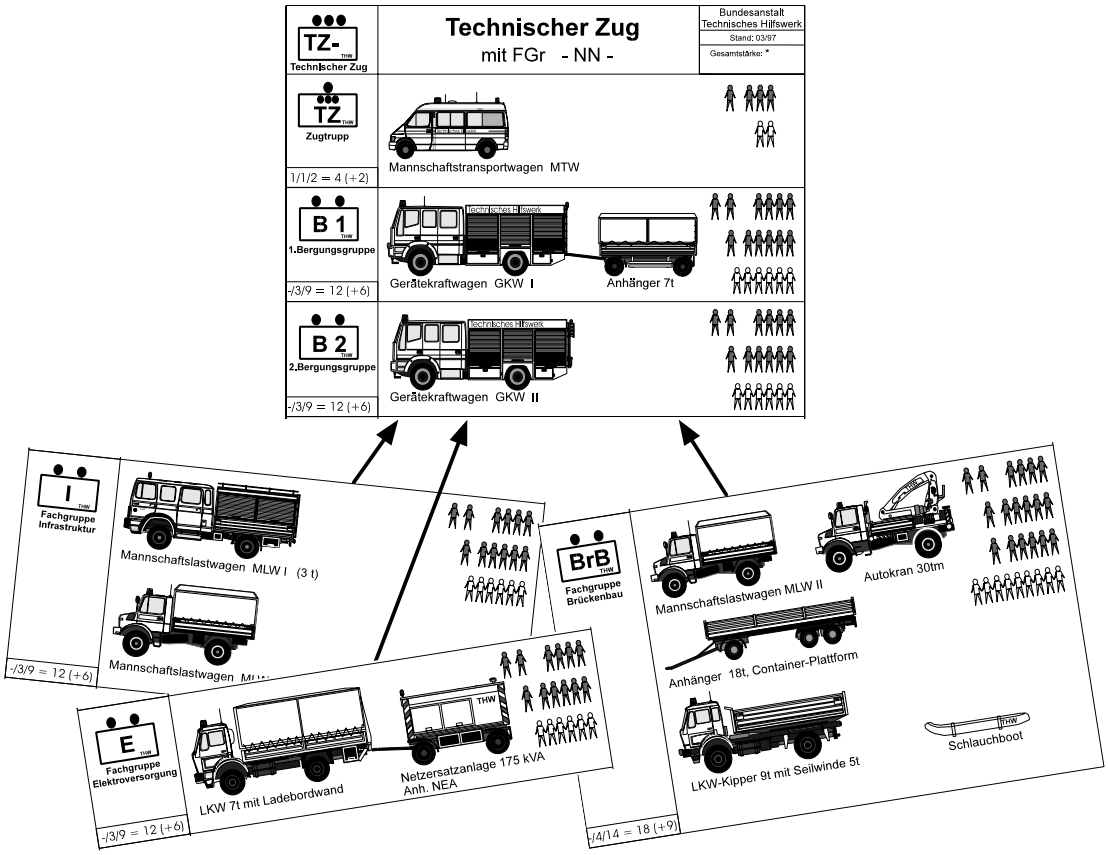


Figure 3.22: Resource aggregation in the disaster relief domain: Platoon structures of the THW.

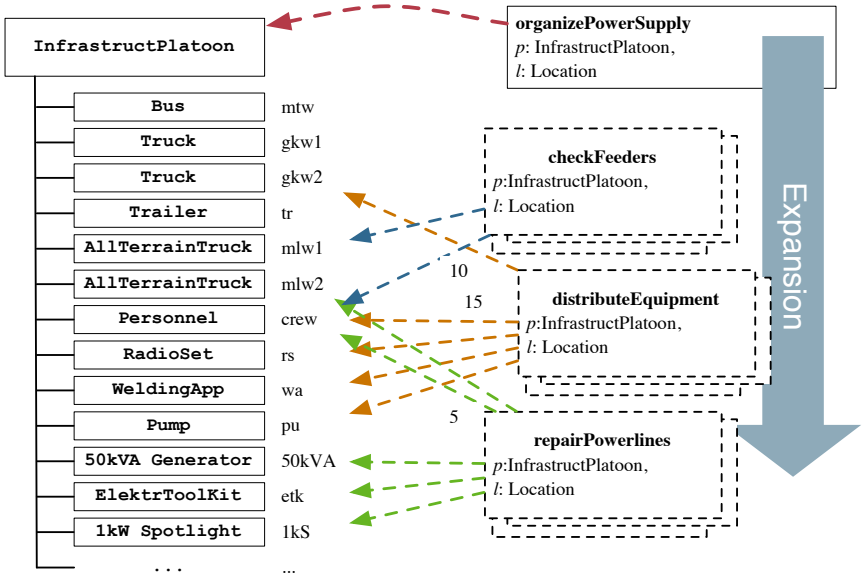


Figure 3.23: Abstraction of resources by aggregating components.

crew : `Platoon` → `Personnel` that gives for a platoon the number of crew members. The platoon aggregation is then described by declaring state abstraction axioms like the following one:

$$\begin{aligned} & \forall p_{\texttt{InfrastructPlatoon}}, l_{\texttt{Location}}, r_{\texttt{Road}} : \\ \texttt{Available}(p,l) \quad \Leftrightarrow \quad & \texttt{StandingAt}(mtw(p),l,r) \wedge \texttt{MapFeature}(l,r) \wedge \\ & \texttt{Operational}(rs(p),l) \wedge \equiv (crew(p),40) \wedge \ldots \\ & \vee \ldots \end{aligned}$$

Reasoning about aggregated resources parallels reasoning about qualified ones, because it combines state abstraction axiom analysis with the notion of implied and possibly implied allocations. In order to project resource usage the system has to consider the following:

1. In every resource profile, every allocation of an aggregated resource implies the allocations for each of its components like they are specified in the respective state abstraction axioms.
2. In every resource profile, every allocation of a potential aggregation component implies a possible allocation of a corresponding aggregate.

Two remarks concerning the above allocation propagation rules: First, the situation is in some sense the inversion of subsumption, because it is the component that possibly implies an aggregate allocation (in this view a top-down rule and a bottom-up heuristic, while subsumption is more of a bottom-up rule and top-down heuristic). Second, the term "potential component" refers to the possibility that the allocated resource is either a component of an existing or not yet identified aggregate or that it is an autonomous object that is not part of any aggregation. If in our example, some buses are defined as autonomous entities, too, then an allocation of a bus may turn out to refer to the independent object and one platoon is released in the corresponding resource profiles. Furthermore, it has to be noted that the application of the second rule also depends on the restrictions that the domain model imposes on sharing components. Given the above example, if a task tries to allocate one radio set, the second rule implies the allocation of an infrastructure platoon. The platoon resource can release the additional allocation only if an existing aggregate can be consistently identified and reused, which means, according to the state abstraction axiom, that there has to be an infrastructure platoon present at the very location at which the radio set is to be used.

Problems and Solutions

A hybrid planning and scheduling problem is a combination of the sub-configurations' problem structures, that means, it is given as the joint problem specification of the ones developed in $\mathfrak{C}_{\text{HYBP}}$ and $\mathfrak{C}_{\text{SCHED}}$: $\pi_{\text{PANDA}} = \langle D_{\text{PANDA}}, s_{init}, \mathfrak{s}_{goal}, \langle TE_{init}, \prec_{init}, VC_{init}, CL_{init}, TC_{init}, RC_{init} \rangle \rangle$. As it is usual for the *null plan* representation, we encode the initial state and the goal state specification as artificial plan steps, including the global resource constraints and temporal annotations.

A problem specification may be given as illustrated in Fig. 3.24: The initial task network describes the problem of organizing the power supplies by a given infrastructure platoon in a given area. The figure sketches a possible implementation (task schema signatures in the beige box) that assumes that electricity can be restored by repairing the damaged power supplies and therefore includes checking the feeders, providing the necessary tools and equipment, and finally performing the concrete repair procedures. This example focuses on the two sub-tasks `distributeEquipment` and `repairPowerLines`: the former is supposed to transport tools and supplies to an on-site camp, the latter installs one of the larger power generators at a presumably broken supply node. The light blue boxes denote the implementations of these two sub-tasks. The indent depth thereby indicates the partial ordering of sub-tasks. For example, the implementing method of `distributeEquipment` begins with two loading tasks followed by a transportation respectively driving step and the unloading operations at the end. We skipped the refinement of the transportation (cf. Fig. 3.19), which is forced by the variable binding constraints of the surrounding network to be instantiated with one of the `Truck` units, and hence no other expansion method (flying, etc.) can be used.

On the left hand side, the preparation of the profile calculation is depicted by showing some of the allocated resources at the current level of plan refinement: The profiles for abstract THW units can identify two disjoint time points for `Unit` allocations in the network of `distributeEquipment` and one in that of

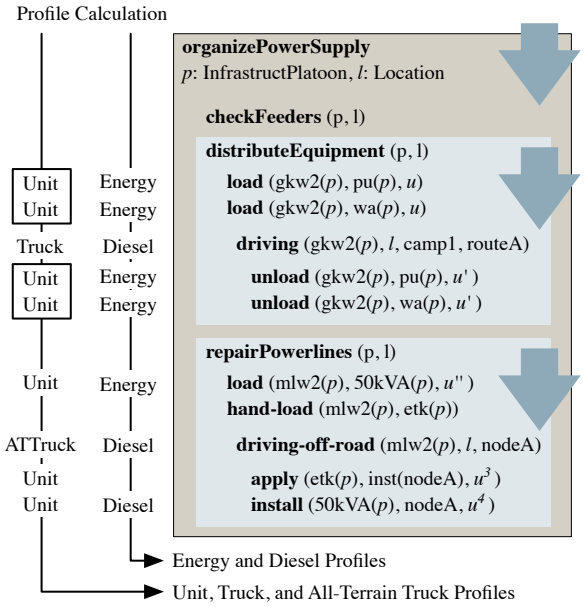


Figure 3.24: Example for some resource profiles after the expansion of an abstract task.

`repairPowerLines`. Let co-designation constraints in the first task network assign the same object to the two loading tasks, and the same to be used again in the unloading procedure respectively: this suggests ordering constraints among the loading and unloading tasks (cf. $\mathfrak{C}_{\text{RTAP}}$). The resource reasoning subsystem takes this into consideration by assuming the allocation of at most two instances of `Unit`, namely two at the beginning of the loading tasks and a third at the beginning of the unloading, which can be re-allocated after the loading procedure releases it. For every loading operation, energy checks are implied, as each of them allocates the symbolic resource `Energy` (cf. Fig. 3.21). In addition, `Diesel` is consumed for the transporting trucks and the power generator, and the trucks itself have to be allocated during driving.

The profile projections for the different resources involved are shown in Fig. 3.25. Their calculation follows straight forward from the task definitions. Please note, that the upper three resources are de-allocated after use, while the others are consumed.

In addition to these explicit profile manipulations, there are implicit manipulations induced according to the abstraction hierarchy like we have described in the previous section. They are reflected in changes of related resource profiles. Fig. 3.25 also shows the changed profiles according to the resource hierarchies. We will focus on the first two time points.

The changes in the beige areas are made according to the resource sort hierarchy, that means, according to subsumption. The two THW `Unit` objects at the first time point are potentially assigned to constants of sub-sort `Truck` or that of the all-terrain vehicles `ATTruck` (other sub-sorts are omitted for brevity). At the second time point, the system propagates the de-allocation of the abstract resource into the sub-resource, while at the same time the explicit allocation of the `ATTruck` unit by the off-road transport is reflected in the super-sort `Truck` as an implicit allocation.

The blue areas describe the propagation of a qualified resource: The abstract loading tasks need in the beginning two `Energy` sorted objects. According to the qualification, each of them can be specialized into 120*l* of diesel fuel (gas and electricity are again omitted). The explicit allocation of another 150*l* of diesel by truck driving implies further `Energy` consumption on the abstract level (2 abstract units).

We may assume that, according to the problem specification, the capacity of diesel fuel is found to be exceeded somewhere in the middle of the plan. The conflict resolution strategies for manipulating the ordering cannot resolve the problem, as there are no suitable production tasks available in the current plan. Since

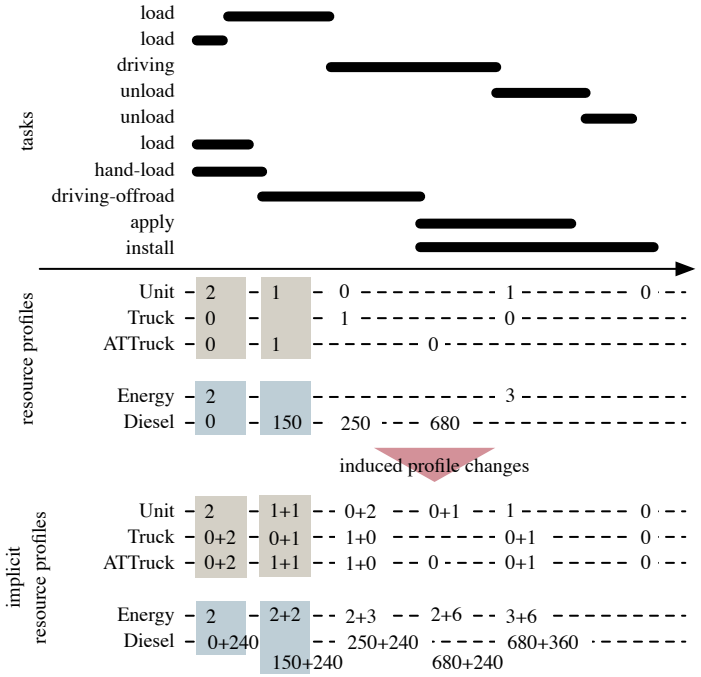


Figure 3.25: Resource profiles for the example scenario shown in Fig. 3.24.

this over-consumption is partly caused by charging `Diesel` implicitly, a solution to this flaw is obviously to build a plan refinement that conducts a specialization of the abstract energy allocations into other resources. As we will see in more detail below, decomposing one of the loading tasks into an implementation that references `ef_load` achieves this, for it consumes electric power (see Fig. 3.21).

In this example, we may furthermore assume that after performing a task expansion an over-allocation occurs on the abstract level for sort `ElectrForkL`, because only one electric fork lift is present on site and the two loading tasks still not necessarily demand for the same object. The conflict can be resolved by co-designation plan modifications that assign the same fork lift resource to the loading operations, followed by an ordering refinement for sequencing them. Alternatively, a resource producing plan step may be introduced, for example transporting a fork lift from a central depot to the operation site.

The example has shown the particular role of action abstraction in the hybrid planning and scheduling configuration $\mathfrak{C}_{\text{PANDA}}$ and has made clear some of the aspects that have to be considered in the expansion refinement. We will examine the corresponding modification generating function in more detail below.

In the described manner, the refinements of the problem specification are systematically explored for the occurrence of a plan that is considered a solution. Given a hybrid planning and scheduling problem $\pi_{\text{PANDA}} = \langle D_{\text{PANDA}}, s_{init}, \mathfrak{s}_{goal}, \langle TE_{init}, \prec_{init}, VC_{init}, CL_{init}, TC_{init}, RC_{init} \rangle \rangle$, the plan $P = \langle TE, \prec, VC, CL, TC, RC \rangle$ is a solution to π_{PANDA} in the PANDA configuration if and only if the following conditions hold:

1. The included hybrid plan $P' = \langle TE_1, \prec_1, VC_1, CL_1 \rangle$ is a solution to the hybrid planning sub-problem $\pi_{\text{HYBP}} = \langle D_{\text{HYBP}}, s_{init}, \mathfrak{s}_{goal}, P_{init} \rangle$ with D_{HYBP} being a fragment of D_{PANDA} that does not contain tasks with temporal annotations.
2. The included schedule $P'' = \langle TE, \prec, VC, TC, RC \rangle$ is a solution to the scheduling sub-problem $\pi_{\text{SCHED}} = \langle D_{\text{SCHED}}, s_{init}, \mathfrak{s}_{goal}, \langle TE_{init}, \prec_{init}, VC_{init}, TC_{init}, RC_{init} \rangle \rangle$ with D_{SCHED} being a fragment of D_{PANDA} that does not contain decomposition methods. We may assume that all pre- and postconditions that do not induce resource manipulations are simply ignored.

Please note that in particular the second criterion ultimately refers to our refinement semantics and does therefore not need decomposition methods to ground the abstract tasks in *sequences* of operators. A partially ordered plan may allow for a safe parallel action execution.

Detection Functions – $\mathfrak{Det}_{\text{PANDA}}$

The set of detection functions for the hybrid planning and scheduling configuration is the union of the respective sets of the system configurations HYBP and SCHED. On the flaw detection side, the integration of resource reasoning and the notion of abstraction is realized via appropriate constraint propagation rules in the profile calculation, in other words, inside the "oracle" for querying flexible terms. Therefore, the configuration does not need any additional detection function for validations across the sub-configurations, and hence, $\mathfrak{Det}_{\text{PANDA}} = \mathfrak{Det}_{\text{HYBP}} \cup \mathfrak{Det}_{\text{SCHED}}$.

Modification Generating Functions – $\mathfrak{Mod}_{\text{PANDA}}$

The modification generating functions for the hybrid planning and scheduling configuration are all the functions of the sets $\mathfrak{Mod}_{\text{HYBP}}$ and $\mathfrak{Mod}_{\text{SCHED}}$ except for two minor alterations. First, the plan modifications for inserting new task expressions have been defined for the partial-order planning paradigm and the corresponding generator function definition is not aware of any resource manipulations and temporal annotations. Since the resource manipulation is syntactically embedded in the regular pre- and postcondition specification and since the temporal information is included in the plan separately via inference functions, we believe that it is not necessary to redefine $f^{mod}_{\texttt{InsertTask}}$ explicitly.

In order to properly refine the temporal structure of an abstract task by its implementation, as well as to keep resource allocations consistent during method applications, the plan modification for task expansion has to be redefined on the basis of the hybrid planning system configuration. The following definition therefore reuses the central parts of Def. 3.20.

Definition 3.40 (Expand Task – 2nd Redefinition)**.** Given a hybrid planning and scheduling domain model $D_{\text{PANDA}} = \langle \mathscr{M}, \Delta, \mathtt{T}, \mathtt{M} \rangle$, a plan $P = \langle TE, \prec, VC, CL, TC, RC \rangle$, and a flaw $\mathtt{f}$, the third version of the modification generating function $f^{mod}_{\texttt{ExpandTask}}$ proposes for *every* occurrence of a complex task schema instance in $\mathtt{f}$ a decomposition according to *every* appropriate method definition in $\mathtt{M}$ as follows.

$$\langle \underbrace{TE_x \cup \prec_x \cup VC_x \cup CL_x \cup TC_x \cup RC_x}_{\text{expansion network}} \cup \overbrace{\prec' \cup VC' \cup CL' \cup TC' \cup RC'}^{\text{new context}}, \underbrace{\{te\}}_{\text{complex task}} \cup \overbrace{\prec'' \cup CL'' \cup TC'' \cup RC''}^{\text{old context}} \rangle \in f^{mod}_{\texttt{ExpandTask}}(P, \mathtt{f}, D)$$

The expansion network is thereby substituting the complex task in consideration of its temporal and causal context. More formally, the plan modification consists of the following components:

1. $te \in TE \cap comp(\mathtt{f})$ and $te = l : t(\bar{v})$ with $t \in \mathscr{T}_c$
2. $\prec''$ is the complete subset of ordering constraints $\prec$ in which te occurs, that means, it either contains $te \prec te_P$ or $te_P \prec te$ for any $te_P \neq te$ in P
3. CL'' is the complete subset of causal links CL in which te occurs, that means, it either contains $te \xrightarrow{\varphi} te_P$ or $te_P \xrightarrow{\varphi} te$ for any φ and $te_P \neq te$ in P
4. $TC'' = RC'' = \emptyset$
5. $m_x = \langle t(\bar{v}), \langle TE_x, \prec_x, VC_x, CL_x, TC_x, RC_x \rangle \rangle \in \mathtt{M}$
6. $TE_x \cap TE = \emptyset$, $\prec_x \cap \prec = \emptyset$, $VC_x \cap VC = \emptyset$, $CL_x \cap CL = \emptyset$, $TC_x \cap TC = \emptyset$, and $RC_x \cap RC = \emptyset$

7. $\prec'$ is a set of new ordering constraints $te_x \prec te_P$ for every $te \prec te_P$ in P and $te_x \in TE_x$, respectively $te_P \prec te_x$ for every $te_P \prec te$ in P and $te_x \in TE_x$
8. VC' and CL' are sets of new variable constraints and causal links such that:
   a) For each incoming causal link $te_P \xrightarrow{\varphi} te \in CL''$, let $TE'_x \subseteq TE_x$ be a set of task expressions such that for any state s and valuation β that is compatible with $VC \cup VC_x \cup VC'$:

   $$s \models_{\mathscr{M},\beta} \Big(\bigwedge_{te_x \in TE'_x} \text{prec}(te_x) \Big) \Rightarrow \varphi$$

   TE'_x is thereby a minimal set in the sense that if any task expression is removed, the equation does not hold. Let furthermore φ_i be a formula for each te_i in TE'_x such that

   $$s \models_{\mathscr{M},\beta} (\text{prec}(te_i) \Rightarrow \varphi_i) \wedge (\varphi \Rightarrow \varphi_i)$$

   For each pair of plan steps te_{x_i} and formulae φ_i, the set CL'' includes a causal link $te_P \xrightarrow{\varphi_i} te_{x_i}$.
   b) For each outgoing causal link $te \xrightarrow{\varphi} te_P \in CL''$, let $TE''_x \subseteq TE_x$ be a set of task expressions such that for any state s and valuation β that is compatible with $VC \cup VC_x \cup VC'$:

   $$s \models_{\mathscr{M},\beta} \Big(\bigwedge_{te_x \in TE''_x} \text{post}(te_x) \Big) \Rightarrow \varphi$$

   TE''_x is thereby a minimal set in the sense that if any task expression is removed, the equation does not hold.[13] Let furthermore φ_i be a formula for each te_i in TE''_x such that

   $$s \models_{\mathscr{M},\beta} (\text{post}(te_i) \Rightarrow \varphi_i) \wedge (\varphi \Rightarrow \varphi_i)$$

   For each such pair of plan steps te_{x_i} and formulae φ_i, the set CL'' includes a causal link $te_{x_i} \xrightarrow{\varphi_i} te_P$.
9. TC' is a set containing the new temporal constraints $start_{te} = start_{te_i}$ and $end_{te} = end_{te_j}$ with te_i and te_j being task expressions in TE_x such that there is no $te_k \in TE_x$ for which $te_k \prec te_i$ or $te_j \prec te_k$ is consistent with TC_x and $\prec_x$. Furthermore, TC' includes temporal constraints $start_{te} \leq start_{te_l}$ and $end_{te_l} \leq end_{te}$ for every plan step $te_l \in TE_x$.
10. RC' is a set of new resource constraints such that
    a) For each consumable resource $\mathtt{r}$ that is manipulated by te, RC' contains the two constraints $in^{\mathtt{r}}_{te} = in^{\mathtt{r}}_{te_i}$ and $out^{\mathtt{r}}_{te} = out^{\mathtt{r}}_{te_j}$ with te_i and te_j being task expressions in TE_x that manipulate $\mathtt{r}$. Furthermore, there is no $te_k \in TE_x$ for which $te_k \prec te_i$ or $te_j \prec te_k$ is consistent with TC_x and $\prec_x$.
    b) For each non-consumable resource $\mathtt{r}$ that is allocated by te, RC' contains the constraint $alloc^{\mathtt{r}}_{te} = alloc^{\mathtt{r}}_{te_i}$. Plan step $te_i \in TE_x$ thereby allocates $\mathtt{r}$, too, and there is no te_k in TE_x such that $te_k \prec te_i$ is consistent with TC_x and $\prec_x$.

•

Some remarks on the above modification generator definition: Item 4 states that no temporal or resource constraints are removed from the plan. The rationale for this refinement implementation is to preserve the temporal context and profile commitments of the abstract plan step with respect to the surrounding plan structures so that these artefacts can serve as a reference for the expansion network. Furthermore, it guarantees a conservative constraint set development and is thus an argument for the function's soundness.

Regarding the new constraints in TC' and RC' (items 9 and 10), we would like to point out that their construction parallels the procedure that adapts those causal links in which the complex task occurred (item 8). Given the deletion of the complex task, the commitment on the more abstract level can only be properly restored in the more concrete expansion network by connecting the information that is stored in the artefact

[13] If the respective plan step is obtained from a primitive task schema, we may assume that post(te_x) denotes a formula that is generated by te_x.

constraints with the new plan steps in TE_x. The new temporal constraints in TC' become necessary because the insertion of the sub-tasks into the qualitative ordering alone is not able to reproduce[14] the precise distances that represent the time window for the deleted te. Item 9 therefore chooses suitable candidates for consistently inheriting the temporal information: The starting and ending time point is assigned to one of the tasks in the leading, respectively final stratum of the partial order, taking into account the temporal constraints in the network TC_x.

The analogous situation can be found for RC' because the constraints in RC_x cannot relate resource manipulations and allocations inside the expansion network with variables that are defined outside. In item 10, the modification generator therefore chooses for each resource an – according to the partial order and the temporal constraints – first plan step that may inherit the input level and a last plan step for the output level, respectively.

Note that with the extended definition 3.40 above, the modification generating function $f^{mod}_{\texttt{ExpandTask}}$ has to deal with a total of four combinatorial factors in constructing an expansion for a specific flawed complex task: The first two are those from the previous definition, that means, the choice of implementation method and the multiple possibilities for redistributing the causal links. The third dimension of complexity is the choice of the start and end time point assignments, and the fourth combinatorial factor is the choice involved in propagating the abstract resource levels. Since one plan modification has to be generated for each permutation, the $\mathfrak{C}_{\text{PANDA}}$ task expansion generator issues

$$\begin{aligned} |\text{methods for } te| \quad \cdot \quad & (|\text{input linkings}| \cdot |\text{output linkings}|) \cdot \\ & (|\text{start assignments}| \cdot |\text{end assignments}|) \cdot \\ & (|\text{input levels}| \cdot |\text{output levels}| \cdot |\text{assignment levels}|) \end{aligned}$$

many plan modifications per flawed abstract task. This product depends of course on the concrete domain model specification and is hence in a worst case scenario a major problem with respect to efficiency. However, our experience is that hybrid planning domain models (for example, see Sec. 5.2.2) tend to have few methods per abstract tasks and allow for only *very few* linking and resource assignment permutations. The reason for this lies in the nature of hierarchical modelling, which inclines to define abstractions via summarizing sequences of different actions rather than merging parallel threads of the same type of actions. That means, that a task implementation relatively seldom uses flexible domain features more than once, because these are typically *processed* (made true/false) by the task network and hence cannot appear more than once without steps undoing the previous effect – in other words, the linking permutations can be expected to be neglectable. The rigid features can be ignored for this argument, because they only occur in task preconditions and will have to work in every permutation, anyway. Regarding resource access refinements, it is also not very common to manipulate the (potentially) same resource in more than one thread of tasks. The dominant modelling style is to *split* the different state features or resources and to treat them in specialized activity chains. At the end of the day, in all somewhat reasonably modelled domains, the expansion combinatorics is always substantially lower than the degrees of freedom for synthesizing a plan from scratch.

Triggering Function α_{PANDA}

The triggering function for the hybrid planning and scheduling configuration implies the flaw - modification relationships of the embedded configurations; in addition it implements the cross-configuration dependencies. This concerns in particular the task expansion plan modifications, which naturally become key refine-

[14] The converse is however given and therefore this step subsumes the insertion of an embedding partial order in item 7 (cf. $\prec'$ specification in par. 6 of Def. 3.20).

ments for dealing with abstract resource manipulations in the extension of scheduling.

$$\alpha_{\text{PANDA}}(\mathbb{F}_x) = \begin{cases} \emptyset & \text{for } x \in \{\texttt{OrdIncons}, \texttt{VarIncons}, \\ & \quad \texttt{TempInconsistency}\} \\ \mathbb{M}_{\texttt{InsertTask}} \cup \mathbb{M}_{\texttt{ExpandTask}} & \text{for } x = \texttt{ResInconsistency} \\ \mathbb{M}_{\texttt{AddResConstr}} \cup \mathbb{M}_{\texttt{AddVarConstr}} & \text{for } x = \texttt{OpenVarBind} \\ \mathbb{M}_{\texttt{AddOrdConstr}} & \text{for } x = \texttt{UnordTask} \\ \mathbb{M}_{\texttt{InsertTask}} \cup \mathbb{M}_{\texttt{AddCLink}} \cup \mathbb{M}_{\texttt{ExpandTask}} & \text{for } x = \texttt{OpenPrec} \\ \mathbb{M}_{\texttt{AddVarConstr}} \cup \mathbb{M}_{\texttt{AddOrdConstr}} & \\ \quad \cup \mathbb{M}_{\texttt{ExpandTask}} \cup \mathbb{M}_{\texttt{AddResConstr}} & \text{for } x = \texttt{Threat} \\ \mathbb{M}_{\texttt{ExpandTask}} & \text{for } x = \texttt{AbstrTask} \\ \mathbb{M}_{\texttt{AddTempConstr}} & \text{for } x = \texttt{OpenTmpBind} \\ \mathbb{M}_{\texttt{AddResConstr}} \cup \mathbb{M}_{\texttt{AddVarConstr}} & \\ \quad \cup \mathbb{M}_{\texttt{AddOrdConstr}} \cup \mathbb{M}_{\texttt{ExpandTask}} & \text{for } x = \texttt{ResThreat} \end{cases}$$

The newly added resolution relationships are the following three: First, the inconsistency of resource constraints can now be resolved by the hybrid planning sub-configuration. The introduction of production steps can either be performed directly by task insertion or indirectly by expanding a participant. The latter may also release some resource by refining the consumption and allocation structure. The second new triggering relation is addressing resource threats by task expansion. This is the identical mechanism to causal threat handling in hybrid planning, that means, the more concrete resource manipulation structure may allow for a resource conform course of action by interleaving production and consumption sequences. Thirdly, causal threats are related to the addition of resource constraints. While it is natural to observe some of the non-consumable resource over-allocations on the causal level, there may also occur the extremely rare situations in which term queries may appear as interfering updates. The treatment of open preconditions is not delegated to the resource constraint handling because this would not add the necessary causal link. A conceptual parallel is however given.

Please note that, in the above triggering function, we do not include the possibility to add temporal constraints in those cases that suggest adding an ordering constraint. Enforcing the partial order ensures that the implicit quantitative temporal structure is restricted minimally with respect to the necessary amount of change.

Inference Functions – $\mathfrak{Inf}_{\text{PANDA}}$

The only inference function in the hybrid planning and scheduling configuration is responsible for an adequate implication of the causal structure on the quantitative temporal model. While the qualitative partial order on the abstract plan steps is not affected by causal links except for the consistency criterion that forbids contradicting orderings, the fine granular temporal model can derive the minimal commitment that a causal link induces on any two linked tasks.

Definition 3.41 (Causally Motivated Temporal Constraint Introduction)**.** For a given planning problem π and partial plan $P = \langle TE, \prec, VC, CL, TC, RC \rangle$, the inference function $f^{inf}_{\texttt{IntroCLinkTemp}}$ adds for any two causally linked plan steps a temporal constraint such that the (potentially complex) producer cannot start later than the (potentially complex) consumer ends.

For each causal link $te_i \xrightarrow{\varphi} te_j \in CL$ for which $start_{te_i} \leq end_{te_j}$ does not hold in the constraint set TC, the application of the inference function $f^{inf}_{\texttt{IntroCLinkTemp}}(P, \pi)$ returns the following plan modification: $\langle \{start_{te_i} \leq end_{te_j}\}, \emptyset \rangle$. •

In fact, the inference function proposes a negative ordering constraint that corresponds exactly to the consistency criterion. We note that this constraint is rather weak and that the inference is implied by the refinement mechanism that tracks causality, combined with the inference of partial-order planning. For the sake of clarity, we present it here, though.

Summary of the PANDA Configuration

$\mathfrak{C}_{\text{PANDA}}$, the system configuration for hybrid planning and scheduling, is composed of the following function sets:

$$\begin{aligned}\mathfrak{C}_{\text{PANDA}} = \langle\{&\overbrace{f^{det}_{\texttt{OrdIncons}}, f^{det}_{\texttt{VarIncons}}, f^{det}_{\texttt{OpenVarBind}}, f^{det}_{\texttt{UnordTask}}}^{\mathfrak{Det}_{\text{TAP}}}, \overbrace{f^{det}_{\texttt{OpenPrec}}, f^{det}_{\texttt{Threat}}}^{\mathfrak{Det}_{\text{POCLP}}}, \overbrace{f^{det}_{\texttt{AbstrTask}}}^{\mathfrak{Det}_{\text{HTNP}}},\\ &\overbrace{f^{det}_{\texttt{TempInconsistency}}, f^{det}_{\texttt{OpenTmpBind}}}^{\mathfrak{Det}_{\text{TTAP}}}, \overbrace{f^{det}_{\texttt{ResInconsistency}}, f^{det}_{\texttt{ResThreat}}}^{\mathfrak{Det}_{\text{RTAP}}}\},\\ \{&\overbrace{f^{mod}_{\texttt{AddOrdConstr}}, f^{mod}_{\texttt{AddVarConstr}}}^{\mathfrak{Mod}_{\text{TAP}}}, \overbrace{f^{mod}_{\texttt{InsertTask}}, f^{mod}_{\texttt{AddCLink}}}^{\mathfrak{Mod}_{\text{POCLP}}},\\ &\overbrace{f^{mod}_{\texttt{AddTempConstr}}}^{\mathfrak{Mod}_{\text{TTAP}}}, \overbrace{f^{mod}_{\texttt{AddResConstr}}}^{\mathfrak{Mod}_{\text{RTAP}}}, f^{mod}_{\texttt{ExpandTask}}\},\\ \{&\overbrace{f^{inf}_{\texttt{OrdConstraint}}}^{\mathfrak{Inf}_{\text{POCLP}}}, \overbrace{f^{inf}_{\texttt{IntroTempVar}}, f^{inf}_{\texttt{IntroTempDist}}, f^{inf}_{\texttt{IntroOrdering}},\\ &\overbrace{f^{inf}_{\texttt{IntroResVar}}, f^{inf}_{\texttt{IntroVarConstraint}}}^{\mathfrak{Inf}_{\text{RTAP}}}, f^{inf}_{\texttt{IntroCLinkTemp}}}^{\mathfrak{Inf}_{\text{TTAP}}}\},\\ &\mathfrak{Str}\rangle\end{aligned}$$

Theorem 3.15 (Properness of $\mathfrak{C}_{\text{PANDA}}$)**.** *$\mathfrak{C}_{\text{PANDA}}$ is a proper system configuration in the sense of Def. 3.2.*

Proof. With the inference function given in Def. 3.41 completeness of the detection function set $\mathfrak{Det}_{\text{PANDA}}$ follows directly from Theorems 3.7 and 3.13.

$\mathfrak{Mod}_{\text{PANDA}}$ is a semi-complete set of sound modification generating functions because the included subsets are, and in addition, the revised task expansion is a sound modification generator that returns every possible refinement.

The inference function set $\mathfrak{Inf}_{\text{PANDA}}$ consists of sound members and is therefore sound itself.

Finally, the correspondence of $\mathfrak{Mod}_{\text{PANDA}}$ and $\mathfrak{Det}_{\text{PANDA}}$ can be directly seen from the definition of the α_{PANDA} triggering function. □

Theorem 3.16 ($\mathfrak{C}_{\text{PANDA}}$ is Modification-Complete)**.** *$\mathfrak{C}_{\text{PANDA}}$ is modification-complete according to Def. 3.3.*

Proof. The proposition follows directly from Theorems 3.8 and 3.14. □

3.5 Discussion

3.5.1 General Comments

All system configuration presentations include a brief discussion of the relevant aspects in which they differ from the state of the art techniques, respectively which established methods are incorporated. This discussion section is dedicated to a number of inter-configuration topics and to some starting points for future developments.

The previous sections started with motivational scenarios that were intended to illustrate the specific requirements of planning-like applications. Our examples stemmed from the area of project planning in order to provide demonstrative problem settings; the depicted (and referenced) software systems therefore have to be viewed as example tools in the flavor of the respective configurations' type of problems. It has to be emphasized that these tools do *not* solve the described problems but give the human users an advanced visual access to some features of the problem such that they can solve the problems by themselves more easily. None of the commercially available software solutions supports a domain model representation that is comparable to any planning or scheduling system.

3.5.2 Hybrid Planning

It is a key attribute of our approach that its notion of abstraction intrinsically couples the abstraction of situations with the abstraction of (courses of) actions. The main advantage of this principle becomes apparent in those system combinations that integrate hierarchical planning aspects with other techniques: the integration effort is relatively small, because the abstraction mechanisms obviously apply transparently to all domain model entities, particularly in advanced configuration extensions. It has to be pointed out that the *central element* of the more sophisticated representations is the integrated, declarative treatment of *causality*.

Approaches like NONLIN [255], O-PLAN [258], SIPE [281], or UMCP [84], annotate causality handling "rules" in their decomposition method definitions. SHOP [200] goes one step further and substitutes causality by arbitrary conditions for validating method applicability. These procedural domain model constructs imply two very serious issues concerning what we may informally call *horizontal* and *vertical* domain model consistency. The horizontal consistency raises the question whether the decomposition method is applicable at all and whether it is compatible with the other methods that are defined for the same abstract task, that means, do they cover overlapping application situations, do they miss situations, and the like. Vertical consistent methods "match" in some sense the abstract action they are intended to implement, which is difficult to decide if no defined connection to the complex task exists.

In contrast to such a handling of causality, our hybrid planning configurations draw upon a well-found, semantically sound, and transparent method concept. The declarative domain model thereby allows for modular specifications in which causal relations can be validated during model construction as well as for analyzing and properly manipulating the causal context of abstract actions during the plan generation process.[15]

In the presented combination, situation abstraction and action decomposition do not only meaningfully complement one another, they also manage to partly compensate the drawbacks of hierarchical planning as they are discussed in the literature for the ABSTRIPS procedure [121]: For situation abstraction alone, we can easily construct problem specifications that cause a poor performance of abstraction planners, in a way that higher abstraction spaces contain no information about what caused backtracking at less abstract levels. Although Giunchiglia's arguments do hold for task abstraction based planning as well, it becomes considerably more difficult to construct the corresponding anomalies, for the state abstraction hierarchies are interwoven with task reduction schemata. In this way, action decomposition secures situation abstraction and vice versa.

[15] Our methodological roots in the partial-order causal-link planning paradigm stand, again, in the tradition of Greek philosophy when mapping all system dynamics on explicit causal structures: "Nothing occurs at random, but everything occurs for a reason and by necessity" (Democritus). "Some people question whether chance really exists. For nothing, they say, happens by chance, but there is a definite cause, other than chance, for everything which we say happens 'spontaneously´." (Aristotle, *Physics*)).

We can, of course, also construct domain models in which causal interdependencies are obfuscated by the decomposition hierarchy. In fact, Sec. 5.2.3 will introduce such a domain for the definition of benchmark problems for hybrid planning. But given a to some extend "reasonable" knowledge engineering phase, we can expect to produce only in this sense benign hybrid domain models.

There is however a fundamental problem that hybrid planning configurations have to face: the co-existence of task decomposition and plan step insertion necessarily induces a certain amount of redundancy in the refinement space. Since the plans that are created by applying decomposition schemata can also be synthesized from atomic actions, symmetric and isomorphic solutions become a major issue that has to be addressed by efficient search strategies. This is in particular relevant in situations where a troublesome plan synthesis can be cut short by an appropriate task decomposition that solves the issue as a side-effect of the task implementation. On the other hand, it may be computationally more efficient to synthesize an implementation variant rather than backtracking over the task expansion plan modification. Some authors also raise the question of "user intent" [149], that means, whether or not all of the tasks are intended by the modeler to be inserted by the system on demand. A similar question is that of premature task insertion as opposed to waiting until all task expansion options are explored (this is related to the balance of task insertion versus causal link insertion in POCL planning). Both problems are typically addressed by tagging task as "insertable" or by annotating conditions as "to be achieved by any means necessary". We agree that both techniques are clearly motivated from an end user perspective, however we believe that they do not fit too well into the refinement generation context but into the planning strategy instead. We understand this to be a question of solution preference that is outside the scope of this thesis.

We finally would like to point out that with the above considerations, we do not regard the insertion of new plan steps as a mandatory characteristic of hybrid planning. The hybrid character is already clearly given if causality is propagated into the abstract action levels and consulted in goal satisfaction and threat analysis. Since closing an open precondition is also handled by the insertion of causal links, a corresponding system configuration $\mathfrak{C}_{\text{HYBP}^*}$ keeps the properness and completeness results. This more HTN-like variant of hybrid planning will be used as our reference configuration in an empirical study (Chap. 6).

3.5.3 Integration of Planning and Scheduling

The integration of planning and scheduling functionality has always been a major topic in the field, since the need for an integrated treatment of in particular temporal aspects is evident in every practical application. In addition to our focused presentation in Sec. 1.2.3, it should be mentioned that this integration is in general classified into the categories of stratified, interleaved, and homogeneous planning and scheduling [239]. While the first category subsumes the typical combination architectures, in which a planning and a scheduling system work out their solutions separately (typically the result of one fed as input to the other), our approach and in particular the PANDA configuration is an example for the interleaved integration paradigm. That means, that the two subsystems' algorithms are synchronized "stepwise" at a certain level of granularity and are typically used as critics of each other. It is a thereby a unique feature of our approach that there is no definite separation of subsystems and due to their shared representation of the refinement space, synchronization occurs seamlessly and the system boundary is completely open. In fact, the planning and scheduling subsystems even share several components. Homogeneous architectures go one step further and apply the same calculus on the unified representation, for example, by formulating as a SAT or ILP problem.

The main advantage that we see in an interleaved integration is that we can employ a natural representation of plans and corresponding refinement spaces as well as flexibly decide on the appropriate reasoning method. By "natural" we mean manipulations to the human-readable plan data structure that can be intuitively comprehended by humans, as opposed to, for example, value assignments in a propositional formula set that may encode book-keeping artefacts such as decisions on plan step instantiations, and the like (cf. [151]). Furthermore, since we rely on a configuration design that uses a meta-CSP representation, we are able to incorporate any temporal, respectively resource model that appears to be appropriate for the application.

A major topic in the context of scheduling models is certainly that of parallel actions. In the TTAP configuration and its extensions, the partial order plan is regarded as an causality-centered abstraction of the

plan. Given their flaw detection functions, our systems produce what Knoblock calls the class of plans with actions that are *independent relative to a goal* [154]. In this sense, our CSP layers are free to decide about the concrete temporal position of unordered tasks, because goal establishment is treated on the causal level and any conflicting plan steps are ordered in the solution plan. However, (completely) *independent actions* are not supported, because there may be conflicting task effects that are not used for any goal establishment and are hence not ordered by the causal subsystem and remain as "parallel" plan steps. If this notion of independency is required, an additional class of causal threat flaw has to be issued that is implied to be solved by an ordering constraint.

An alternative is to build the formal framework on action semantics that take parallel action threads into consideration. A survey of relevant planning representations and reasoning techniques, including a characterization of existing approaches can be found in [170]. Relatively popular are formal specifications that are extensions of the situation calculus. Their means for expressing concurrency resembles our notion of complex tasks, for example by interleaving *compound* actions that are built from primitive action specifications [15]. This interleaving semantics is also used on a continuous time base and for describing natural processes, that means, physical phenomena, in planning models [220]. These representations are of course not only found in the situation calculus but also in frameworks that are built on causal theory [120] or that are mechanized in the ConGolog logic programming system [119]. Related techniques have also been introduced into partial-order planning as designated concurrency conditions [36]. Such a meta-goal for achieving state features *simultaneously* (or in any other specific temporal configuration) is not yet supported by our configurations.

But, again, since we are interested in a natural representation and do not depend upon providing an integrated calculus for our framework, we prefer constraint-based extensions to the causal plan. In doing so, we agree with authors like [31], who conclude

> Our results confirm that temporal reasoning is a sufficiently self-contained activity to be implemented entirely independently of the overlying application, modulo some assumptions about how problem-solving is to proceed. We have also shown that constraint-based temporal reasoning supports a "least-commitment" style of planning and scheduling that is efficacious in a wide variety of complex problem domains.

Of course, the most prominent representational issue of an integrated system is the employed temporal model, and since temporal reasoning is central to many application domains and kinds of problems, there is an endless number of approaches available (for a survey, see for instance [249, 276]). We decided in favor of the Simple Temporal Problem (STP) representation [71], a simpler version of the more general Temporal Constraint Satisfaction Problem, because it is a reasonable trade-off between expressivity and efficiency. It can not only be solved in cubic time by applying Floyd's well known all-pairs-shortest-paths algorithm, it can also be dealt with incrementally, for example, by retracting constraints and repropagating the network locally [51], which makes it a very efficient and practicable method for the application in a dynamic scenario (dynamic in the sense of recomputing the network during consecutive refinements). As we have noted in the domain model specifics section of the temporal planning configuration (Sec. 3.3.2), using STP as a foundation for temporal reasoning has been successfully demonstrated in a number of planning systems, for instance, in IxTeT [116, 163, 266], *parc*PLAN [81], the SIADEX HTN planner [42], the RAX Planner/Scheduler [142], and VHPOP [297].

It is also worth noting that we are not necessarily restricted to the convex STP representation. During the plan generation process, and in particular concerning abstract task refinements, *disjunctive* temporal constraints would be a very useful mechanism to explicitly represent ambiguity. For example, a task implementation may either take a very long time in the morning or may be done very quickly in the afternoon. This increased expressiveness demands of course a significantly more complex reasoning process, but on the other hand this additional effort stays manageable [235]. The so-called *disjunctive temporal graph* has been developed as an efficient reasoning technique in this context [109], which has been applied as a meta-CSP in a temporal partial-order planner [175] as well as an augmentation for a planning-graph based system [112].

A popular alternative for modelling temporal phenomena is to assign the concept of duration not to the actions but to the states. In this view, temporally extended actions are interpreted via states that persist for a

specified amount of time and then instantly change. While this model may offer some practical advantage regarding computational aspects, it is however not adequate with respect to our semantics, because it would imply that during a state the interpretation of the time constant (continually) changes.

Reasoning about time can be based on two temporal primitives, namely *instants* (points of time, which is our representation) and *periods* (intervals of time). The latter is historically of particular interest to the field of AI planning, because the interval algebra introduced by Allen [5] is a well understood calculus for qualitative temporal reasoning, which suggested itself for extending the classical planning approaches.

Allen's approach to reasoning about time is based on the notion of time intervals and binary relations on them. Given two concrete time intervals, defined via real numbers denoting their end points, their relative positions can be described by exactly one of thirteen atomic interval relations, for example, equivalence, precedence, succession, strict and partial inclusion, and the like. From these atomic interval relations, 2^{13} possible union sets can be constructed, which form the set of binary interval relations. This set forms together with the operations intersection, relational converse, and relational composition the mentioned interval algebra. In this algebra, we may construct formulae like the composition of "interval *A* precedes interval *B*' ' and "*B* strictly includes interval *C*", which implies the formula that "*A* precedes *C*".

A qualitative description of an interval configuration is usually given as a set of formulae of the above form, or, equivalently, as a temporal constraint graph with nodes as intervals and constraint arcs labeled with interval relations. The reasoning problem is to decide whether the configuration description is satisfiable, that means, whether there exists an assignment of real numbers to all interval endpoints that are consistent with the constraints; this problem is known to be NP-complete. The most often used method to determine this kind of satisfiability is the path-consistency method [5] in which repeatedly the compositional closure of the binary relations is computed. If that closure contains an empty binary relation, the configuration is obviously not satisfiable; the converse implication is not valid, however.

However, there exist subsets of the interval relations such that if the configuration descriptions are restricted to this subset, the reasoning problem's complexity level drops down to a polynomial one. These subsets are known as the continuous endpoint class, the pointizable class [272], and the ORD-Horn class [205]. These classes do not only lead to a gain in performance but lead also to completeness of the efficient path-consistency method [203].

Concerning the temporal action model, we employ a view on temporal reasoning that employs a subsystem in the fashion of a black box: Time change happens at the end of the durative interval. This kind of interpretation of temporal information can be found in the well-known temporal GRAPHPLAN extension TGP [241] and many more. We believe that we have motivated this semantic design choice sufficiently, however would like to discuss the alternatives briefly.

Many researchers were unsatisfied with the idea of actions that are somewhat "undefined" during the interval of their execution. This is true to some extent: According to our semantics, there is no possibility to evaluate the interpretation of any state feature between two states, because that would imply the existence of a third, an intermediate state and hence contradict the notion of actions as atomic state transitions. However, on second thought, this is not an issue because first, the temporal information is interpreted outside the state transition model and second, we can build on these atomic transitions and formulate abstract ones, for which the concept of "intermediate" steps naturally exists. The presence of abstract actions and their corresponding implementations gives us the means to model a variety of phenomena without the restrictions of non-hierarchical planning. Furthermore, we would like to note that we are inclined to believe that perceiving atomicity as a deficiency tends to indicate a misconception of the semantic basis of planning domain models such that formal properties are confused with issues that arise when constructing appropriate representations of the application domain.

The following quotation taken from the temporal PDDL specification [99] supports our claim:

> A simplistic way of ensuring correct action application is to prevent concurrent actions that refer to the same facts, but this excludes many intuitively valid plans.

In order to refine the granularity of temporal information about an atomic action, more "sophisticated" approaches allow preconditions to be annotated with time points or intervals, so that the requirement that a condition be true at some point, respectively over some interval, within the duration of the action can be expressed [72]. Effects can be treated analogously (*delayed* effects [11]) and it thereby becomes "possible to distinguish between conditions that are local to specific points in the duration of an action and those that are invariant throughout the action" [99]. The incoherency that is introduced by the coexistence of a notion of an atomic actions as singular state transitions together with the technique of splitting up actions along the time line (cf. PDDL action triples discussed in introduction of Sec. 3.3.2) does not only lead to some confusion among the researchers (cf. discussion in Sec. 2.8.1) but also to artefacts regarding the encoding of domain models: "However, invariants cannot be specified because the preconditions are checked at the instant of application and subsequent delayed effects are separated from the action which initiated them" (ibid.).

Concluding the discussion about representation alternatives, it has to be emphasized that in principle, any sophisticated temporal or resource model could be deployed as a substitute for the current ones. The substitutes may range from any of the previously mentioned representational frameworks to functional dependencies of task parameters or even fuzzy temporal intervals [45] and probabilistic models [25]. The only requirement of the refinement planning paradigm is that the deployed technology complies with constructive search, that means in particular, that it is not based on local search like in repair-based scheduling, etc. If we are furthermore interested in maintaining completeness (see discussion in Sec. 2.8.5), the generated modification proposals have to cover the option range completely and they preferably have to do so in a compact manner. The presented configuration is therefore focused on narrowing down the respective intervals in order to separate inconsistent assignments as early as possible, complemented by a mechanism that blindly guesses values in the (probably consistent) intervals.

This discussion solely focuses on representational issues and does not consider another dimension of temporal planning that became a major topic in the context of the planning competitions: Is the reasoning method capable of finding a solution – effectively or efficiently – for *really* temporal planning problems [67, 69]. Since the authors' arguments for classifying the temporal problems are formulated in terms of the influence that the temporal dimension has on the solution quality (as opposed to problems in which the temporal information is more or less "syntactic sugar"), we subsume this question in the general optimization discussion below (Sec. 3.5.5).

3.5.4 Hybrid Planning and Scheduling

Although the successfully fielded planning systems are dominantly hierarchical ones (cf. introduction p. 13) the integration of time and resources rarely includes the notion of abstraction corresponding to the one that is introduced by the action hierarchies. We believe that this is due to the weak semantic foundation for grounding abstractions; practically all hierarchical systems restrict their view on complex tasks and their implementations to the notion of procedures or macros, which, like in a simple programming language, allow for organizing code or for expressing recursive structures.

As a consequence, those approaches that may qualify as related to our PANDA configuration solely focus on integrating temporal information on the higher abstraction levels. An abstract task is thereby understood as the temporal wrapper structure that under-estimates lower bounds and over-estimates upper bounds [45].

The approach presented in [61] propagates bound estimators, so-called "summarized information", from primitive action specifications up the decomposition hierarchy. This methodology could be combined with the presented user specified *approximation* abstraction, for example in those cases in which no approximated resource usage is explicitly specified. We note that the method of summarized information relies on completely specified models for "classical" hierarchical task network planning in which no additional tasks appear like they do in hybrid planning.

Our concept of relating task abstraction and approximated resource manipulation can be found in the notion of abstraction as it is described in [68]: These authors also understand abstraction as a kind of precise discretization. The contrary, non-declarative approach is realized in the SHOP2 system: Durative actions can be

modelled via manually specified updates of a clock object [123]. A specific translation schema is then formulated to simulate the respective PDDL temporal triples. There is however no further semantic connection between the execution intervals on the different levels of abstraction, which makes, as the authors say, "dealing with the temporal variables in larger problems [...] somewhat cumbersome".

Completely orthogonal to our abstraction representations and reasoning methods is the notion of hierarchical resources in the temporal planning system IXTET. In this approach, a resource is regarded to be more "important" than another, that means, it is located above the other in the hierarchy, if the manipulation of one implies the manipulation of the other. The hierarchy of importance thereby implies the order in which the various resources should be addressed by the reasoning process. This implicit dependency of resources is induced by the action schemata and the corresponding hierarchy can be constructed dynamically, based on condition analysis in the current partial plan [104]. This technique proved to be extremely efficient and since it is perfectly compatible with our formal framework semantics on the one hand, and with our view on independent refinement generators on the other, it will be considered as an additional strategic advice in future implementations of our proposed approach (in particular $\mathfrak{C}_{\text{RTAP}}$ and its extensions).

The presentation of the respective sections is loosely based on our previous work, in which we developed the idea of resource hierarchies [232] and the integration of hierarchical planning and scheduling [233].

3.5.5 Perspective

Our perspective on future developments can be roughly divided into two threads: extending the currently defined system configurations and creating new configuration concepts.

Configuration Enhancements

While the reasoning algorithms are completely robust with respect to implicit information stored in the various constraint sets, the human user is typically not. In order to make the (intermediate) results of the actual configuration implementations easier to read and to comprehend for a human user (cf. future work Sec. 7.2.2), we propose two inference functions for simplifying the variable and ordering constraint sets.

We do not want to detail these inference mechanisms but only briefly present the idea: A constraint is subsumed by other constraints if it does not contribute to a restriction of variable values. For instance, once a variable is assigned a constant value, the variable is substituted in all constraints by the respective constant and all co-typings and non-cotypings are removed from the constraint set. Finally, the trivial inequations (different rigid constant values denote different objects) are removed as well. Another example is co-typing, which implies the removal of all co-typings that refer to a super-sort. In this way, a canonical representation of the constraints can be reached that offers an easy access to the inferential closure of the constraint set and that is substantially smaller. The same principle can be applied to ordering constraints as well.

A similar functionality may also be useful for the temporal and resource constraints. Such an inference would not only clear the subsumed constraints but would also remove artefact variables, if possible, since these have no correspondence in the partial plan steps and are therefore hard to explain to non-specialist users.

Concerning partial-order causal-link planning and all its extensions, the insertion of plan steps raises the question of how to control the addition of plan steps. Most of the modern approaches are based on techniques that focus on plans of a given length and are therefore producing solutions with an optimal number of plan steps. Planning by search in the space of partial plans does however face the problem of recursion, that means, the more or less same plan step is to be added repeatedly, and it has to be decided whether this chain is developed purposefully or not [240]. A prominent example for reasonable recursion is travelling: moving from point a to point b may require to visit some point c in between, which may in turn

only be reachable from d, and so forth. An example for a recursion that has to be cut off is opening a door that has to be closed in order to be opened and that has to be opened in order to be closed; this is a futile development that will either turn out to fail in an inconsistent refinement or lead to an infinite[16] plan development. Although a form of recursion detection may be implemented by a corresponding detection function, the proper handling of this flaw has to be addressed by suitable planning strategies (see the following chapter 4).

Giving scheduling a more active role in the system configurations will be a high priority topic in future research. Building optimizing system configurations is not only a matter of search strategies but also includes providing suitable refinement options, in this case optimality-driven interval reductions for the temporal and resource constraints. Although the presented configurations do not take optimality aspects into account, we are convinced that only small, local changes to the modification generating functions of the $\mathfrak{C}_{\text{TTAP}}$ and $\mathfrak{C}_{\text{RTAP}}$ configurations can realize a refinement generation process that is "biased" towards more valuable solutions. To this end, our next developments will try to integrate the above described techniques from IXTET resource hierarchies and time-optimal plan generation heuristics, for example generating a relaxed planning graph in the fashion of [105, 106][17]. This will be probably also involve giving up completeness of the refinement options and is therefore beyond the scope of this thesis.

The last enhancement concerns the temporal representation for which it would be worthwhile to develop abstraction mechanisms like for other resources. Reasonable candidates would be approximation and qualification for dealing with different levels of temporal granularity on different levels of abstraction. An example would be approximating concrete time points by discrete hour slots or qualifying a time interval by "in the morning". Relative to the granularity level, different thresholds could be defined for discriminating temporal phenomena, for example all abstract tasks would happen simultaneously on Monday morning, but sequentially every full hour on the primitive level.

Dynamic Configurations

This chapter has presented a number of system configurations and discussed numerous deviations from and optional components of these configurations, including the strategy that has been treated as a fixed but exchangeable entity. In particular in the resource and time aware configurations there will co-exist a number of alternative configuration components that may address special cases of problem specifications or that may provide very efficient solutions to specific (sub-) problems. Examples are efficient solvers for particular problem classes, for example, modification generators that are able to compute very time-efficient travel plans.

These specialist configuration components lead to the idea of *dynamic configurations*, that means, configurations in which components are altered during the run time of the refinement planning algorithm. On the one hand, these may be problem-specific configurations like the ones described above, on the other hand, these may be progress-specific configurations: The beginning of the plan generation episode deploys a different set of configuration components than the system requires when it comes closer to a solution. For example, constant assignments are typically not adequate to start with and should be therefore "switched off" until the plan reached a certain level of maturity.

With our definitions of sound configurations, realizing a dynamic configuration change is more a question of how to initialize the change. We could define soundness criteria for performing a change, for example, at least a criterion like "sound configurations can always be changed into sound extensions". More general, every type of change has to guarantee that the structure of the refinement space is never compromised. This may include associating refinements with configurations, such that the partial plans in all kinds of backtracking scenarios are coherently treated by the configuration that operates on the respective branch in the search tree.

[16] Since we deal with natural models only, infinite is here meant in the sense of a plan development that finally fails due to memory limits or because all task expression label symbols have been used.

[17] This approach to time-optimal planning, that means, generating plans with minimal duration, modifies the GRAPHPLAN representation of action and fact levels such that time points are assigned to the levels at which these levels happen, respectively are observable. Since GRAPHPLAN is able to produce maximum parallel plans, the duration of the solutions is minimal.

One major benefit of dynamic configurations is finally the realization of refinement-based hybrid plan-repair in this formal framework. Sec. 7.2.1 describes how a plan is first developed by a regular configuration, then disassembled by a configuration that isolates execution failures, and finally reconstructed by a regular configuration with a strategic component that tries to restore as much of the previous commitment as possible.

3.6 Summary and Conclusion

This discussion concludes the chapter on system configurations, the incarnations of the formal framework that we have developed in Chap. 2, which make a generic refinement planning algorithm operational and eventually define the structure of the corresponding software artefacts. We have presented the concepts for building meaningful combinations of flaw detectors, refinement option generators, auxiliary inferences, and strategy components, that means, for building *proper* and *modification-complete* system configurations. They are the foundations for an extremely modular system setup, in which on the one hand functionality can be easily added in a plug-and-play fashion, but that is on the other hand well-defined and guarantees that the semantic integrity of the explored refinement space is never jeopardized.

An important result of our theoretical considerations about system configurations is the idea of *extending* configurations, that means, systematically enhancing the functionality of configurations. Based on this extension concept, we have developed a hierarchy of configurations, in which simple base configurations are progressively extended into implementations of our refinement planning framework that perform standard partial-order planning, hierarchical task network planning, and scheduling.

The standard configurations are first and foremost reference realizations of well-known functionality. Thereby, they demonstrate the flexibility of our approach, which is able to bridge the gap between the major disciplines in the field. But we have also gone one step forward and presented extensions of the standard implementations that produce *hybrid* systems. The configurations for *hybrid planning* and *hybrid planning and scheduling*, the main results of this chapter, combine hierarchical and non-hierarchical methods, combine causality-oriented planning and resource-focused scheduling in a seamless and well-founded way like it has never been done before. Within one unified representation, in terms of the domain model entities and the refinement space operations, the system architecture and its strategies are free to opportunistically develop plans (i) across different levels of abstractions as well as (ii) switching between the planning and the scheduling paradigm.

We discussed for each configuration its typical application scenarios, its domain model specifics, and all relevant reasoning issues. In particular, we would like to point out that in the final PANDA configuration, a cross-breed between hybrid planning and scheduling, we introduced the novel concept of abstract resources and the corresponding hierarchical scheduling methods.

Now that we have developed the mechanics for addressing different kinds of planning problems in our refinement planning framework, the following chapter focuses on search control in the refinement space. It provides us with the strategic support to effectively utilize the system configurations. We will revisit the hybrid planning configuration $\mathfrak{C}_{\text{HYBP}}$ later in Chap. 6 where it will serve as a benchmark configuration for some of our planning strategies in an empirical evaluation.

4 Planning Strategies – Search Control in the Refinement Planning Framework

THIS chapter focuses on a systematic introduction of practical realizations of the selection functions concept in the refinement planning framework (Sec. 2.6.4). It will present a broad assortment of planning strategies, ranging from adaptations of established work in the field to completely novel and powerful techniques. It is not only intended to demonstrate the individual procedures but also to detail their theoretical basis and their application perspective. A special emphasis is also given to provide deployment guidelines for strategies in general. Please keep in mind that all presented strategies are, by their design, applicable in all previously presented system configurations.

As it has been introduced in the previous sections, our framework does not employ one single strategy entity but it divides the sphere of responsibility into the selection of the refinement options and into the selection of the path that is to be followed in the refinement space generated by the options. In this chapter, we want to go one step further into a modular and systematic construction of planning strategies – in contrast to many approaches in which "the strategy" appears as a concealed oracle-like component that encapsulates its computations and finally reaches a decision. Note that this decision has to be proven with hindsight to find a solution in a structured and valid way. Our framework design explicitly answers these questions beforehand and developers can concentrate on the search heuristics in isolation.

The modular composition of the refinement planning framework is also reflected in the definition of planning strategy functions. Modification and plan selection functions impose a partial order on the respective input options and hence these functions are suited for a sequenced arrangement. Subsequent functions can be utilized to modulate preceding decisions by transferring all consistent additional orderings, thereby reinforcing the accumulated partial order. In other words, if the primary strategy does not prefer one option over the other, the secondary strategy is followed, and so on, until finally a random preference is assumed[1] by the linearization function of the refinement planning algorithm (Alg. 2.2, lines 22 and 24). Figure 4.1 shows an example sequential composition for three selection functions. The application of the first function divides the presented options in three equivalence classes. Within these equivalence classes f_1 is undecided, and in the example the subsequent selection f_2 is able to subdivide the first and third equivalence class, and the finally consulted third strategy can contribute with two more partitions. More formally, the sequential composition of modification and plan selection functions is defined as follows:

Definition 4.1 (Sequential Composition of Strategies)**.** Let f_a^{modSel} and f_b^{modSel} be two modification selection functions. The function obtained by composing them sequentially, denoted by $f_a^{modSel} \triangleright f_b^{modSel}$, is a modification selection function such that $\mathtt{m}_i < \mathtt{m}_j \in f_a^{modSel} \triangleright f_b^{modSel}$ if and only if

[1] It is an interesting fact that retaining a specific order of processing flaws or modifications (LIFO vs. FIFO, etc.) has been discussed frequently, for example in [218,236] while our approach does not only include a random element but in addition does not necessarily retain the order of plans in the fringe (and that roughly corresponds to the discussed issues). The reason for this decision is that we agree with authors like Williamson and Hanks [288], who pointed out that the planning procedure becomes highly sensitive to the order in which operator preconditions are specified in the domain description, and the like.

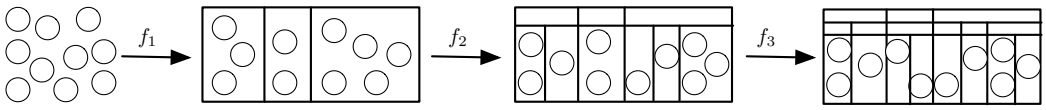


Figure 4.1: The sequential composition of strategic selection functions.

1. $\mathtt{m}_i < \mathtt{m}_j \in f_a^{modSel}$ or
2. $\mathtt{m}_j < \mathtt{m}_i \notin f_a^{modSel}$ and $\mathtt{m}_i < \mathtt{m}_j \in f_b^{modSel}$

In a sequential composition $f_1^{modSel} \triangleright \ldots \triangleright f_n^{modSel}$ we call f_1^{modSel} the *primary strategy* and the others $f_2^{modSel} \triangleright \ldots \triangleright f_n^{modSel}$ the *subsequent strategies*.

Sequential compositions of plan selection functions are built accordingly. •

From the definition of sequential composition it follows directly that a composite strategy can be deployed in a system configuration without any change to the refinement planning algorithm.

The following sections present in this sense a "catalogue" of planning strategies that have been realized so far in our refinement planning framework. All of them are single-objective heuristics and are intended to constitute the building blocks for more sophisticated strategies in (sequential) compositions. The presentation is organized according to selection principles; therefore, modification and plan selection functions are treated in alternation.

4.1 Strategy Components

The component catalogue begins with the classical strategy principles of un-informed search and simple plan evaluation heuristics. We proceed with strategy components that operate on the concrete flaw and modification situation of plans and show how such functions can be generalized from inflexible partial-order planning tactics to flexible general-purpose strategies. The section concludes with a presentation of our latest developments, the so-called HotSpot and HotZone technology. Both are universally successfully applicable heuristics for the refinement planning algorithm.

4.1.1 Search-Space-Based Strategies

One principle of selecting plans that suggests itself is to choose the first or last plan in the fringe. The two corresponding plan-selection functions are called $f_{First}^{planSel}$ and $f_{Last}^{planSel}$, which are the usual un-informed **depth-first** and **breadth-first** search schemata. They are defined as follows:

$$P_i < P_j \in f_{First}^{planSel}(P_1, \ldots, P_n) \text{ for } 1 \leq i < j \leq n \tag{4.1}$$

and

$$P_i < P_j \in f_{Last}^{planSel}(P_1, \ldots, P_n) \text{ if } i = n \text{ or } 1 \leq i < j < n \tag{4.2}$$

Both function definitions preserve the ordering of the fringe as it is induced by the respective modification selection function. Note that no subsequent selection can contribute because the plans are returned completely ordered. Deploying any of the two strategies consequently shifts search control completely towards the modification selection.

An alternative plan selection with a flavour of depth-first behaviour is the $f_{LongerHistoryFirst}^{planSel}$ function. This selection function prefers plans that have been obtained by a longer sequence of modification applications. It thereby considers not only refinements issued by modification generating functions but also inference steps. The rationale for such a selection function is to focus on plans that are more elaborated than others and it is typically used as a subsequent component in a composed strategy. How to realize the inverse function, that means, avoiding long refinement-paths and thereby adding a "breadth-first" flavour to a strategy composition, will be discussed below in Section 4.1.6.

It is worth noting that if the depth-first style schemata should fail to produce a solution in a reasonable amount of time, they can easily be modified into an iterative-deepening search strategy [157]. All that the strategies $f_{First}^{planSel}$ and $f_{LongerHistoryFirst}^{planSel}$ need is an additional parameter representing the maximal depth that is taken into account when the favoured plan is determined.

Many metrics have been proposed in the literature that are estimating the maturity of a partial plan. The obvious components for computing a numerical value out of a plan's symbolic representation is the size of the constraint sets and in particular the number of plan steps (since introducing an action implies "planning effort"). A very simple but effective plan-selection function that represents such a plan metric is the following definition:

$$P_i < P_j \in f^{planSel}_{ConstrPlans}(P_1,\ldots,P_n) \text{ if } \frac{|\prec_i|+|VC_i|+|CL_i|}{|TE_i|} > \frac{|\prec_j|+|VC_j|+|CL_j|}{|TE_j|} \tag{4.3}$$

The idea behind this plan selection function is to focus on plans that are more constrained than others and that are therefore more likely to either get completed or turn out a failure.[2] Since in general such processed plans tend to get processed again, it is more of a depth-first search but it is less vulnerable to getting trapped in useless plan refinement paths of repeated task-insertions (for solely inserting tasks reduces the heuristic value). Controlling the size of *TE* is in general an important factor when dealing with configurations that allow the insertion of new tasks. It has to be noted that there exists a large number of similar metrics that develop a set of weights for the different plan components, for instance, factors for emphasizing the influence of causal links, and the like. Conducting experiments with suchlike parameter families in which suitably balanced values are to be determined is however out of the scope of this thesis.

4.1.2 Flaw- and Modification-Based Strategies

All plan-generation algorithms that operate on the plan space have to deal with the notions of plan deficiencies and refinements. Although none of them does so in an explicit way like in our approach, these systems nonetheless assess the current flaw situation and seek to perform suitable plan modifications – the flaw assessment and refinement selection is however embedded (sometimes even deeply buried) inside the algorithmic procedure. As a consequence, thereby implicitly defined strategies adhere to a systematic way of developing plans that we refer to as *inflexible planning strategy*: following a selection schema that is solely based on a pre-defined strict preference on implicitly given plan-deficiencies and refinements. In our framework, inflexible planning strategies translate into preference relations on flaws and plan modifications, respectively on the plans that impose particular constraints on flaw detection and modification generation results.

A traditional form of modification selection is either to prefer or to disfavor categorically specific classes of plan modifications (see also discussion in Sec. 4.2). For example, one can prefer the expansion of complex tasks over the insertion of tasks or one tries to avoid the assignment of variables to constants "as long as possible". System implementations with built-in strategies of the previously described category are typically equipped with hand-tailored heuristics such that certain plan properties imply a modulation of the preference schema (for instance, the presence of a causal threat over-rules the preference of task expansion), but for the time being we focus on singular, un-modulated modification-based forms of decisions.

In the presented framework, a preference of a modification class can be encoded as follows: Let $\mathbb{M}_P$ be the modification class that is to be preferred by a modification-selection function, then the corresponding strategy $f^{modSel}_{Pref-\mathbb{M}_P}$ is defined as follows:

$$\mathtt{m}_i < \mathtt{m}_j \in f^{modSel}_{Pref-\mathbb{M}_P}(P,\{\mathtt{f}_1,\ldots,\mathtt{f}_m\},\{\mathtt{m}_1,\ldots,\mathtt{m}_n\}) \text{ if } \mathtt{m}_i \in \mathbb{M}_P \text{ and } \mathtt{m}_j \notin \mathbb{M}_P \tag{4.4}$$

for all plans $P \in \mathscr{P}$, sets of flaws $\mathtt{f}_1,\ldots,\mathtt{f}_m \in \mathbb{F}$, and sets of plan modifications $\mathtt{m}_1,\ldots,\mathtt{m}_n \in \mathbb{M}$. The inversely defined function, that means, avoiding the selection (and consequently the application) of instances of particular modification classes, appears to be useful in some strategy combinations. Please note that $\mathbb{M}_P$ can also be specified as the union of more than one base class, for example $\mathbb{M}_P = \mathbb{M}_{P_a} \cup \mathbb{M}_{P_b}$.

[2] This procedure is resembles the Davis-Putnam procedure (DPLL), which has been originally proposed for solving satisfiability problems in propositional logic. It became an established technique for solving constraint satisfaction problems because it gains considerable performance through the computationally simple heuristic of dealing with the most constrained variables first.

A modification class preference can also be reflected in the plan selection function. This realizes a plan-space navigation that is based on the availability of preferred refinement options for a plan. Please note that for the simplicity of presentation, algorithm 2.2 performs three *consecutive* sections: a first loop for the flaw detection, a second one for the modification generation, and finally a third loop in which the strategic choices happen. In the actual framework implementation, after applying the selected plan modifications to the current plan (line 20), the flaws and modifications are calculated for every new plan in the fringe *in advance*. That means, that the detected flaws and available plan modifications are accessible at the strategies' computation time. As a consequence, the following plan selection function can be defined for expressing a preference on a specific modification class $\mathbb{M}_P$:

$$P_i < P_j \in f^{planSel}_{Pref-\mathbb{M}_P}(\{P_1,\ldots,P_n\}) \text{ if } |mods(P_i) \cap \mathbb{M}_P| > |mods(P_j) \cap \mathbb{M}_P| \tag{4.5}$$

The function $mods : \mathscr{P} \rightarrow 2^{\mathbb{M}}$ is thereby used to retrieve all plan modifications that have been published for a given plan in the respective modification generation phase of the algorithm.

The previous plan-selection function can be re-formulated alternatively in terms of a relative measure, in contrast to an absolute number of refinement options. One of our defined measures calculates the proportion of "desired" plan modifications of type $\mathbb{M}_P$ against un-desired ones.

$$P_i < P_j \in f^{planSel}_{Pref-\mathbb{M}_P/\mathbb{M}}(\{P_1,\ldots,P_n\}) \text{ if } \frac{|mods(P_i) \cap \mathbb{M}_P|}{|mods(P_i)|} > \frac{|mods(P_j) \cap \mathbb{M}_P|}{|mods(P_j)|} \tag{4.6}$$

There exist subtle particularities that have to be taken into account when applying the above absolute and relative modification measures to plan-space navigation. On superficial examination, both types of strategies differ substantially: While any relative measure prefers to maneuver in the direction of plans for which supported refinements dominate, absolute selection functions favor plans with a large number of specific refinement options, notwithstanding that these alternatives may constitute only a minority of the available options. Which effect is the desired one depends on subsequent strategic components, that is to say, the next modification selection or the subsequent plan selection: if a plan is dominated by a class of plan modifications, other choices of refinement get to be *restricted* and it becomes more probable that one of the preferred options is eventually selected. The absolute measure on the other hand seeks to *broaden* the choice for subsequent decisions, because it opts for a larger and hopefully properly weighted collection of options. It is worth noting at this point that although an absolute modification type preferring measure inherently produces large sets of successor plans, subsequent strategies may profit from the broader data base for their performance.

In the above view, even though both measure types are not precisely antithetic, they pursue opposite lines. A closer examination however reveals that both types of measure exhibit an identical and particularly unpleasant long-term behavior: modification-type preferring plan selection functions intrinsically postpone well-developed plans, especially nearly-solutions. With flaws being resolved, their associated plan modifications are not published anymore. The successors of the selected plan thereby become less attractive for the selection function in subsequent decision points until finally no plan in the fringe features any modification of the preferred type. At this point, however, the described functions offer no strategic advice and become irrelevant with respect to finding a solution. It has to be noted that even inverse definitions of these strategies are dead ends: Avoiding plan modifications leads to a situation where all plans in the fringe are a collection of not-yet solutions to which a number of plan modifications (of the avoided class) is still to be applied – given that the refused modifications do contribute to a solution. At this point the planning algorithm is deserted by the strategic advice and has to select randomly.

For the reasons mentioned above, the selection schemata degenerate into a breadth-first type of search, thus exhaustively traverses the refinement space; any modification-based plan selection function (alone) will therefore not contribute strategically in a reasonable way.

Another relative measure relates the number of preferred plan modifications to the plan size, that is to say, to the number of plan steps.

$$P_i < P_j \in f^{planSel}_{Pref-\mathbb{M}_P/TE}(\{P_1,\ldots,P_n\}) \text{ if } \frac{|mods(P_i) \cap \mathbb{M}_P|}{|TE_i|} > \frac{|mods(P_j) \cap \mathbb{M}_P|}{|TE_j|} \tag{4.7}$$

Given the arguments in the previous paragraphs, this variation is also afflicted with the problem that it may traverse the plan space towards a point of indifference. But the normalization terms are worth a second thought: if we assume that the domain model is concise, then the size of the set of task expressions is non-decreasing on any path in the refinement space.[3] If we furthermore assume that in the long run the number of plan modifications decreases, not to speak of the preferred modification class instances, then the normalized measure will necessarily decrease for more developed plans. These considerations indicate that an inverted preference may be a reasonable subsequent strategy.

Strategies that concentrate on the flaw situation rather than on the available refinement options are considered to be more "problem-oriented". They are typically found in agenda-based planning algorithms, for instance the O-PLAN architecture [65], the goal ordering in the IPP planner [160], or the flaw handling in IXTET [163]. These systems have in common that they collect the plan defects and then decide which one to tackle first. Let $\mathbb{F}_P$ be the preferred class of flaws and let $modsFor : \mathbb{F} \times \mathscr{P} \rightarrow 2^{\mathbb{M}}$ be the function that returns all modifications that have been answered to a given flaw in a given plan. A modification-selection strategy-function for preferring refinements that address flaws of class $\mathbb{F}_P$ is then defined as:

$$\begin{aligned} \mathtt{m}_i < \mathtt{m}_j \in f^{modSel}_{Addr-\mathbb{F}_P}(P, \{\mathtt{f}_1, \ldots, \mathtt{f}_m\}, \{\mathtt{m}_1, \ldots, \mathtt{m}_n\}) \\ \text{if } \mathtt{m}_i \in modsFor(\mathtt{f}_a, P),\ \mathtt{m}_j \in modsFor(\mathtt{f}_b, P) \text{ and } \mathtt{f}_a \in \mathbb{F}_P,\ \mathtt{f}_b \notin \mathbb{F}_P \end{aligned} \tag{4.8}$$

It is again to be noted that $\mathbb{F}_P$ can also be specified as the union of more than one base class, for example $\mathbb{F}_P = \mathbb{F}_{P_a} \cup \mathbb{F}_{P_b}$.

Lifting the flaw-orientation principle to the selection of plans provides system configurations with an estimation of the difficulty of making the current plan a solution in terms of the number of plan modifications that will have to be applied. The definition of a plan selection function that prefers those plans that are flawed by many instances of a given class $\mathbb{F}_P$ is the following:

$$P_i < P_j \in f^{planSel}_{Addr-\mathbb{F}_P}(\{P_1, \ldots, P_n\}) \text{ if } |flaws(P_i) \cap \mathbb{F}_P| > |flaws(P_j) \cap \mathbb{F}_P| \tag{4.9}$$

With $flaws : \mathscr{P} \rightarrow 2^{\mathbb{F}}$ being the function that retrieves all flaws that have been issued for a given plan.

Unfortunately, the flaw-oriented plan selections basically face the same drawbacks that we discussed above for the modification-oriented strategies. Having said that, since in general a large number of open issues in a plan will most likely require a large amount of plan manipulations, an inverted flaw-oriented plan selection can be reasonably deployed in order to navigate towards more "worked-out" plans. The heuristic however inconsistently under- and over-estimates the remaining costs, so the quality of this kind of estimation **alone** is not too reliable. This is because plan modifications may address several flaws at a time, which makes the function less informed, and that refinements may introduce new flaws, which spoils the function's admissibility. Besides these considerations, it has to be kept in mind that $f^{planSel}_{Addr-\mathbb{F}_P}$ may be used (positive, not inverted) as a profitable subsequent strategy element, as it may make sense to look into plans that offer more choices.

A flaw-centered plan selection can also be formulated as a relative measure, analogously to the modification-based counter-part strategies. Normalizing the preferred subset of flaws with respect to all fellow flaws grounds the estimation of future costs during phases of the solution development in which the fringe bears a large variety of deficiency situations:

$$P_i < P_j \in f^{planSel}_{Pref-\mathbb{F}_P/\mathbb{F}}(\{P_1, \ldots, P_n\}) \text{ if } \frac{|flaws(P_i) \cap \mathbb{F}_P|}{|flaws(P_i)|} > \frac{|flaws(P_j) \cap \mathbb{F}_P|}{|flaws(P_j)|} \tag{4.10}$$

Like for the modification-based selection $f^{planSel}_{Pref-\mathbb{M}_P/TE}$ we can also relate the number of occurrences of a preferred flaw class $\mathbb{F}_P$ to the number of plan steps. The rationale for this method is that the size of the plan is considered to represent an amount of planning work done, now put in perspective to what

[3] Domain model conciseness (see Def. 2.8) implies that there exists no refinement for a plan that "merges" a group of task expressions into one single task schema instance. Hence, the number of plan steps remains at least constant from refinement to refinement, independent of the deployed system configuration.

still has to be done. In this way, the strategy is more retrospectively estimating how elaborated a plan is.

$$P_i < P_j \in f^{planSel}_{Pref-\mathbb{F}_P/TE}(\{P_1,\ldots,P_n\}) \text{ if } \frac{|flaws(P_i)\cap\mathbb{F}_P|}{|TE_i|} > \frac{|flaws(P_j)\cap\mathbb{F}_P|}{|TE_j|} \tag{4.11}$$

Some general remarks regarding flaw-centered plan selections: We cannot recommend a preference selection based on flaw types, because these strategies select plans on the occurrence of deficiencies that are finally solved. After applying a plan modification, the flaw situation typically changes such that the addressed flaw is eliminated and (if at all) new types of deficiencies arise. It is therefore very probable that flaw-centered selections loose track of these new refinements and switch back to the clique of the previous decision point, and this behaviour resembles an exhaustive breadth-first schema. They cannot give any advice beyond a certain point in the search space and the planning system has to operate randomly until a preferred flaw emerges among the refinements. They may however contribute as consecutive strategy components in order to provide preceding components some options that are free of the focused flaw class. In this sense, the flaw-oriented strategies can be labelled as "selective breadth-first" strategies and constitute an alternative to the history-aware selection function defined above.

A similar line of argument holds for the applicability of inversely defined flaw-oriented plan selection functions. Primary strategies that favour fewer occurrences of a specific flaw class, either on an absolute or relative measure, face the problem of becoming obsolete on all refinement paths in which the avoided flaw class appears. If solving the planning problem is intrinsically connected to solving such deficiencies (and there is practically no flaw class that can be avoided in general), then the strategies build up the refinement space until all plans in the fringe carry flaws of the avoided class. But, if deployed as a subsequent strategy, the relative flaw avoidance may bridge the strategic gap in cases where preceding strategies are undecided: the composed strategy may then focus on refinements that have overcome a certain deficiency and are therefore closer to a solution.

If (some of) the flaw classes are known for the configuration at hand, more specific estimates for the current state of plan development can be given. From the partial-order planning literature comes a set of plan selection functions that embody heuristics for an A^* algorithm [129], originally intended to perform node-selection in the UCPOP system.

Gerevini and Schubert proposed in [113,236] to relate the size of certain plan component sets to specific plan deficiencies. They identified two very effective strategies called S+OC and CL+OC. According to the usual definition of the A^* heuristic function as $f(n) = g(n) + h(n)$, that means, the sum of actual and estimated future costs, these functions use the number of open preconditions as the estimating term (*OC*) and the number of plan steps (*S*), respectively the number of causal links (*CL*) as the actual cost term. In our framework, the two strategies can be directly transcribed into the plan selection functions

$$P_i < P_j \in f^{planSel}_{\text{S+OC}}(P_1,\ldots,P_n) \text{ if } \frac{|TE_i| + |flaws(P_i)\cap\mathbb{F}_{\texttt{OpenPrec}}|}{|TE_j| + |flaws(P_j)\cap\mathbb{F}_{\texttt{OpenPrec}}|} < 1 \tag{4.12}$$

and

$$P_i < P_j \in f^{planSel}_{\text{CL+OC}}(P_1,\ldots,P_n) \text{ if } \frac{|CL_i| + |flaws(P_i)\cap\mathbb{F}_{\texttt{OpenPrec}}|}{|CL_j| + |flaws(P_j)\cap\mathbb{F}_{\texttt{OpenPrec}}|} < 1 \tag{4.13}$$

In both equations, the number of flaws of the open precondition type `OpenPrec` enters the equation as the estimation term for future costs.

Since the authors only had to deal with partial-order planning (refinements), the above heuristics do restrict all cost calculations to the question to which extent the causal chains have been developed. The natural estimate is consequently given by the number of causal links that have to be added[4] – either by using existing plan steps or by introducing new task schema instances. This view makes the number of task expressions, respectively the number of causal links an adequate cost measure. Which one to deploy depends on two factors: including the number of plan steps makes S+OC an inconsistent heuristic and A^* consequently cannot guarantee to find the optimal solution in terms of plan steps. The causal link cost measure in CL+OC

[4] Please remember that UCPOP action schemata use quantified literals as preconditions, with the quantification interpreted over the universal base of the model [214]. Hence, there is a "number" of open or closed precondition literals associated with each plan step.

uses an admissible estimate but it targets the heuristic at the minimal number of causal link *insertions* and does therefore not state anything about the number of plan steps (re-use of links produces the same cost as introducing a task instance). Nevertheless, the above plan selection heuristics appear to be among the most successful planning strategies for partial-order planning during the early 90's.

In an hybrid configuration, however, cost and estimate measures are not balanced because they do not take into account the wide-ranging changes of task expansion modifications to the plan structure. We have adjusted this deficiency in two steps: Firstly, we extended the flaw census to the `AbstrTask` class. The following two plan selection functions consequently take into account that abstract tasks produce some efforts in terms of dealing with future tasks and in terms of maintaining the causal structure of the expansion networks.

$$P_i < P_j \in f^{planSel}_{\text{S+OCA}}(P_1,\ldots,P_n) \text{ if } \frac{|TE_i| + |flaws(P_i) \cap \left(\mathbb{F}_{\texttt{OpenPrec}} \cup \mathbb{F}_{\texttt{AbstrTask}}\right)|}{|TE_j| + |flaws(P_j) \cap \left(\mathbb{F}_{\texttt{OpenPrec}} \cup \mathbb{F}_{\texttt{AbstrTask}}\right)|} < 1 \tag{4.14}$$

$$P_i < P_j \in f^{planSel}_{\text{CL+OCA}}(P_1,\ldots,P_n) \text{ if } \frac{|CL_i| + |flaws(P_i) \cap \left(\mathbb{F}_{\texttt{OpenPrec}} \cup \mathbb{F}_{\texttt{AbstrTask}}\right)|}{|CL_j| + |flaws(P_j) \cap \left(\mathbb{F}_{\texttt{OpenPrec}} \cup \mathbb{F}_{\texttt{AbstrTask}}\right)|} < 1 \tag{4.15}$$

It is an unpleasant property of the above selections that the cost-factor of a task expression does not take into consideration that the expansion of an abstract task expression typically introduces a group of plan steps within a single refinement. Our consequently defined A^*-Variant PSA+OCA employs a heuristic function $psa : \mathscr{P} \rightarrow \mathbb{N}$ for retrieving the number of plan modifications in a plan's history that have added a task expression (by inspecting the respective $E^{\oplus}$ set). With this definition, a plan that obtained its plan steps via expansion becomes cheaper than a plan that had to insert its task expressions step by step. It is also easy to see that this view better correlates with the cost-estimate factor of abstract plan steps, since it models the incurring costs adequately. The resulting plan selection function is defined as follows:

$$P_i < P_j \in f^{planSel}_{\text{PSA+OCA}}(P_1,\ldots,P_n) \text{ if } \frac{psa(P_i) + |flaws(P_i) \cap \left(\mathbb{F}_{\texttt{OpenPrec}} \cup \mathbb{F}_{\texttt{AbstrTask}}\right)|}{psa(P_j) + |flaws(P_j) \cap \left(\mathbb{F}_{\texttt{OpenPrec}} \cup \mathbb{F}_{\texttt{AbstrTask}}\right)|} < 1 \tag{4.16}$$

4.1.3 Generalized Flaw- and Modification-Based Strategies

A first generalization from specific flaw and modification classes (and from concrete data structures for representing plans) is to base the decision of plan selection functions on the sizes of the sets of detected flaws and proposed modifications. We will see that these functions incorporate a very strong notion of flexibility.

The simplest form of flexibly generalizing from the provided flaw and modification classes can be realized for the plan selection functions. Absolute measures for such strategic advice are defined as follows:

$$P_i < P_j \in f^{planSel}_{Fewer-\mathbb{F}}(P_1,\ldots,P_n) \text{ if } |flaws(P_i)| < |flaws(P_j)| \tag{4.17}$$

And the analogous function for the published modification proposals:

$$P_i < P_j \in f^{planSel}_{Fewer-\mathbb{M}}(P_1,\ldots,P_n) \text{ if } |mods(P_i)| < |mods(P_j)| \tag{4.18}$$

If the above strategy definitions appear to be trivial, please bear in mind that the utilized information is only accessible so easily due to our system design in which flaws and modifications are presented explicitly.

It is a major drawback of these selection functions that in general a larger number of tasks is more likely to be flawed than a small one and the same holds for the modifications. As a result, these absolute measures tend to favour plans that are at an early stage of their development. This preference in turn leads to an exhaustive schema that works in phases in a breadth-first manner. In order to take account of the different development stages in the fringe, the above measure can be normalized by relating the set sizes to the number of plan steps. This relative preference can be directed at the plan deficiencies:

$$P_i < P_j \in f^{planSel}_{\mathbb{F}/TE}(P_1,\ldots,P_n) \text{ if } \frac{|flaws(P_i)|}{|TE_i|} < \frac{|flaws(P_j)|}{|TE_j|} \tag{4.19}$$

as well as at the plan modifications that have been issued:

$$P_i < P_j \in f^{planSel}_{\mathbb{M}/TE}(P_1,\ldots,P_n) \text{ if } \frac{|mods(P_i)|}{|TE_i|} < \frac{|mods(P_j)|}{|TE_j|} \tag{4.20}$$

Experimental evidence has shown that in particular the $f^{planSel}_{\mathbb{F}/TE}$ heuristic works very effectively (see Chap. 6), even as a primary strategy. Although it is also not an admissible heuristic – like the previous flaw- and modification-oriented plan selections –, the total number of flaw instances seems to be a fairly precise estimate in most practical scenarios for the amount of modification work that lies ahead in the refinement space. An important influence for its success is its support of paths on which the deficiencies are steadily eliminated. If a plan is chosen by $f^{planSel}_{\mathbb{F}/TE}$ and at least one of its refinements eliminates one flaw without producing a new one, then this successor plan will be selected in the following planning phase. This form of stability evidently pays off.

The modification ratio is focusing on the local implications that a decision has in terms of commitment. It does not directly constitute an estimate regarding the distance to a solution but rather quantifies the amount of distraction on a (potential) solution path. Since typically most of the plan modifications for un-addressed flaws persist into the refinement and are re-issued, the branching factors of a plan's refinements can be estimated by its own plan modification set. To put it more pointed: the question whether to use the detection ratio or the modification ratio corresponds to the question whether to search for the plan with the minimal distance to the solution or to search for the plan that offers the least alternatives for its refinements.

It has to be stressed again that both metrics can be expected to offer valuable strategic advice and that they do so without any significant computational overhead. Regarding an inverse definition of the above plan selections, it makes no sense to consider the flaw-oriented functions (4.17) and (4.19) but in cases where they are used as a last component that has to provide alternatives where preceding components are undecided. If they are deployed too early in the strategy composition chain, all plans that are close to a solution are generally delayed.

Another consideration that is frequently met in the planning-strategy literature is to reason about the deficiencies "position" with respect to the intended plan execution time-line. There is a number of planning systems that include the ordering of the plan steps in their strategic reasoning: The examples range from flaw selection heuristics in HTN planning [184,270] and resource-planning [163] to planning algorithms that inherently incorporate the "address early flaws" strategy [93, 200] (plus all heuristic state-space planners). The underlying assumption is that the emerging execution order imposes in an organic manner the adequate constraints on the subsequent plan segments.

$$\begin{aligned} \mathtt{m}_i < \mathtt{m}_j \in f^{modSel}_{\prec}(P,\{\mathtt{f}_1,\ldots,\mathtt{f}_m\},\{\mathtt{m}_1,\ldots,\mathtt{m}_n\}) \\ \text{if } \mathtt{m}_i \in modsFor(\mathtt{f}_a,P), \mathtt{m}_j \in modsFor(\mathtt{f}_b,P) \\ \text{and } \forall te_a \in (\mathtt{f}_a \cap TE), \forall te_b \in (\mathtt{f}_b \cap TE): \\ (\mathtt{f}_a \cap TE \neq \emptyset \wedge \mathtt{f}_b \cap TE \neq \emptyset) \Rightarrow te_a \prec te_b \in \prec \end{aligned} \tag{4.21}$$

The fundamental aspect of the *prefer early flaws* heuristic is of course less esoteric and more of technical nature: developing a plan from the beginning has the advantage of a complete state information in the initial state. In fact, this is the main explanation for the speed and expressivity of forward-search state-based planning and ordered HTN planning [202]. This principle can be transfered to our refinement-based planning framework, its impact is however not clearly evident yet. For the time being, we are inclined to believe that its benefit in our context is not even domain but problem dependent.

The principle of *least commitment* has always been a major issue in Artificial Intelligence planning. The basic idea is to make a decision in view of the implications that it may have on all decisions yet to come and in particular in view of the consequences a wrong decision has on the process of finding a solution. Its motivation lies in the observation that the number and length of backtracking paths increase if nodes with a high branching-factor appear closer to the root-node and the density of solutions in the search space is low, that means, there is a large number of futile alternatives "below" a faulty decision point. This problem is addressed by the least-commitment principle such that it organizes the search space in a way that every decision does put as few restrictions on future decisions as possible.

Partial-order planning is a prime example for the idea of least commitment, because the ordering of plan steps is deferred until a specific ordering becomes strictly needed and the task parameters are finally assigned at the latest decision-point possible. In terms of the refinement planning framework, least commitment means to select refinements that impose only as few constraints on further refinements as necessary. A measure for quantifying the level of commitment is therefore the size of the refinement space that is covered by the selection versus the space that is ruled out by the decision. In the view of our framework, the plan modifications for different flaws represent a disjunctive search space. This implies that basically most of the plan modifications are only mutual exclusive with respect to their fellow modifications that address the same flaw instance. As a consequence, selecting one modification rules out the application of its alternative flaw-solvers but is likely to be independent from those of other flaws. In other words: the level of commitment for a single plan modification can be estimated in terms of the number of solving alternatives that exist for the corresponding flaw. The more alternatives exist, the higher is the commitment for choosing among them.

Along these considerations, we developed a modification selection function called *least committing first* f_{LCF}^{modSel}, which prefers modifications according to the number of fellow-modifications that have been published to address their shared corresponding flaws. It is defined by the following equation:

$$\begin{aligned} \mathtt{m}_i < \mathtt{m}_j \in f_{LCF}^{modSel}(P, \{\mathtt{f}_1, \ldots, \mathtt{f}_m\}, \{\mathtt{m}_1, \ldots, \mathtt{m}_n\}) \\ \text{if } \mathtt{m}_i \in \mathit{modsFor}(\mathtt{f}_a, P), \mathtt{m}_j \in \mathit{modsFor}(\mathtt{f}_b, P) \\ \text{and } |\mathit{modsFor}(\mathtt{f}_a, P)| < |\mathit{modsFor}(\mathtt{f}_b, P)| \end{aligned} \tag{4.22}$$

It can easily be seen that this is a *flexible* strategy, since it does not depend on the actual types of issued flaws and modifications: it just compares answer set sizes in order to keep the branching-factor in the search space low. Experimental evidence has shown that the LCF modification selection function is not only among the best performing primary strategies but also that it maintains its performance across the domains. This form of stability confirms previous results in the field concerning least commitment strategies. As an interesting historic note, many classical strategies have been formulated in terms of such more general flaw repair costs. Their algorithmic procedure however jams these principles into a fixed preference schema like the ones presented below and in the related work sections, and uses them only as a tie-break rule for equal preferences.

Least committing first can also be seen as a generalized adaptation of the successful *Least Cost Flaw Repair* partial-order planning strategy, a method that repairs first the (causal threat and open precondition) flaws associated with the minimal number of modifications [143, 218]. In this sense, it roughly corresponds to an LCFR with uniform modification costs. Other approaches formulate it more conservatively in terms of delaying over-committing modifications, for example, postponing the treatment of causal threats for which multiple resolutions exist [215]. Thanks to our framework's formal basis and uniform representation, we can easily extend the field of application for this heuristic to all configurations of hybrid planning and scheduling as well. It is thereby combined it with the principles from the UMCP *fewest alternatives first* task expansion strategy, which selects expansion networks according to the number of alternatives that are applicable [271]. We think that f_{LCF}^{modSel} is also a generalization[5] of the resolver selection heuristic that is used in the IXTET temporal resource-planning system. Using a depth-first plan selection, this algorithm computes the commitment costs for applying a resolver (which corresponds to a subset of our plan modifications) in terms of actual interval restrictions and a lower bound for expected costs [116]. Since the least commitment principle appears to be successful in all key-areas of hybrid planning and scheduling, and because our *least committing first* strategy is able to cover its functionality to a major extent, we are very confident that we have implemented a powerful "general-purpose" strategy for all presented configurations.

A directly corresponding plan selection to LCF can be characterized by preferring plans in the fringe according to the number of fellow-plans that are refinements of the same (processed) plans. The strategy $f_{Siblings}^{planSel}$ appears however not to be an adequate adaptation of the least commitment principle. Since it performs an exhaustive search among members of small families, its contributions to a strategy repertoire are considered marginal. It may therefore only play a part as a final judgement in larger strategy function arrangements.

[5] It has to be noted that in resource-planning configurations, more specific modification selection functions can be employed. They can actually perform the described interval arithmetics for quantifying the commitment level in the required detail.

4.1.4 The HotSpot Technology

The presented implementation of the least-commitment principle is in practice a very efficient strategy. It has however two obvious drawbacks: First, it would be much better informed if it took the actual structure of the compared options into account. It is, for example, not very intuitive to presume that choosing one of two large task expansions represents less commitment than choosing one of three variable co-designations. The expansion obviously makes considerably more changes to the plan structure, and we expect that the "amount of change" has side effects on the overall flaw and modification availability in the future. In particular, it is not always the case that the modifications do not influence each other if they have been issued for different flaws, because the flawed elements are the same or depend on each other indirectly, etc. For example, think of an expansion, which typically affects the (number of consistent) variable co-designation modifications that are publishable for the refinement plan. The second drawback concerns the restriction of the *least committing first* strategy to modification selection: if the mutual influence of flaws and modifications was quantified, then there would be a measure for comparing plans according to their overall flaw and modification interaction "level", and hence a plan selection was available.

Our proposal is to lift the least-commitment principle from a blind comparison to a more informed decision-making on the plan structure that is still flexible. The principal hypothesis is that a plan typically has "weak points" at which numerous deficiencies can be identified and that refinements for fixing these problems typically interfere with each other by imposing constraints on subsequent refinements. We therefore advise the system to solve *isolated sub-problems first* and expect that the resolving operations' effects will have only local consequences on the plan structure – this is a strong notion of least commitment. The following sections define corresponding modification and also plan selection functions that operate with this novel concept of multi-flawed plan components, so-called *HotSpot* components. We present the different modes of HotSpot handling that we have identified so far.

We begin with a simple flaw-based modification selection that tries to avoid *direct uniform* HotSpots. That means, the function relates the announced flaws with the number of their shared references to plan components and prefers the available plan modifications according to the number value of their associated flaws. An early commitment (cf. [144]), that means, selecting modifications that affect more frequently referenced plan components, is thereby avoided.

$$\begin{aligned} \mathtt{m}_i < \mathtt{m}_j \in f^{modSel}_{DirUniHS}(P, \{\mathtt{f}_1, \ldots, \mathtt{f}_m\}, \{\mathtt{m}_1, \ldots, \mathtt{m}_n\}) \\ \text{if } \mathtt{m}_i \in modsFor(\mathtt{f}_a, P),\ \mathtt{m}_j \in modsFor(\mathtt{f}_b, P) \\ \text{and } \sum_{\mathtt{f} \in (\{\mathtt{f}_1,\ldots,\mathtt{f}_m\} \setminus \mathtt{f}_a)} |\mathtt{f} \cap \mathtt{f}_a| < \sum_{\mathtt{f} \in (\{\mathtt{f}_1,\ldots,\mathtt{f}_m\} \setminus \mathtt{f}_b)} |\mathtt{f} \cap \mathtt{f}_b| \end{aligned} \tag{4.23}$$

Consider the following example: Let $\mathtt{f}_1 = \{te_a, \phi_a\}$ and $\mathtt{f}_2 = \{te_b, \phi_b\}$ be two flaws of class $\mathbb{F}_{\mathtt{OpenPrec}}$, and let $\mathtt{f}_3 = \{te_a\} \in \mathbb{F}_{\mathtt{AbstrTask}}$. According to the direct uniform HotSpot definition, flaws $\mathtt{f}_1$ and $\mathtt{f}_3$ both refer to a plan step te_a, that means, they are both assigned a HotSpot value of 1. $\mathtt{f}_2$ is the only flaw that marks its plan components and is therefore assigned 0. As a consequence, the selection function prefers all modifications that address $\mathtt{f}_2$ over the modifications for $\mathtt{f}_1$ and $\mathtt{f}_3$ likewise.

Direct HotSpot calculations offer a first insight into the inter-flaw relationships but do not capture relatively trivial dependencies between the flawed plan components. We therefore employ the notion of plan sub-components (see Sec. 2.6.2) in order to define HotSpot heuristics of finer granularity. The function *comp*, which returns all sub-components of a set of plan components, has been introduced in Sec. 3.2.1. Given that, the following strategic function defines the *indirect uniform* HotSpot modification selection:

$$\begin{aligned} \mathtt{m}_i < \mathtt{m}_j \in f^{modSel}_{IndUniHS}(P, \{\mathtt{f}_1, \ldots, \mathtt{f}_m\}, \{\mathtt{m}_1, \ldots, \mathtt{m}_n\}) \\ \text{if } \mathtt{m}_i \in modsFor(\mathtt{f}_a, P),\ \mathtt{m}_j \in modsFor(\mathtt{f}_b, P) \\ \text{and } \sum_{\mathtt{f} \in (\{\mathtt{f}_1,\ldots,\mathtt{f}_m\} \setminus \mathtt{f}_a)} |comp(\mathtt{f}) \cap comp(\mathtt{f}_a)| < \sum_{\mathtt{f} \in (\{\mathtt{f}_1,\ldots,\mathtt{f}_m\} \setminus \mathtt{f}_b)} |comp(\mathtt{f}) \cap comp(\mathtt{f}_b)| \end{aligned} \tag{4.24}$$

The justifiable question arises why to maintain both measure types, the direct and the indirect HotSpot. We do so, because there are different target application areas for both of them: The lower granularity of a direct HotSpot heuristic is more suitable in a strategy composition where subsequent strategies are deployed. In

this case, we are not interested in the fine-granular computations of the indirect HotSpot that produces much smaller equivalence classes, which can hardly be post-processed in a useful way. On the other hand, if we are looking for a component that performs a final decision, the direct variant is possibly not able to partition the remaining equivalent choices adequately.

For the definition of plan selection functions, we can derive the respective HotSpot metrics directly from the presented modification selections. The flaw-oriented *direct* and *indirect uniform HotSpot* plan selection functions $f^{planSel}_{DirUniHS}$ and $f^{planSel}_{IndUniHS}$ accumulate the direct, respectively indirect HotSpot values for each flaw in a plan and relate these values to the total number of detected flaws per plan.

$$P_i < P_j \in f^{planSel}_{DirUniHS}(P_1,\ldots,P_n) \quad \text{if } \frac{\sum_{\mathtt{f}\in flaws(P_i)}\sum_{\mathtt{f}'\in(flaws(P_i)\setminus\mathtt{f})}|\mathtt{f}\cap\mathtt{f}'|}{|flaws(P_i)|} < \frac{\sum_{\mathtt{f}\in flaws(P_j)}\sum_{\mathtt{f}'\in(flaws(P_j)\setminus\mathtt{f})}|\mathtt{f}\cap\mathtt{f}'|}{|flaws(P_j)|} \tag{4.25}$$

The indirect HotSpot calculation is defined as follows:

$$P_i < P_j \in f^{planSel}_{IndUniHS}(P_1,\ldots,P_n) \quad \text{if } \frac{\sum_{\mathtt{f}\in flaws(P_i)}\sum_{\mathtt{f}'\in(flaws(P_i)\setminus\mathtt{f})}|comp(\mathtt{f})\cap comp(\mathtt{f}')|}{|flaws(P_i)|} < \frac{\sum_{\mathtt{f}\in flaws(P_j)}\sum_{\mathtt{f}'\in(flaws(P_j)\setminus\mathtt{f})}|comp(\mathtt{f})\cap comp(\mathtt{f}')|}{|flaws(P_j)|} \tag{4.26}$$

By computing the average number of overlappings between the announced flaws, the HotSpot plan selections provide a first estimation of the degree of interdependencies between defects in plans and the consequent mutual influences of the associated plan modifications.

Instead of only concentrating on the actual number of commonly affected plan components, we also developed an adaptive HotSpot calculation that tries to *predict the conflict potential* between two flaw *classes*. During the exploration of the refinement space, the strategy might find that for the given planning problem and domain, open preconditions often overlap with abstract plan steps. This may be an artefact of the strategy itself, an inherent property of the domain, or a characteristic of the problem instance, but the system can nevertheless induce from *experience* that these two flaw classes appear to be somehow connected. Consequently, for the current plan generation episode, the HotSpot computation can be modulated by these experience values.

Regarding the calculation of direct HotSpots for selecting modifications, the following strategy does not treat the flaws uniformly but associates all cross-references by flaws with an adaptive weight function w. The weights are the estimates for the "interactiveness" of flaw classes on a HotSpot. Apart from that, it is constructed like the uniform version.

$$\begin{aligned}\mathtt{m}_i < \mathtt{m}_j \in f^{modSel}_{DirAdaptHS}(P,\{\mathtt{f}_1,\ldots,\mathtt{f}_m\},\{\mathtt{m}_1,\ldots,\mathtt{m}_n\}) \\ \text{if } \mathtt{m}_i \in modsFor(\mathtt{f}_a,P),\ \mathtt{m}_j \in modsFor(\mathtt{f}_b,P) \\ \text{and } \sum_{\mathtt{f}\in(\{\mathtt{f}_1,\ldots,\mathtt{f}_m\}\setminus\mathtt{f}_a)} |\mathtt{f}_a\cap\mathtt{f}|\cdot w(\mathtt{f}_a,\mathtt{f}) < \sum_{\mathtt{f}\in(\{\mathtt{f}_1,\ldots,\mathtt{f}_m\}\setminus\mathtt{f}_b)} |\mathtt{f}_b\cap\mathtt{f}|\cdot w(\mathtt{f}_b,\mathtt{f})\end{aligned} \tag{4.27}$$

The adaptive weight function is updated between two applications of the HotSpot modification selection. It is defined as the function $w : \mathbb{F}\times\mathbb{F}\to\mathbb{R}$ by the following equation

$$w(\mathtt{f}_1,\mathtt{f}_2) = \frac{spot_d(\mathtt{f}_1,\mathtt{f}_2)}{\max_{\mathtt{f}_x,\mathtt{f}_y\in\mathbb{F}}(spot_d(\mathtt{f}_x,\mathtt{f}_y))}\cdot w'(\mathtt{f}_1,\mathtt{f}_2)$$

The function $spot_d$, which will be defined below, keeps record of the accumulated amount of (direct uniform) HotSpot values between members of the involved flaw classes. The division by the maximum value in the HotSpot record over all flaw classes normalizes the weight across the earlier selection-function application episodes. A second weight-factor is the function w', a pre-defined modulation table in which *directed* conflict weights can be set. This mechanism allows us to define, for example, that a HotSpot relationship between

an open precondition flaw and an abstract task flaw is more dominant on the open precondition's side. Since our experience indicates that resolving an open precondition issue often interferes with necessary constraints in the task expansion networks, the user-defined HotSpot weight puts slightly more penalty on the open precondition flaw's HotSpot value in this case. Appendix A contains an example modulation table that we used in our experiments. If however these two flaw classes rarely coincide, the modulation function w' becomes less and less influential as the normalization term dominates.

We now define the function that records the sum of HotSpot values between two instances of a flaw class over the current plan generation episode. Let $\mathscr{P}_{history}$ be the set of all plans that have been visited by the refinement planning algorithm so far and let $\mathbb{F}_1$ and $\mathbb{F}_2$ be the respective flaw classes for flaw instances $\mathtt{f}_1$ and $\mathtt{f}_2$:

$$spot_d(\mathtt{f}_1,\mathtt{f}_2) = \sum_{P\in\mathscr{P}_{history}} \left(\sum_{\mathtt{f}\in(flaws(P)\cap\mathbb{F}_1)} \left(\sum_{\mathtt{f}'\in((flaws(P)\cap\mathbb{F}_2)\setminus\{\mathtt{f}\})} |\mathtt{f}\cap\mathtt{f}'| \right) \right)$$

Note that the argument flaws are not necessarily of different classes.

The adaptive HotSpot modification selection can also be defined over indirect flaw relationships. We omit an explicit presentation of the $f^{modSel}_{IndAdaptHS}$ strategy component and ask the reader to refer to the previous selection function definition. The novel part lies of course in the adaptive weight function that uses an appropriate *indirect HotSpot value* accumulating function:

$$\begin{aligned} &spot_i(\mathtt{f}_1,\mathtt{f}_2) = \\ &\quad \sum_{P\in\mathscr{P}_{history}} \left(\sum_{\mathtt{f}\in(flaws(P)\cap\mathbb{F}_1)} \left(\sum_{\mathtt{f}'\in((flaws(P)\cap\mathbb{F}_2)\setminus\{\mathtt{f}\})} |comp(\mathtt{f})\cap comp(\mathtt{f}')| \right) \right) \end{aligned} \tag{4.28}$$

The HotSpot strategies defined so far exclusively focus on the flaws' commonalities. As already suggested in the introductory paragraph to this section, overlappings in plan modifications can also be analyzed for commitment estimates. A direct modification HotSpot calculation as a counterpart to the flaw-oriented selection (4.23) is of course possible, however not very reasonable, since elementary additions never share components directly (all of them are new) and deletion overlappings hardly ever occur (in all presented configurations, they only arise in alternative expansions). We therefore define the modification selection function $f^{modSel}_{ModBasedHS}$ analogously to the indirect uniform HotSpot strategy (4.24) using the following "trick": instead of tracking the usual sub-component references, this function consults the domain model when determining the respective entities' sub-components. By doing so, two task inserting modifications overlap if they share the same task schema, and the like.

$$\begin{aligned} &\mathtt{m}_i < \mathtt{m}_j \in f^{modSel}_{ModBasedHS}(P,\{\mathtt{f}_1,\ldots,\mathtt{f}_m\},\{\mathtt{m}_1,\ldots,\mathtt{m}_n\}) \\ &\qquad \text{if} \sum_{e_i\in E_i^{\oplus}\cup E_i^{\ominus}} \left(\sum_{\mathtt{m}'\in(mods(P)\setminus\{\mathtt{m}_i\})} \left(\sum_{e'\in E^{\oplus\prime}\cup E^{\ominus\prime}} spot_m(comp^*(e_i),comp^*(e'),0) \right) \right) \\ &\qquad < \sum_{e_j\in E_j^{\oplus}\cup E_j^{\ominus}} \left(\sum_{\mathtt{m}'\in(mods(P)\setminus\{\mathtt{m}_j\})} \left(\sum_{e'\in E^{\oplus\prime}\cup E^{\ominus\prime}} spot_m(comp^*(e_j),comp^*(e'),0) \right) \right) \end{aligned} \tag{4.29}$$

The selection function performs a pair-wise comparison between the elementary modifications in the issued plan modifications. It does so by employing $comp^*$, a function that returns not only the referenced components, but also the respective domain model references for elementary additions. With this information, the $spot_m$ function computes the overlap between plan components. Since there are many connections into and within the domain model components, it is very probable that any overlapping will cover the complete domain model. For example, following a task insertion to its task schema is only one step away from the task schema usage in an expansion network, which in turn is only one step away from the respective abstract task schema. Therefore, the heuristic influence of transitive references inside the domain model must decay and the overlap computation can be defined as the following function: Let E_1 and E_2 be two sets of plan components and domain model elements and let $n,\theta \in \mathbb{N}$ be natural numbers that represent the

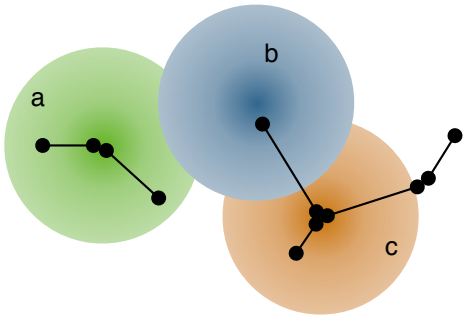


Figure 4.2: Flawed plan components establish HotZone areas.

"distance" of an indirect component relationship, respectively the slope of the decay of this relationship's influence.

$$spot_m(E_1,E_2,n) = |E_1 \cap E_2| \cdot \exp(-\frac{n}{\theta}) + \sum_{e_1 \in E_1} \sum_{e_2 \in E_2} spot_m(comp^*(e_1), comp^*, n+1)$$

Please note that the computation terminates because all domain models are finite.

Like we have shown for the flaw-oriented modification and plan selection functions, the above modification-based HotSpot computations can be conveyed into plan selections as well: We developed a strategy that prefers those plans that exhibit a lower average value of discovered modification HotSpots. By that, it corresponds to the direct uniform HotSpot plan-selection. The $f^{planSel}_{FewerModBHS}$ strategy function accumulates the computed modification overlappings as provided by the above $f^{modSel}_{ModBasedHS}$ for a single modification. It finally relates the HotSpot values to the total number of plan modifications issued for that particular plan in order to normalize the values.

4.1.5 From HotSpots to HotZones

The notion of HotSpots is to focus on outstanding components of the partial plan data-structure. To some extent, this imitates the approach of a human user who usually concentrates on specific parts of a plan when reasoning about development options. In order to improve the discovery of commitment and the quality of the respective measure we extend the HotSpot concept by taking into account how the affected plan (sub-) components are connected with each other in the plan: thereby connected flaws constitute so-called *HotZones* in the plan. Fig. 4.2 illustrates this conceptualization: The plan components that are directly referenced by flaw c are depicted in the center of the orange disc. Their HotSpot value is propagated into the components that are connected to them with a certain decay, and so the plan components that are directly flawed by b receive some additional score. The sphere of a is isolated, that means, resolving flaw a is more likely to imply only local commitments than b or c. This way of strategic reasoning is completely original and can only be accomplished in our refinement planning framework with its explicit option representation. The employed algorithmic procedure, again, appears surprisingly simple: It builds an initial map of all HotSpots in the partial plan and then re-evaluates each individual HotSpot according to the HotSpots on its neighbourhood components.

$$\begin{aligned} \mathtt{m}_i < \mathtt{m}_j \in f^{modSel}_{HZone}(P, \{\mathtt{f}_1,\ldots,\mathtt{f}_m\}, \{\mathtt{m}_1,\ldots,\mathtt{m}_n\}) \\ \text{if } \mathtt{m}_i \in mods(\mathtt{f}_a, P), \mathtt{m}_j \in mods(\mathtt{f}_b, P) \\ \text{and } spot_{zone}(\mathtt{f}_a, \{\mathtt{f}_1,\ldots,\mathtt{f}_m\} \setminus \mathtt{f}_a, 0) < spot_{zone}(\mathtt{f}_b, \{\mathtt{f}_1,\ldots,\mathtt{f}_m\} \setminus \mathtt{f}_b, 0) \end{aligned} \tag{4.30}$$

The evaluation of the flaws is defined recursively with respect to the value of flaws that refer to the same sub-components. The pre-defined constant parameter $\theta \in \mathbb{N}$ thereby modulates the slope of the decay with

respect to the neighbours' influence over an increasing distance of $n \in \mathbb{N}$.

$$spot_{zone}(\mathtt{f}, \{\mathtt{f}_1, \ldots, \mathtt{f}_m\}, n) = \begin{cases} 0 & \text{for } m = 0 \\ \sum\limits_{\mathtt{f}' \in \{\mathtt{f}_1, \ldots, \mathtt{f}_m\}} |comp(\mathtt{f}) \cap comp(\mathtt{f}')| \cdot \exp(-\frac{n}{\theta}) & \text{else} \\ + spot_{zone}(\mathtt{f}', \{\mathtt{f}_1, \ldots, \mathtt{f}_m\} \setminus \{\mathtt{f}'\}, n+1) & \end{cases}$$

The HotZone concept is also applicable to the task of plan selection, namely in two ways: the simpler form is identifying the plan with the smallest maximal HotZone value, the *least hottest zone* so to speak. The $f^{planSel}_{LeastHZone}$ strategy function performs for each flaw in each plan the calculations of f^{modSel}_{HZone}, stores the maximum HotZone value for each plan, and eventually prefers the less rated plans as described.

$$\begin{aligned} P_i < P_j \in f^{planSel}_{LeastHZone}(P_1, \ldots, P_n) \\ \text{if} \quad \max_{\mathtt{f} \in flaws(P_i)} (spot_{zone}(\mathtt{f}, flaws(P_i) \setminus \{\mathtt{f}\}, 0) \\ < \quad \max_{\mathtt{f} \in flaws(P_j)} (spot_{zone}(\mathtt{f}, flaws(P_j) \setminus \{\mathtt{f}\}, 0) \end{aligned} \tag{4.31}$$

The other HotZone plan-selection implementation is $f^{planSel}_{FewerHZones}$, a function that prefers plans with fewer HotZone "clusters". Such clusters are identified by an algorithm that operates similar to the basic HotZone computation: When tracking the transitive overlappings of components, the set of flaws can be partitioned accordingly into dependent sub-sets. The number of identified sub-sets is the number of (independent) HotZones. Such a partitioning is achieved by the following clustering function $clusters : 2^{\mathbb{F}} \rightarrow 2^{2^{\mathbb{F}}}$. It takes a set of flaws as an argument and returns a set of sets of flaws with each set representing the inter-connected members of one HotZone.

$$clusters(\{\mathtt{f}_1, \ldots, \mathtt{f}_m\}) =$$
$$\begin{cases} \emptyset & \text{if } m = 0 \\ \{\{\mathtt{f}_1\}\} & \text{if } m = 1 \\ clusters(\{\mathtt{f}_1, \ldots, \mathtt{f}_{m-1}\}) \cup \{\{\mathtt{f}_m\}\} & \text{if } \sum_{S \in clusters(\{\mathtt{f}_1, \ldots, \mathtt{f}_{m-1}\})} spot_{zone}(\mathtt{f}_m, S, 0) \\ \bigcup_{S \in clusters(\{\mathtt{f}_1, \ldots, \mathtt{f}_{m-1}\})} \{add(\mathtt{f}_m, S)\} & \text{else} \end{cases}$$

The auxiliary function *add* adds a flaw $\mathtt{f}$ to a "HotZone", that means, a set of flaws S, if $\mathtt{f}$ overlaps with some flaw in S:

$$add(\mathtt{f}, S) = \begin{cases} S & \text{if } spot_{zone}(\mathtt{f}, S, 0) = 0 \\ S \cup \{\mathtt{f}\} & \text{else} \end{cases}$$

Finally, the plan selection function $f^{planSel}_{FewerHZones}$ can be defined as

$$P_i < P_j \in f^{planSel}_{FewerHZones}(P_1, \ldots, P_n) \text{ if } |clusters(flaws(P_i))| < |clusters(flaws(P_i))| \tag{4.32}$$

The HotZone strategies perform in general very well in all of our experimental set-ups, of course in particular for problems and domains where isolated threads of actions are the rule. Regarding strategy compositions, it has to be taken into account that they develop their potential only on literally "large" flawed plans. As a consequence, the size of the problem instance determines the moment at which the plans' data structures provide enough components to feed the HotZone algorithm properly and the fringe candidates become discriminable. For problems with small solutions (few tasks and constraints), this horizon is typically never reached. It has also to be mentioned that all presented HotZone strategies are able to guide search on the last refinement steps towards a solution. While it is trivially given that a solution has no HotZone at all, it has also to be considered that with the "last" flaws getting eliminated, the HotZone metric drastically changes in favour of the close-to-solution candidates.

4.1.6 Miscellaneous Strategies

This section is dedicated to two selection functions that do not perform a strategic reasoning in the narrow sense but take one of the previously defined strategies and post-processes its results.

As it has been mentioned above, it is not trivially decidable for some strategies whether to use them in the way they have been defined or to prefer their opposite choices. For instance, it makes sense to explicitly prefer one modification class as well as explicitly evading another class. To this end, we define the *inversion* strategy, which is defined as a modification selection by the following equation: Let $f_{\text{inner}}^{modSel}$ be the modification selection function that is to be inverted, then

$$\begin{aligned} \mathtt{m}_i < \mathtt{m}_j \in f_{f_{\text{inner}}^{modSel}{}^{-1}}^{modSel}(P, \{\mathtt{f}_1, \ldots, \mathtt{f}_m\}, \{\mathtt{m}_1, \ldots, \mathtt{m}_n\}) \\ \text{if } \mathtt{m}_j < \mathtt{m}_i \in f_{\text{inner}}^{modSel}(P, \{\mathtt{f}_1, \ldots, \mathtt{f}_m\}, \{\mathtt{m}_1, \ldots, \mathtt{m}_n\}) \end{aligned} \tag{4.33}$$

Note that if the inner modification selection function is undecided, so is the inverted selection. The inversion for plan selections is defined analogously.

$$\begin{aligned} P_i < P_j \in f_{f_{\text{inner}}^{planSel}{}^{-1}}^{planSel}(P_1, \ldots, P_n) \\ \text{if } P_j < P_i \in f_{\text{inner}}^{planSel}(P_1, \ldots, P_n) \end{aligned} \tag{4.34}$$

A second "meta strategy" concerns all selection functions that are based on a (quantitative) metric. In our experimental studies it turned out that most of these functions, for example, the S+OC plan selection, can hardly be combined with other strategies because they typically produce a complete linear ordering on the presented options. On the other hand, the actual numerical differences are often very small, sometimes they are only arithmetic artefacts that are caused by the actual algorithm's implementation. We therefore propose a simple *clustering* strategy that takes the metric function of a respective modification or plan selection and sets up equivalence classes according to the sample mean distances that are produced by the metric. Let $f_{\text{inner}}^{planSel}$ be a plan selection strategy that is based on a metric g, and let *linearize* be the known linearization function for partially ordered items, then the clustering strategy for this selection is defined by the following equation:

$$\begin{aligned} P_i < P_j \in f_{\overline{\Delta} f_{\text{inner}}^{planSel}}^{planSel}(P_1, \ldots, P_n) \\ \text{if } P_i < P_j \in f_{\text{inner}}^{planSel}(P_1, \ldots, P_n) \\ \text{and } |g(P_i) - g(P_j)| < \frac{\sum_{P'_x, P'_{x+1} \in linearize\, f^{planSel}(P_1, \ldots, P_n)} |g(P) - g(P')|}{n-1} \end{aligned} \tag{4.35}$$

The clustering strategy thereby prefers all options to the same degree if their evaluation differs below average according to the embedded metric. A definition for a modification selection with the clustering method can be given analogously.

4.2 Discussion

4.2.1 General Comments

With such a great emphasis on the flexibility of the framework and its planning strategies, the question naturally arises what the the price is for providing high-quality search control in such a "verbose" architecture in which everything is explicitly represented. The answer is that dealing with every option explicitly is of course a costly procedure per se, but put into perspective of the gained predictive power and simplicity of deployment, the actual costs are satisfactory. The strategic "overhead" can be divided into two sections:

1. Computing the decision of a compound strategy, that means, aggregating the partially ordered options can be done with a computational effort that is cubic with respect to the input options. The factors in the polynomial bound are very small, because any subsequent strategy's result is only probed for elements in the same equivalence class. In practical applications with longer chains of selection functions, the last components are often not activated at all on large portions of the search space, because the primary strategy are decisive enough.
2. The computational complexity of most components is completely negligible (set size comparisons, and the like). The effort for the more complex computations in the HotSpot and HotZone heuristics is mitigated by caching the invariant results.

Related to the previous computation considerations is the question of being able to cut certain branches in the search tree. As it has been stated periodically, our philosophy is to conduct a *complete* search as long as it makes sense to the application area. There are, however, two cases in which *plan modifications* can be *safely discarded*: We will refer to these two cases as *unit modifications* and *unreachable modifications*.

Unit modifications are inspired by unit clauses in the resolution calculus: if there is exactly one plan modification issued for a flaw, this modification is executed immediately and all other options are discarded. This zero commitment rule is not compromising the solution space if the strategy is complete (which is the case for all our components). A minor misconception is to argue that this unit modification might produce the paradox of a premature commitment. For example, consider a plan with an open precondition, for which a new plan step is inserted "automatically" if there are no suitable condition providers available. The argument is that if the system "waited" a provider could appear elsewhere and then be re-used accordingly. The argument is however invalid because our approach can rely on the current configuration to produce always all available refinements and if there exists an alternative plan that corresponds to the described re-use scenario, the system will eventually discover that plan on a different refinement path.

The notion of an unreachable modification is derived from the idea that the modification selection functions are choosing those plan modifications that are considered appropriate *in the given order* for addressing the flaws in the current plan. Let $\mathtt{m}_1, \ldots, \mathtt{m}_n$ be a sequence of plan modifications obtained from a respective strategy call for a plan P and flaws $\mathtt{f}_1, \ldots, \mathtt{f}_m$. Let j_i be the minimal modification index $1 \leq j_i \leq n$ associated with a flaw $\mathtt{f}_j$ such that $\mathit{modsFor}(\mathtt{f}_j) \cap \{\mathtt{m}_{j_i+1}, \ldots, \mathtt{m}_n\} = \emptyset$. The minimum j of these indices is the index of the modification that is the very first in the sequence for which the addressed flaw is completely treated. If we respected the modification selection's choice we would traverse the refinement space according to this sequence. When we finally backtracked over the refinement obtained from modification j, we would in particular have exploited all flaw resolution options that were available for the respective flaw in P. As a consequence, the respective flaw persists over the subsequent refinements and hence there will be no solution beneath plan P. For this reason, every modification after the index j is regarded as unreachable and therefore no associated refinement is produced.

Another topic that arouse frequently in discussions on conferences and reviewers' feedback was the role of least commitment in refinement-based planning. The "bone of contention" lay in the preference of HotSpot, respectively HotZone selection functions: The avoidance of multi-flawed plan elements appeared counter-intuitive, since one would expect that concentrating on "common issues" would be comparably effective like it has turned out to be the case in constraint satisfaction. The DPLL search algorithm, for instance, gains its performance through dealing with the most constrained variables first. We believe to settle this issue with

two arguments. First, early empirical evidence showed that HotSpot affinity performs significantly worse than the presented preference and hence suggested that there is no point in applying an inversion strategy to it. This result can be explained by dependencies between the modification steps on different levels of plan development. As we noted before, the competing modification proposals introduce a significant branching as well as a high probability of backtracking. To put it short: choosing a resolution for a HotSpot/HotZone member commits indirectly on specific resolutions for fellow members.
Our second argument concerns the misconception, as we understand it, of the relationship between the least commitment principle in planning and the classical DPLL heuristic in constraint satisfaction solving. There is in fact no conceptional difference between "selecting the most constrained variable" and the f_{LCF}^{modSel} heuristic "selecting the modification with the least alternatives" since there is a subtle difference between a focusing on *constrained* results and *constraining* refinements. In view of the original DPLL, and this caused some confusion, both notions coincide. A closer look on the HotSpots' "selecting the least connected modification" principle reveals that it corresponds to some extend the efforts in the CSP community to identify independent partitions of a constraint net.

A last general remark has to be made about the component repertoire comprising flaw- and modification-based variants of most selection functions at the same time. The predictive power of plan modifications (in HotSpots, etc.) is in general more precise than that of flaws, because we can literally look into the forthcoming changes and assess their conflict potential directly. However, this projection is basically not only limited to the next level of refinement but also producing to many false-positive interactions. Both properties are attributable to the circumstance that plan modifications typically carry two classes of elementary modifications: the first ones are essential for the plan modification and the others are semantic "glue" that adapts the modification to the plan context. An example is the expansion modification, which contains not only all elements of the expansion network but also additional constraints for preserving causal commitments, and the like. It is these adaptation elements that produce disagreeable overlapping artefacts. The flaw perspective is however mostly context-invariant, for example an open precondition, if un-solved, remains syntactically the same. Problems persist, but solutions change.

4.2.2 Modelling Classical Strategies

Dealing with the specifics of search control has always been and still is a central topic in the area of Artificial Intelligence planning and scheduling. However, may it be because of a focus on other representations and methods or may it be because of its huge amount of freedom, to our knowledge only very few research has been devoted to search strategies for hybrid planning and scheduling. In addition, even the existing strategies have never been systematically compared and assessed in this context. We believe that this is because of two reasons: Firstly, as we just mentioned, the space of potential strategy implementations that covers the presented system configurations is several orders of magnitude larger than for one paradigm alone. The proposed strategy components of the preceding section alone can be composed into billions of plan and modification selection triples, and this makes a complete investigation only on a handful of domains virtually impossible. The second reason is that the search strategy of practically all existing planning systems is defined only implicitly, an integral part of the actual plan generation algorithm and strongly dependent from the supported plan representation. As a consequence, changing one of them inevitably affects the other and handicaps any systematic strategy evaluation conceptually as well as from a software technological point of view. It is in particular the second reason that leads to the unsatisfactory perspective that an empirical analysis will ever be attempted.

But what can be learned from the existing work on universal, that means, not system-specific planning strategies and how can it be applied to hybrid planning and scheduling?

The principle of identifying deficiencies and computing resolution steps is a key characteristic in partial-order planning algorithms. Their procedures for navigating through the plan-space decide first on the flaw to solve and then on the appropriate resolution method. In order to do so properly, plan-space planners like UCPOP [214] and SNLP [177] have to address flaws according to a systematic schema. One way to define it is for example the following:

> "The PoP procedure has a distinct control for sub-goals and for threats. At each recursion, it first refines with respect to a sub-goal, then it proceeds by solving all threats due to the resolver

> of that sub-goal. Consequently, there are two non-deterministic steps: (i) the choice of a relevant action for solving a sub-goal, and (ii) the choice of an ordering or a binding constraints for solving a threat."
>
> [118, Section 5.4.2] and [279]

The same strategy types (with different names and labels) occur even in very general refinement-based frameworks like [148], where the various refinements are tackled in a given ordered structure (and can be seen as combined flaw finding and resolution methods). The referenced framework, for example, first performs a goal establishment phase and in a second phase conflict resolution with a preference on adding ordering constraints.

We call such strategies *in-flexible* and they are implemented in most of todays partial-order planning systems. The area of least-commitment planning made an effort to provide a broader theory for setting up more general preference schemata and for predicting their performance. Their research is concerned with studying the performance of strategies as well as to some extent how these schemata can be reasonably combined. While there has been an extensive amount of work in the field of partial-order planning, including [29, 143, 177, 215, 218, 236, 275, 297], only few studies have been published for HTN planning, for example [86, 101, 268, 271],[6] not to speak of hybrid planning [180, 298]. All these approaches explore in great detail how specific flaw and modification combinations can be classified. They distinguish, for example, causal threats that are resolvable by variable separation from those that are not. Also the priority within a flaw class becomes important, for instance, the ordering of open precondition deficiencies. We believe that at this stage such a close focus on configuration-specific flaw and modification sets contradicts the claim of flexibility of our strategies and in particular hampers their applicability across system configurations. The situation becomes even more obscure if the context-dependent ambiguity of plan modifications with respect to the commitment consequences is taken into account. For example, Veloso and Blythe reported in [273] of a direct correspondence of commitment to causal links in partial-order planning and ordering constraints in total-order planning. We therefore leave our most specialized selection functions at the class-specific level. A supposable extension of our strategies to more specific treatment of plan modifications is, for example, adding more reasoning capabilities for ranking constraint-set manipulations [86].

Let us briefly examine how some key strategies from the literature can be expressed in our framework. Each of them has been implemented in a leading hierarchical planning system.

The first candidate is the strategy of the HTN planning system UMCP. The originally published strategy only considered the systematic expansion of abstract tasks, while all executability checks and variable bindings were left to a closing constraint reasoning process [85]. We can therefore simply say that

$$f^{modSel}_{UMCP} = f^{modSel}_{Pref-\mathbb{M}_{\texttt{ExpandTask}}} \tag{4.36}$$

Regarding the plan selection, the authors proposed elsewhere [83] three procedures: depth-first, breadth-first, and a best-first heuristic over the number of non-primitive tasks in the plan. While the first two are trivially the previously introduced components (Def. (4.1) and 4.2), the suggested enhanced plan selection function can be described as follows:

$$f^{planSel}_{UMCP} = f^{planSel}_{Addr-\mathbb{F}_{\texttt{AbstrTask}}{}^{-1}} \tag{4.37}$$

Later work on UMCPincorporated the previously mentioned *fewest alternatives first* strategy (FMF) for modulating the expansion of abstract tasks: select for expansion that task for which the minimal number of alternatives is available [271]. It has to be noted that the term "alternatives" is here interpreted over the transitive closure of method applications, that means, all methods are applied off-line on all abstract tasks thereby inducing an AND/OR tree that displays all constructable task expansions for reference. It is our understanding that this, however, gives no deeper insight into the actual applicability of the expansion alternatives and that therefore our abstraction (only probing the given modifications) is appropriate. Furthermore, the FMF

[6] It has to be noted that HTN strategies as such cannot be compared to those for non-hierarchical planning. This is due to the expressive power and system-specific extensions of expansion methods: Most systems support some sort of tagging or programming facility that influences the algorithms search strategy directly. This ranges from annotating conditions [258] to specifying the succession of variable bindings [199].

heuristic implies a strong interdependency between the refinement generator (computing the expansions) and search control. Our interpretation of the advanced UMCP strategy is consequently

$$f^{modSel}_{UMCP+} = f^{modSel}_{Pref-\mathbb{M}_{\texttt{ExpandTask}}} \triangleright f^{modSel}_{LCF} \tag{4.38}$$

with an un-altered plan selection.

The remaining improvements finally addressed the question, in our terminology, how to deal with plan modifications that bind variables. It is reasonable to presume that this translates into the preference of `OpenVarBind` flaws and does not refer to other modifications that introduce variable constraints as a required part of their refinement. The proposed commitment strategies are:

> (1) a strategy that delays variable bindings as much as possible; (2) a strategy in which no non-primitive task is expanded until all variable constraints are committed; and (3) a strategy that chooses between expansion and variable instantiation based on the number of branches that will be created in the search tree.
>
> [268]

We think that the following realizations are fairly straight-forward translations into our strategy repertoire and demonstrate the elegance of our component chaining:

$$\begin{aligned}
f^{modSel}_{UMCP+(1)} &= f^{modSel}_{Addr-\mathbb{F}_{\texttt{OpenVarBind}}{}^{-1}} \triangleright f^{modSel}_{Pref-\mathbb{M}_{\texttt{ExpandTask}}} \triangleright f^{modSel}_{LCF} \\
f^{modSel}_{UMCP+(2)} &= f^{modSel}_{Pref-\mathbb{M}_{\texttt{AddVarConstr}}} \triangleright f^{modSel}_{Pref-\mathbb{M}_{\texttt{ExpandTask}}} \triangleright f^{modSel}_{LCF} \\
f^{modSel}_{UMCP+(3)} &= f^{modSel}_{Pref-\mathbb{M}_{\texttt{ExpandTask}\cup\texttt{AddVarConstr}}} \triangleright f^{modSel}_{LCF}
\end{aligned} \tag{4.39}$$

The choice between preferring a flaw versus the associated modifications is a subtle one. In variant (1) we intend not to postpone variable bindings that are, for example, introduced by threat resolution steps, while the rationale of variants (2) and (3)'s modification preference is that all kind of reason for binding the variable is accepted.

In the context of the UMCP system we have to mention a strategy that deals with an issue that most HTN systems have to address: which condition or constraint establishers are necessarily outside the expansion network and which will eventually occur inside of it. O-PLAN, for comparison, offers an explicit modelling mechanism to indicate which conditions are expected to be provided from outside the task network [258]. UMCP was in contrast provided with the ExtCon strategy, the "external conditions task selection" strategy, that performed even better than FAF [269]. External conditions are thereby not specified explicitly be the user but instead are found automatically by the planning system when it pre-compiles its knowledge base. The strategic algorithm can be sketched as follows: when selecting a task to decompose, priorities are given to (1) tasks that can possibly establish the current top condition or (2) tasks which can possibly threaten the current top condition. During method instantiation, the external conditions of the method are pushed onto the top of the applicability condition stack. This top element is the current priority to the planner. We note that this technique becomes obsolete in a framework that provides task on all levels of abstraction with preconditions and effects.

Another well-established strategy for hierarchical and in particular hybrid planning systems is called *expand-then-make-sound* and as such an implementation of the OCL_h approach [180]. It is described as a strategy executing two alternating modes: In the first phase exactly one abstract plan step is decomposed. In every second phase, the causal structure of the plan, respectively OCL_h's equivalent to it, is completed and problems regarding causal interactions are fixed. The alternation is realized in our framework by simply preferring threat resolving and open condition closing modifications over task expansions. As a consequence, a task decomposition is only chosen if the plan is "sound".

$$f^{modSel}_{EMS} = f^{modSel}_{Addr-\mathbb{F}_{\texttt{Threat}}} \triangleright f^{modSel}_{Addr-\mathbb{F}_{\texttt{OpenPrec}}} \triangleright f^{modSel}_{Pref-\mathbb{M}_{\texttt{ExpandTask}}} \tag{4.40}$$

The rationale for using a modification-oriented strategy rather than a flaw-oriented one is that we try not to evade those expansions that have been issued for threat resolution, etc. (see hybrid configuration descriptions in Chapter 3). Concerning the plan selection, we believe that the EMS behaviour is adequately represented by the modification selection alone and hence, any depth-first like plan selection is appropriate. We would

also like to point out that this strategy also covers the strategy of the hybrid planning approach proposed in [44]. As a minor deviation, we do not impose a strict stratification on the actions in our domain models and hence do not exactly decompose in a stratum-like manner.

The last showcase is the well-known hierarchical planning system SHOP (see also Sec. 1.1.3). It has been successfully deployed in numerous application domains and for that reason its strategy came naturally into our focus. SHOP's specific characteristic is to expand abstract tasks in the order in which they are to be executed [200]. One implication of this procedure is that the current world state can easily be tracked and updated, beginning from the initial situation until the task that is to be decomposed next. This gives much expressive power to SHOP's formalism, because a wide range of computations can be employed to evaluate the state before the execution point of the "current abstract task". As it has been discussed before, there are however arguments for adopting such a principle also in partial-order planning scenarios because of the constraining influence of the (complete) initial state description (cf. Sec. 1.1.1). The SHOP planning strategy can be modelled by the following strategy sequence (again, all particular SHOP-like strategic reasoning is performed already in the modification selection):

$$f_{SHOP}^{modSel} = f_{\prec}^{modSel} \triangleright f_{Pref-\mathbb{M}_{\texttt{ExpandTask}}{}^{-1}}^{modSel} \tag{4.41}$$

The early flaw preference (Def. 4.21) is the key heuristic and as the primary strategy ensuring that all flaws on the leading sections of the plan are addressed first. Since expansion modifications are devalued, this leads to a primacy of a "make leading tasks sound, then expand" principle. Since we allow for partially ordered network specifications, this strategy makes our system more like the SHOP version as described in [202].

It has to be stressed that these showcases are reduced to their more general search-control aspects. Every original approach has its own dedicated plan generation principle, application domains, and, last but not least, algorithmic and representational optimizations. The latter could of course neither be considered when defining an imitating strategy nor was this intended in the first place. This is in particular the case for the OCL_h approach that draws a lot of its performance from the state automata transformations and for the SHOP system that includes so much domain-dependent heuristics in its expansion methods that many people consider it to be a problem-solving programming-language (see respective introductory discussion in Sec. 1.1.3).

4.2.3 How To Build A Successful Strategy

The main challenge is obviously to find an efficient as possible search strategy for the domain and problem at hand. This is not trivial and our experimental evaluation (Chap. 6) is merely the "tip of the iceberg" and covers only a tiny fraction of the combinatorial possibilities of sequenced strategy compounds. In practice, no strategy component turned out to be obsolete or contra-productive and this leaves the system designer with the responsibility to determine the best[7] strategy for his/her application. However, there are some domain-independent directives one should adhere to in order to obtain a promising strategy candidate.

Elucidate Configuration-Specifics

The first key point can be deduced from our experiences with (the limitations of) inflexible strategies: All strategy components in all selection functions have to comply with the repertoire of flaw and plan modification classes that are issued by the system configuration's detection and modification generating functions. If a strategy depends on a specific flaw or modification class that never occurs in the current configuration, it will constantly make no or wrong decisions, depending on the selection definition. It is also worth noting that the inverse case, that means, the situation in which a strategy definition does not cover a flaw or modification type is problematic as well. In particular, if a flaw or modification type is not considered

[7] For the time being, we regard one strategy to be *better* than another one, if and only if it takes less search nodes to find the first solution. The objective may however change with the application, for example a solution quality metric has to be optimized, and the like. Please note that the considerations in this section basically remain valid.

by the (inflexible) strategy, the strategy's performance becomes typically un-predictable, since a problem characteristic is apparently overlooked.

Focus On Primary Strategy

The primary strategy components have to be selected most carefully and this holds for modification selection as well as for plan selection functions. An ideal (primary) strategy would prefer in every search node exactly the solution path and thus the three essential factors for choosing a primary strategy are:

1. how many wrong decisions are made,
2. how many options are undecided, and
3. are solutions postponed?

Because no preference decision can be overruled by subsequent strategies, a wrong decision cannot be counterbalanced. This result alone is obvious, less trivial is however to estimate the amount of wrong decisions. One guideline are experiments in the application domain in order to determine the optimum the primary strategy can achieve over a number of settings. If the number of experiments is sufficiently large, the measured optimum is close to the quality bound of the (combined) strategy and the subsequent components can only reduce the variance. An empirically sound method is however not always tractable and the application designer has often to use her/his experience instead.

The number of undecided options often goes hand in hand with the first aspect because a more relaxed preference is likely to make less wrong decisions by leaving more options un-ordered. The more options are undecided, the more we can infer from subsequent strategy components but also the more unstable the combined strategy's performance can become. We believe that it is advisable to risk stability for the sake of evading wrong decisions and to balance the strategy with suitable subsequent components. This motivated, for example, our clustering strategies (4.35).

Last, but not least, it has to be verified that the primary component is able to do the last mile, that means that it does not postpone close-to-solution plans. This is, of course, also an issue for subsequent strategies but it is a particular problem of the primary components and the worst case scenario with respect to factor 1.

Consider Strategy Interdependencies

If the primary strategy is appropriate, that means, if the modification selection function is delivering a reasonable selection of options from which the plan selection functions is able to choose a good refinement, it is the task of the subsequent strategies to improve the composition. Improving the performance is however nothing else but improving the strategy's stability, that is to say, reducing the variance that is caused by the randomisation of undecided options and the variations in the problem characteristics. From these considerations follows that the interdependencies of the strategy composition have to be considered carefully such that subsequent decisions:

1. do not too frequently contradict primary decisions (which reduces improvement),
2. do make decisions on undecided options, and
3. do narrow down the decisions in a way that is solution-oriented.

For the time being, all three points are subject to experience of the application designer and have to be checked pair-wise for all deployed components. There are of course obvious contradictions in the strategy compositions, like the preference and avoidance of the same class of flaws. Many practically relevant interdependencies are however hard to identify and some of them have been mentioned above in the component definitions. In general it is also particularly important to pay attention to the interplay of modification and plan selection. If their orientation is identical, a pure depth-first schema emerges; if their orientation diametrically opposes, search becomes implicitly breadth-first. We may assume that neither case is generally intended.

As a general recommendation for compositions, we propose to sequence modification-based components after flaw-based ones. It appears that flaws exhibit a superior predictive power, which qualifies the respective components for a prioritized position. Since then the primary selection focus on problems and the subsequent on solution proposals, contradictions are less probable and subsequent decisions become likely available (multiple modifications are issued for one flaw).

4.2.4 Perspective

The previous sections have covered a number of search control aspects, but the presented solutions can only be considered a first step on the road to a complete coverage of the planning framework's capabilities. We have mentioned the use of iterative search variants and the A^* algorithm, but there is a large number of traditional Artificial Intelligence search techniques that is yet to be reviewed [158]. We believe that our framework is general enough so that all of them can be in principle deployed. Promising future strategy candidates include the following: learning strategies that are trained on small problem instances of a domain for inferring branching factor estimates from encountered flaw-modification pairs (cf. [35]); strategies that concentrate on those flaws, respectively HotSpots or HotZones, that refer to the most recently introduced plan components (cf. *local flaw selection* [297]); strategies that focus on those flaws that refer to rigid symbols (cf. *static-first strategy* [296]); symmetry detection and avoidance.

It has also to be noted that the presented strategy material is dominantly oriented towards the characteristics of (hybrid) planning and does not take into account yet the specifics of resource reasoning. Although practically all of our components are formulated completely in an configuration-independent way (the A^* variants, for example, are exceptions that have to be adapted accordingly), we obviously have to develop an appropriate strategic advice for the quantitative metrics that come into play when dealing with resource manipulations. An interesting line of research may be HotSpot and HotZone calculations that are not only defined in terms of commonly referenced plan elements but also take into consideration to which amount a resource is collectively allocated. Other worthwhile directions in this context are investigating into resource-profile balancing methods, multi-objective strategies, and, last but definitely not least, optimization of resource usage.

The proposed sequential composition is of course only *one* possibility of combining individual preferences. Multi-agent research and work in distributed problem solving provide a large number of approaches for distributed decision making [278, Chap. 5]. Our framework represents a programming-oriented view on the strategies and the sequential aggregation is therefore an obvious combination method. Although our strategic potential is already unmatched, we think about extending our strategy composition to consensus strategies (an option is preferred if it is preferred by all strategies), favorite strategies (an option is preferred if it is preferred by at least one strategy), and even agent-like concepts like voting and auctioning. All of these extensions are directly implementable in the framework.

Another interesting area of planning strategies will be opened by introducing the concept of *dynamic configurations* (see also Sec. 3.5.5). As correctness of the planning algorithm cannot be compromised by the strategy, we can think of *changing* the strategy at run time, that means, during the plan generation process. The are several triggers for changing a strategy imaginable, for example a plateau in the fringe's plans' quality, an increasing undecidedness of the plan selection function, and the like. Switching between strategies is also an issue when deploying an algorithm in the fashion of least discrepancy search [287]. This topic will involve focusing on the role of the solution selection function, which is not in the scope of this thesis. During the collection of multiple solutions, this function will play a major role in optimization applications, etc. For example, it may cause a strategy change due to a plateau or decrease of solution quality.

The probably most pressing issue for future research activities is concerned with identifying the appropriate strategy compositions subject to domain model and planning problem characteristics. We believe that there is a substantial amount of work lying ahead to which the methods of empirical studies will be central. The corresponding "perspective" section of a later chapter that is dedicated to an experimental strategy evaluation (Chap. 6) will address this topic in more detail.

4.3 Summary and Conclusion

This chapter has presented a comprehensive repertoire of plan and modification selection functions that can be sequenced into more complex strategy arrangements. We have shown several families of components; how their approach is motivated, how they are defined and implemented in the planning framework's context, and what their contribution to our strategy canon is. We could in particular demonstrate how all relevant strategic aspects of key approaches in the field are entirely reproducible by our component-based strategies.

Furthermore, we clearly extended the state of the art by the novel HotSpot and HotZone techniques. Both methods go beyond every known pre-defined schema, respectively rule-based approach in the area of planning and scheduling. Together with our ability to model practically every single classical strategy, this gives further evidence to the potential and expressive power of our refinement planning framework.

It has to be stressed that all presented strategies are complete in the sense that no refinement option is ignored (cf. Sec. 2.8), they are domain independent, and they are even configuration independent (some minor exceptions are discussed above). We are therefore confident to have set up a rich catalogue of efficient strategies for a broad range of application scenarios. Chapter 6 will adopt some strategy instances and evaluate them experimentally in several domains.

5 Implementing Planning and Scheduling Applications

An AI planning and scheduling application has to be understood as the unity of a plan generation software, a domain model, and the appropriate problem specifications. This chapter is consequently dedicated to the two main aspects of fielding a planning system, namely the question of implementing the formal framework as a concrete software artefact and the issues that arise when modelling concrete application domains.

The first section will present the PANDA (Planning and Acting in a Network Decomposition Architecture) prototype, an architecture for planning and scheduling systems that addresses key requirements of real-world applications in a unique manner [229, 230]. The system provides a robust, scalable and flexible framework for planning and scheduling software through the use of industrial-strength middleware and multiagent technology. The architectural concepts extend knowledge-based components that dynamically perform and verify the system's configuration. The use of standardized components and communication protocols allows a seamless integration with third-party libraries and existing application environments.

The software-technical treatment is followed by sections that describe three concrete domain models. We will not only present the model components, that means, the underlying language, the task schemata, etc., but also explicate their design rationale and describe possible alterations. Sections 5.2.1 to 5.2.3 present in this sense a satellite observation scenario, a logistics application, and an artificial domain that has been designed for strategy evaluation purposes.

We finally discuss planning-related software architectures, respectively document our experiences from constructing the application show cases. Some remarks on future research directions in these contexts conclude the chapter.

5.1 System Implementation

While Sec. 2.7 presented an architecture that made theoretical framework operational in a straight-forward system design, a number of functional and non-functional requirements are obviously not met by such an architectural nucleus when it comes closer to real-world application scenarios like crisis management support, assistance in telemedicine, personal assistance in ubiquitous computing environments, and the like. Like any other mission critical software in these contexts, planning and scheduling systems should feature characteristics that call for highly sophisticated software support:

1. declarative, automated system configuration and verification – for fast, flexible, and safe system deployment and maintenance, and for an easy application-specific configuration tailoring
2. scalability, including transparency with respect to system distribution, access mechanisms, concurrency, etc. – for providing computational power on demand without additionally burdening system developers
3. standards compliance – for integrating third-party systems and libraries, and for interfacing with other services and software environments

Each of these characteristics represents a challenge in its own for any software environment, and this is in particular the case for planning and scheduling applications. This section describes a novel planning and scheduling system architecture that essentially addresses all of the above challenges, and shows how our

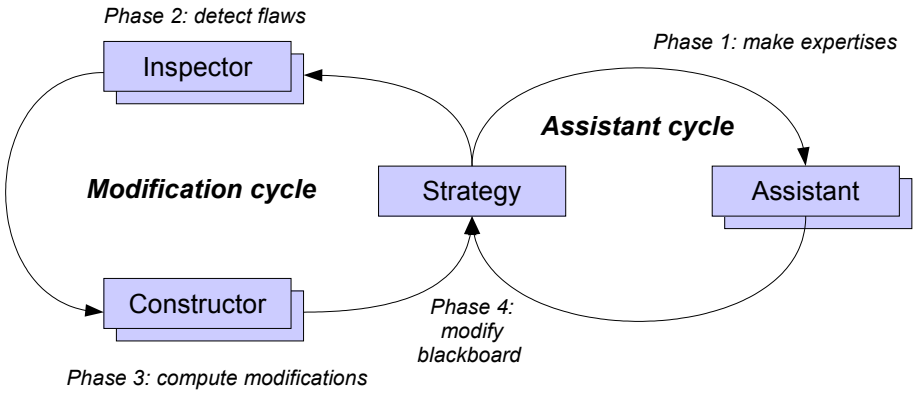


Figure 5.1: The reference planning process model for PANDA.

formal framework has been incorporated. It shows not only how modern software technology –in particular middleware and knowledge-based systems– can be successfully applied to a prototypical academic planning software, but also illustrates how (in principle) any planning and scheduling system can benefit from it. The resulting system performs a dynamical configuration of its components and even reasoning about the consistency of that configuration is possible. The planning components are transparently deployed, distributed (including an optimized concurrency), and load-balanced while retaining a relatively simple programming model for the component developer. Standardized protocols and components finally provide easy access to other software products and services.

5.1.1 Architecture Overview

In following the proposed design of the generic refinement planning algorithm (Sec. 2.7), the basic architecture of PANDA corresponds to a multiagent-based blackboard system [208]. The agent societies map directly on the presented function structure, with the agent metaphor providing maximal flexibility for the implementation.

- *Inspectors* are implementations of flaw detection functions. There is one such agent per flaw class.
- *Constructors* are agent incarnations of the plan modification generating functions. We may assume that each modification class is represented by one such agent.
- *Assistants* represent the inference functions and provide shared inference and services that are required by other agents. Assistant agents propagate implications of temporal action information transparently into the ordering constraints, simplify variable constraints, etc.
- *Coordinators* implement the planning strategy functions by synchronizing the execution of the other agents and performing the modification selection. Currently only one coordinator is allowed in the system and we call it the *Strategy*.

Figure 5.1 shows the reference planning model for PANDA, which defines the agent interaction. A planning *cycle* corresponds to an iteration of a monolithic algorithm (cf. Alg. 2.2). It consists of two sub-cycles: both are divided into 4 phases (see Fig. 5.1) in which the agents execute concurrently.

Phase 1: Assistants repeatedly derive additional information and post it on the blackboard. This phase ends when no member of the assistant community added information anymore.

Phase 2: Inspectors analyze the current plan residing on the blackboard and post the results, that means, the detected and prioritized flaws, to the Strategy agent and to the constructors assigned to them.

Phase 3: Constructors compute all possible modifications for the received flaws and send them along with a prioritization to the Strategy agent.

Phase 4: The Strategy, respectively its incorporated modification selection, compares all results received from the inspectors and constructors and selects one of the modifications to be executed on the current plan. A planning cycle is hereby completed and the system continues with phase 1 to execute the next planning cycle.

Phase transitions are performed only by the Strategy agent when all participating agents have finished execution. Thus, the phase transitions can be viewed as synchronization points within the planning process. In fulfilling the contract of the generic planning algorithm, the Strategy agent modifies the plan until no more flaws are detected or an inspector published a flaw for which no resolving modification is issued. In the first case, the current plan constitutes a solution to the given planning problem, in the latter case the planning process has reached a dead end and the system has to discard the current plan and turn to an alternative plan in the fringe of the search space – we call this a *backtracking* step, since the consecutive plan under observation is not obtained from the previous one. The blackboard provides random access to the plans in the search space, where each plan is stored together with all its derived information and performed modifications. This structures enable the Strategy agent to navigate the system to any point in the explored plan space it finds promising.

The following sections will show how the reference planning process model has been implemented by using middleware and knowledge-based technology. The chosen multiagent-system is based on on an industrial-strength middleware and uses an explicit knowledge representation in the implementation of the necessary protocols. A refined version of the reference model will then allow us to exploit agent concurrency more efficiently.

5.1.2 A Knowledge-based Middleware

Core Components

An obvious implementation for a planning system following the reference process model would still run in a sole JAVA virtual machine, that is to say, on a single computational resource. This stands in contrast to the requirements that complex and dynamic application domains demand. For crisis management support, for example, information must be gathered from distributed and even mobile sources, the planning process requires a lot of computational power, etc. So scalability and distribution play key roles in the proposed system architecture, while maintaining the (simple but effective) reference process.

The main aspect in middleware systems like application servers is to hide the mechanisms that enable the distributed handling of objects from the programmer. Thus, it is possible to develop distributed applications much more efficiently. In other words, such middleware systems make distribution issues *transparent* to the programmer. Examples for transparency in middleware systems are location transparency, scalability transparency, access transparency, concurrency transparency, and the like [82]. Scalability transparency, for instance, means that it is completely transparent to the programmer how a middleware system scales in response to a growing load. In summary, middleware systems take care of the complexity of handling distributed objects and provide an abstract and easy to use Application Programming Interface (API) to the programmer.

In order to benefit from application server technology, the PANDA system builds upon the open-source implementation JBoss [245]. The most important components that a JAVA *Platform, Enterprise Edition* (JAVA EE) [251] based application server delivers with respect to this work are the following:

- *Enterprise* JAVA *Beans* (EJB) are the objects that are managed by an application server. All transparency aspects apply to them. They are the building blocks of a distributed JAVA EE application [252].
- The JAVA *Naming and Directory Interface* (JNDI) is the directory service that enables location and access transparency. It provides a mapping between JAVA names and remote interfaces of JAVA objects. The access to all EJBs and other services of the application server is provided through this interface.

- The JAVA *Messaging Service* (JMS) enables asynchronous and location transparent communication between JAVA components (especially EJBs) beyond virtual machine boundaries. So this service will be of interest when it comes to communication between the different components of PANDA.

In addition to the features described above, application server implementations also cover aspects like security, database access, transaction management etc. They all belong to the JAVA EE specification. However, a full discussion of their benefits for the PANDA system is beyond the scope of this thesis.

Although the application server technology provides powerful mechanisms, we still need more support for realizing the multiagent system functionalities of the reference model. Instead of investigating a proprietary agent life-cycle management and appropriate communication mechanisms, we decided to take advantage of the work of the *Foundation for Intelligent Physical Agents* (FIPA), which has been developing standards for that area since 1996. The second core component that is chosen to implement all agent specific features, that means, to take care of the agent computing capabilities of the system, is therefore BlueJADE [63]. BlueJADE integrates the well-known multiagent framework JADE (JAVA *Agent DEvelopment Framework*) [20] with JBoss. This integration puts the agent-system life cycle under full control of the application server, in particular all distribution capabilities of the application server thereby apply to the agent societies. Access to the JADE agent API is provided by BlueJADE's `ServiceMBean` interface. It has been selected as the agent computing platform for PANDA because of the following key features:

- It is a FIPA-compliant agent platform and provides a library of ready-to-use agent interaction protocols. This enables the PANDA system to interact with other multiagent-systems and their services.
- It is a distributed system, that means, its agents can be spread transparently over several agent containers running on different nodes in a network, including the migration of running agents between containers. These features can be exploited for distributed information gathering and (automated) load balancing. It has to be noted that this kind of distribution management is "on top" of the middleware facilities: agent migration typically anticipates *pro-actively* the computation or communication load in a relative abstract manner, while middleware migration *reacts* on such load changes based on very low-level operating system specific sensors. It makes sense to provide both mechanisms in parallel, for example, to migrate scheduling inspector agents, which are known to require much computational resources onto dedicated compute servers.
- The LEAP extension (*Lightweight and Extensible Agent Platform* [39]) adds support for ubiquitous computing. Agents are able to run even on mobile devices such as JAVA capable cellular phones, PDAs, etc., which are all coveted user-interfaces in many application domains.
- BlueJADE supports application defined content languages and ontologies.
- The system comes with a set of sophisticated graphical debugging tools. This speeds up the development process significantly.

The knowledge representation and reasoning facilities that are used throughout the system constitute the third core component. During its development, the PANDA framework required an increasing amount of knowledge that represents planning related concepts (flaw and modification classes, etc.), the system configuration (which inspectors, constructors, and strategies to deploy), and the plan generation process itself (the reference process, including the backtracking procedure, etc.). Most of this knowledge is typically represented implicitly through algorithms and data structures. To make it explicit and modifiable without touching the system's implementation, it must be extracted and represented in a common knowledge base. In addition, this knowledge base has to use a representation formalism that is expressive enough to capture all modeling aspects as well as it allows an efficient reasoning procedure. As a result, the system can be configured generically and this configuration can be verified on a higher semantic level.

There is a large number of knowledge representation systems available on the market that promise to meet the requirements. But since special regard is spent on standards compliance, the *DARPA Agent Markup Language* – DAML [139] has been chosen as the grounding representation formalism for this task. It combines the key features of description logics [10] with Internet standards such as the *Extensible Markup Language* (XML) or the *Resource Description Framework* (RDF) [174] and – even more important – powerful reasoners and other freely available tools exist to integrate the language into applications. In our case, the knowledge encoded in DAML must be made available to the JAVA programming language. Therefore, a JAVA object model is necessary that provides mappings in both directions – from JAVA to DAML and vice

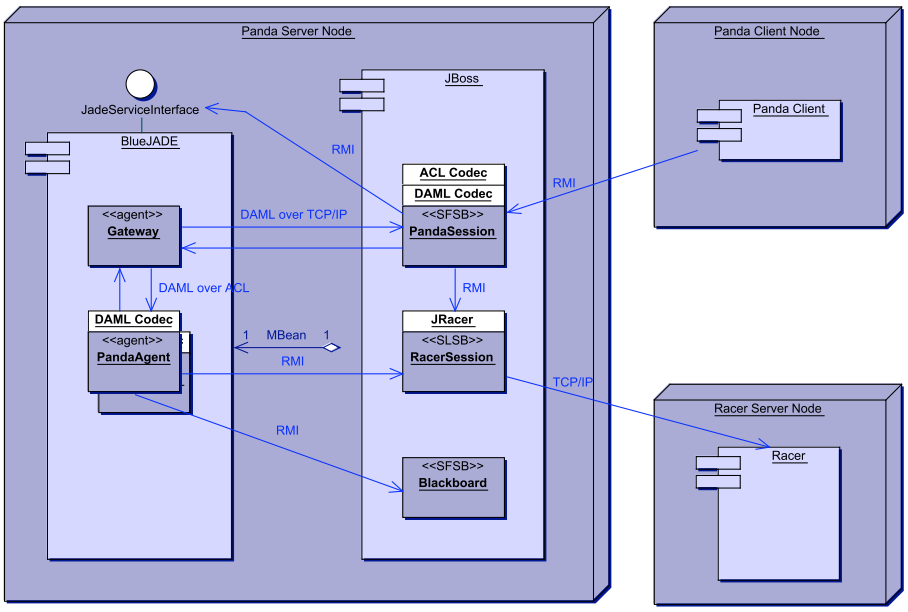


Figure 5.2: Static system structure of PANDA.

versa. The JENA API [178] from Hewlett Packard delivers an in-memory object model of a DAML document along with a rich API to query and manipulate it. By using DAML as the content language for the BlueJADE agent communication and also as the language for describing system configurations and communication means, we achieve a homogeneous representation in the system.

Last, but not least, it is of course necessary to integrate a suitable description logic system to store the knowledge and to reason about it. The RACER system [125] has the essential capabilities that are required: a DAML codec, an efficient reasoning component, and a knowledge store based on a client-server architecture.

The System Structure

Figure 5.2 shows PANDA's static system structure as a UML deployment diagram. The association arrows indicate which components communicate with each other. Their labels denote the transport protocols being used. BlueJADE is aggregated by JBoss as a (Service-) MBean. It's functionality is exposed via the `JadeServiceInterface`.

The PANDA *Client* component represents the client application that controls the PANDA system. Currently, an RMI[1]-based communication is used. The PANDA client obtains an interface to the PANDA system by querying JBoss's directory service JNDI. But also web-based approaches using SOAP[2] or simple HTTP are supported. In this way, JBoss provides technologies like Web Services and JAVA Servlets[3].

Regarding the integration with the JBoss infrastructure, the PANDA prototype defines three specializations of EJBs for non-agent system components: the interface to the RACER system, to the blackboard, and to the agent society (from outside the system).

Access to the RACER server is provided by the `RacerSessionBean`. The main reason for integrating the RACER system via an EJB proxy is that all depending components – EJBs and Agents – are

[1] JAVA Remote Method Invocation: see `http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp`
[2] Simple Object Access Protocol: see `http://www.w3.org/TR/soap/`
[3] Servlets are part of the JAVA EE technologies, see `http://java.sun.com/products/servlet/`

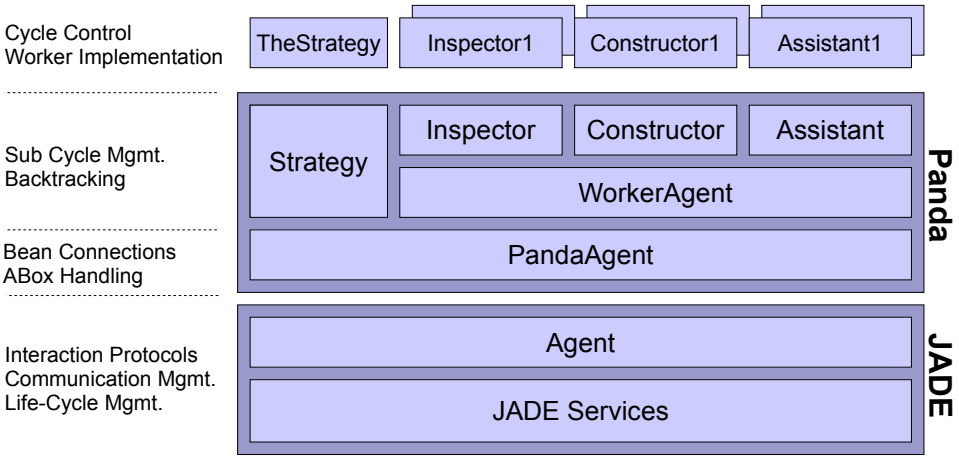


Figure 5.3: The logical layer structure of the agent framework.

able to access it transparently. They do not need to know its IP address or socket number. Furthermore, the `RacerSessionBean` can be viewed as generic integration approach for all kinds of reasoner architectures. The communication between RACER and the `RacerSessionBean` is realized with the *JRacer* client API, which translates JAVA method calls into Lisp function calls. It should be emphasized that each component that obtains a reference to the `RacerSessionBean` gets its own instance – as usual for `SessionBean` objects. Therefore, queuing of requests is delegated to the RACER server. In a similar fashion, the `BlackboardSessionBean` represents a proxy to the blackboard component.

The `PandaSessionBean` represents the facade[4] by which the PANDA client configures and controls the planning process. It uses the `RacerSessionBean` to derive the agents and their implementations that must be instantiated and creates them using the `JadeServiceInterface`. Communication with the agent framework is done via the `JadeBridge` class[5] of the BlueJADE package, which creates and accesses agent messages (see below) in an object-oriented manner.

Basically, two main *classes of agents* exist in the BlueJADE agent container. The first is the class of standard agents that come with the JADE and BlueJADE software packages. They provide the FIPA infrastructure, several debugging tools and JADE specific communication services. The *Gateway* agent's role is to mediate[6] messages between JADE agents and components outside the JADE agent container. It's counterpart in the EJB container is the `JadeBridge`. The Gateway sends and receives stringified messages in the *Agent Communication Language* (ACL) via a TCP/IP socket connection.

Custom agents in the PANDA system, that means, all agent types from the reference model, are derived from the `PandaAgent` class, which encapsulates low level data conversion and communication mechanisms. The PANDA agents form the second class in the JADE agent container. The `PandaAgent` class on its part is derived transitively from the JADE agent base class `Agent`, which provides the integration into the JADE agent container (cf. Fig. 5.3).

The reference model (Fig. 5.1) omits the actual means for *calling* agents; in a distributed implementation, these remote calls are typically based on message exchange. From the agent container's point of view, agents in the JADE agent container and the `PandaSessionBean` communicate by using messages encoded in the agent communication language *FIPA-ACL* [95] (in short ACL). ACL is a language based on Searle's speech-act theory, that means, every message describes an action that is intended to be carried out with that message simultaneously (for instance, the request "compute detections"). Such intentions are called *performatives*. ACL defines formal semantics for performatives [94] that induce basic interaction protocols upon which more complex protocols like contract nets and auctions are built.

[4] Cf. the Facade design pattern [103].
[5] Cf. the Bridge design pattern [103].
[6] Cf. the Mediator design pattern [103].

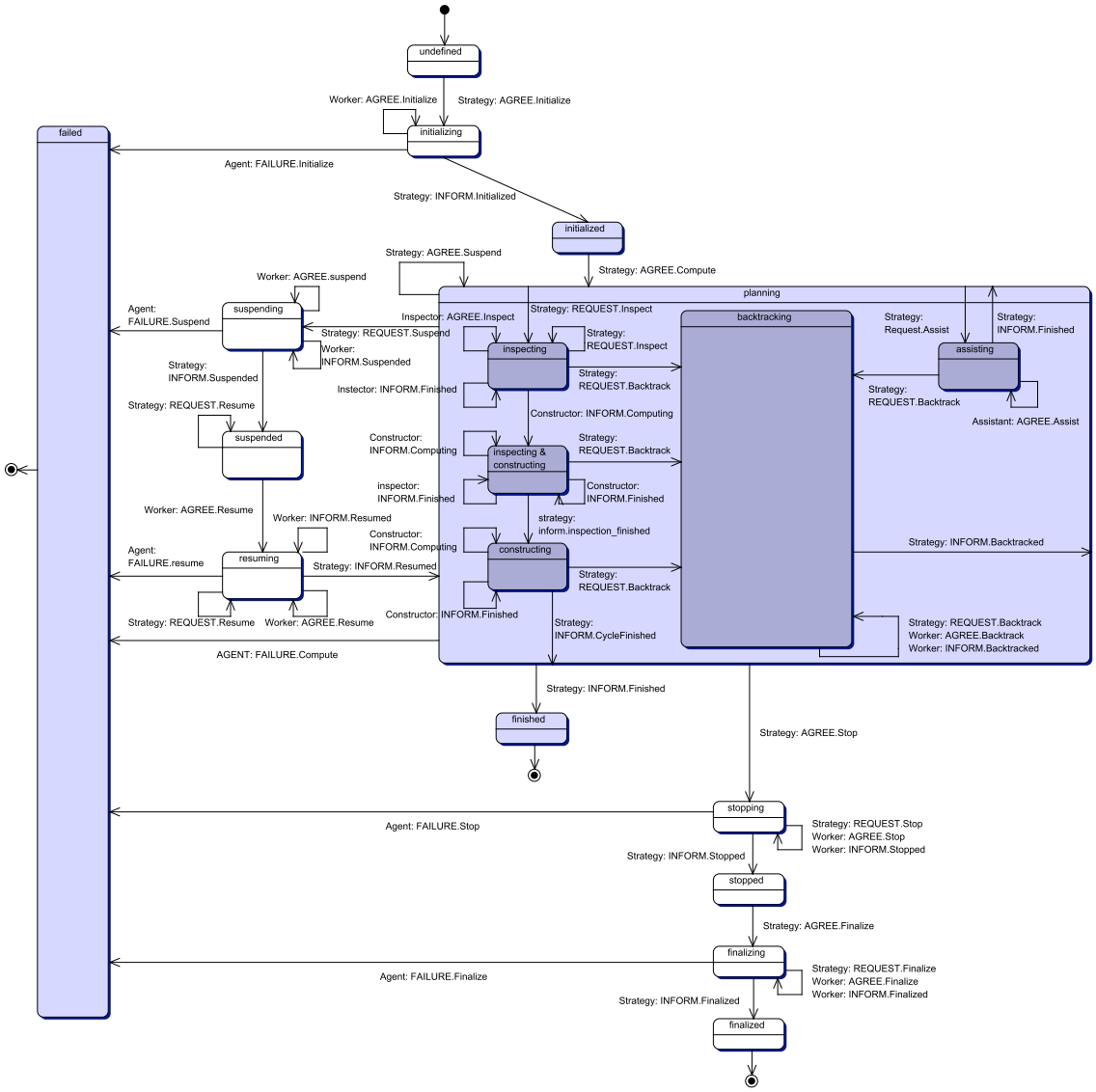


Figure 5.4: The refined PANDA planning process model.

Besides parameters that are necessary for communication like performative name, participant information, etc., an ACL message includes parameters for describing the content that is intended for the receiving participant like the *content language* the content is encoded in, the domain the content refers to, etc. In order to be qualified for the use as a content language in an ACL message, a language must meet certain requirements that are induced by the semantics of the performatives. For example a *request* requires always at least an action to be delivered with the content. Otherwise the agent that receives the request does not know what it is requested to do. Furthermore, when an agent *informs* another agent about the result of an action, the content must contain the propositions that represent the result. Thus, a content language must at least provide representations of actions and propositions, so the agents are able to interact in a meaningful way. The content language that is used by the PANDA agents is described below.

The Planning Process

Figure 5.4 gives an overview of the refined model of the planning process that was taken as the basis for implementation. The white colored states specify the life-cycle management of a planning session (initializing

the process, starting planning, suspending it, etc.). Each state transition is labeled with the triggering ACL message and its originator: `sender:performative` followed by `action` or `proposition`. Senders can also be described by their class, for example, `Worker` denotes all worker agents. The same applies to propositions and actions, for instance, `Compute` denotes the action `Compute` and all sub-actions like `Inspect`, `Construct`, etc.

The planning process starts in an artificial `undefined` state in which all agents are deployed and send agreements for their initialization process. After that, the Strategy agent informs all other agents, that the system is initialized, and this is where the reference model started with phase 1: The assistants are requested to perform their inference on which they have to agree. After their processing (leaving the `assisting` state), the inspectors are requested to search for flaws (phase 2), and so on.

Please note that not all states have been modeled in the state machine. Most states are abstract in order to reduce complexity of the state automaton while maintaining a degree of granularity that allows the user to monitor the planning process. For example, the state `backtracking` summarizes all possible sub-states that describe the interaction between each particular worker agent and the Strategy agent.

The refined planning process model has two major improvements over the reference model: First, it extends agent concurrency. The reference planning process model in Fig. 5.1, defines the agent classes to execute one after another, synchronized by phase transitions. In that model, concurrency is only allowed within a particular phase. But especially between phase 2 and 3 such a synchronization is too strict, because a constructor must wait until the last inspector has finished, even if a constructor has already received all flaws it requires for computation. Constructors should therefore be able to decide on their own when to start execution. The refined process model reflects this by a combined `inspecting&constructing` state. The constructors' behavior has therefore been changed from reactive to pro-active – resulting in a stronger notion of agency.

Second, an enhanced backtracking procedure allows for the implementation of optimized and more sophisticated reasoning techniques, including worker agents, that means assistants, inspectors, and constructors, to be equipped with a state history or caches, etc. Such enhanced agents can take advantage of a plans history such that they can re-play their results if no relevant change happened to the plan and that they are also able to focus on the difference between a plan and its refinements. In order to keep backtracking consistent, that means, in order to "migrate" the history-aware agents properly onto the new plan generation path, the worker agents now actively participate in the backtracking process: they have to synchronize via agreement statements and then notify the Strategy agent when they are finished (cf. state transitions from `backtracking`).

Thus, the agent behavior is extended by a backtracking mechanism with three core capabilities: First, a synchronized restart of the system must be guaranteed, that means, a restart can only take place, if all worker agents have finished backtracking. Second, the different granularity of the state histories of agents working in different sub-cycles is considered. Assistants can be executed multiple times in a planning cycle, whereas Inspectors and Constructors will be executed only once. Therefore, assistants must be backtracked independently from Inspectors and Constructors. Third, in order to backtrack the system immediately, the Strategy agent must be able to interrupt the worker agents' execution, the agents therefore perform their computations in a non-blocking way.

In summary, the enhanced mechanisms for concurrency and backtracking allow the system to benefit from early fail decisions in terms of an increased performance: Non-repairable inconsistencies are typically very quickly detected and processed by constructors.

Ontology-based Components

Like it has been mentioned before, DAML is used as representation formalism to describe and share knowledge in the PANDA system, ranging from flaw communication to system state transitioning. It is one of the emerging standards in the Semantic Web community for representing and communicating knowledge [139]. It ensures interoperability with third-party systems like RACER and forms the basis for communicating knowledge among agents. Most important, it enables knowledge to be represented in a uniform, explicit, and declarative manner, so the system becomes more robust, flexible, and maintainable.

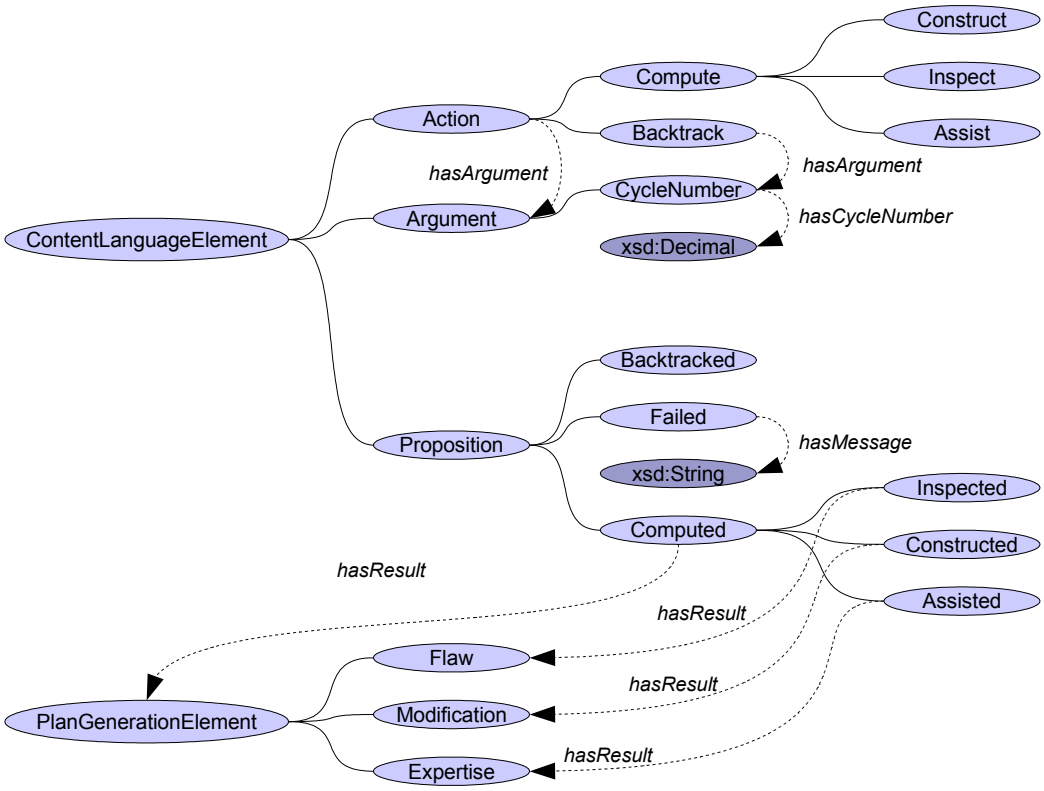


Figure 5.5: The content language ontology.

To be able to use DAML as a content language in ACL messages (recall the speech act structure), at least actions and propositions must be able to be represented within DAML [228]. This is sufficient for the needs of PANDA. Figure 5.5 shows the ontology that provides the concepts to enable DAML-based communication. Property cardinalities have been omitted for clarity.

Actions can have arguments, for example, the action sub-concept `Backtrack` must come with a `CycleNumber` the value of which is represented as the XML-schema type `decimal`. In this way, an action can be compared to a method signature without argument order. In PANDA, every argument of an action is modeled in the ontology in order to give it a formal semantics. Therefore, in contrast to [228], the argument order does not have to be considered. Propositions are currently only used to represent `ActionResults`. The subconcept `Computed` carries the `PlanGenerationElements` that are the results of the worker agents' computations, for example, a `Constructed` proposition has an `Modification` element as a result. The content of an ACL message is represented by instances of the PANDA system ontology embedded in a DAML-document. JENA takes care of encoding and decoding the DAML content of ACL messages. For any content that has to be sent, its JENA object model is constructed using the described ontology. After that, the model is serialized and inserted into the appropriate ACL message. The decoding of DAML content works exactly the opposite way. The object model of the DAML content is constructed by parsing its serialized representation and can then be queried with the JENA API.

DAML plays its second key role in the automated configuration of the agent container (Fig. 5.6 shows the underlying ontology). The configuration process is composed of two sub-processes. First, the agents that are part of the planning process must be instantiated. The object class `PandaSessionBean` uses RACER to derive the leaf concepts of `PandaAgent` and to determine the implementation assignments `ImplementationElements` of the worker agents. In the example of Fig. 5.6, the assigned implementation for the `Inspector1` agent is an instance of JAVA class `panda.jade.agent.Inspector1Impl`. After being created, the PANDA agents insert their descriptions into the ABox of RACER so RACER keeps track of the deployed agent instances.

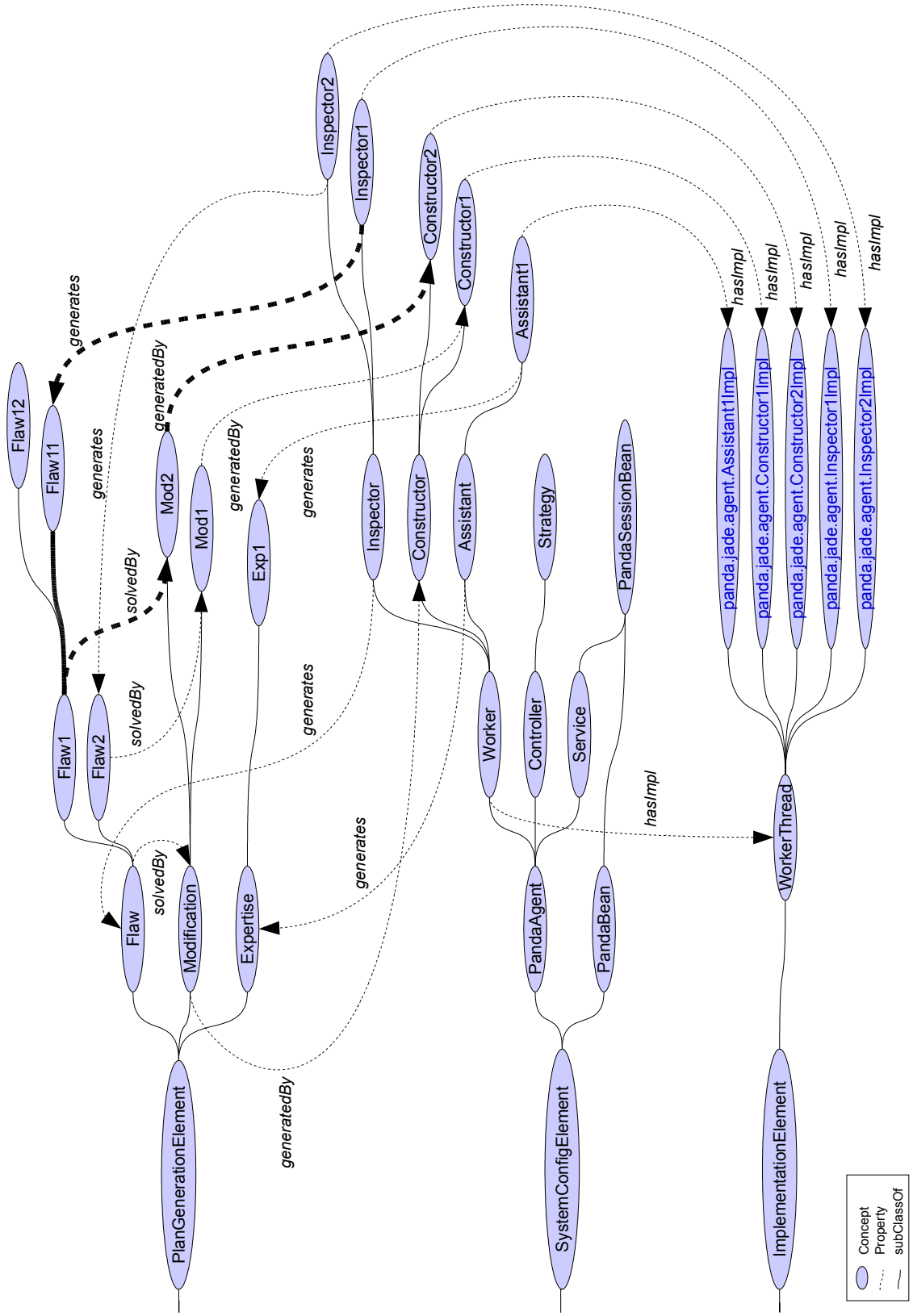


Figure 5.6: The system configuration ontology.

Second, the communication links, that is to say, the implementation of the triggering function α, must be established between the instantiated agents. RACER is used by each PANDA agent on startup to derive its communication links to other agents, that means, from which agents it will receive and to which agents it has to send messages. This is done by using the ontology to derive the dependencies between agents from defined dependencies between the flaws and modifications: The system ontology specifies which agent instance implements which type of `Inspector`, and it does the same for the constructor agents. RACER derives from this information which flaw and modification types are going to be generated by the agent instances, and if the model includes an α-relationship between them (the `solved-by` property), the agent instances' communication channels are linked. Based upon the subsumption capabilities that come with DAML and description logics, it is even possible to exploit subclass relationships between `PlanGenerationElements` (illustrated by the bold printed concept connections in Fig. 5.6). An example for a modification class hierarchy are ordering relation manipulations with subclasses *promotion* and *demotion*. Regarding flaws, the system ontology distinguishes *primitive* open preconditions and those involving state abstraction axioms (Def. 2.1).

A knowledge based configuration offers even more benefits: Imagine a less informed configuration mechanism, say, reading a respective file in XML format that holds the descriptions on the agents to be loaded and the message links to be established between them as a representation of the triggering function α. Semantic verification can then only be based on type checking by, for instance, JAVA class loaders. In the presented architecture, the system model can be checked on startup for possible inconsistencies in a verification step of the planning process in state `initializing` before plan generation starts (cf. Fig. 5.4). An example for such an inconsistency is a constructor that is missing a link to a flaw (cf. modification-complete system configurations, Def. 3.3), warnings can be issued for flaws and modifications without implementations of their generating agents, etc.

5.2 Application Domain Models

In order to illustrate how concrete application domains for the presented planning framework and its configurations are built, we have chosen three showcase domains models as representatives for hybrid planning problems (see $\mathfrak{C}_{\text{HYBP}}$ in Sec. 3.3.1). Our motivation for this focus is twofold: Firstly, hybrid planning is the first of our configurations that extends the classical planning methods and therefore nicely demonstrates the immediate gain of flexibility in terms of modelling and solution generation. The latter aspect will be elaborately addressed in the experimental strategy evalution in Chap. 6. Secondly, the chosen domain models represent three facets of dealing with that modelling flexibility and thus give an idea what a modelling methodology for the higher-level extensions may be appropriate. We furthermore believe that our results can considerably easier be judged the more the domains for demonstration and evaluation correspond to known ones.

The main focus in the area of planning system development has recently started shifting from *what* kind of knowledge can be processed and used for plan generation to *how fast* can the planning system come up with a solution. This trend started with the *International Planning Competition* (IPC),[7] a biennial event in which the participants are presented a set of domain models and problems to run their planning software on. The system's performance is then measured in terms of running time (respectively the number of problems solved within a given time limit) and solution quality (in most cases the number of plan steps in the solution). The competition had two major impacts on the planning research community: The rising of a de-facto standard PDDL modelling-language and the emergence of a set of benchmarks. Both are hardly to be imagined not to be referenced in any modern paper in the field. It has however also to be noted that neither the language is undisputed (its adequacy for real-world problems, semantic issues, etc. see Sec. 2.8.1) nor are the benchmark problems (artificial problems, some considered to be toy examples). But nonetheless we decided to pick two former benchmark problems and translate them into our extended representation formalism.

[7] `http://ipc.icaps-conference.org/`

As stated above, the choice of our example domains is motivated in covering specific modelling aspects: The first domain is consequently a "hierarchization" of the classic *Satellite* benchmark model and demonstrates how a PDDL-like representation can be transcribed into our formalism and then reasonably enriched by action abstraction elements. Our second showcase is called *UM Translog* and shows the inverse process of model acquisition by starting from a pure HTN representation that is augmented by advanced causal structures along the decomposition hierarchy. The third and final example is an artificial construct, the *Criss-Cross* domain, that has been specifically designed for providing problems that exhibit a dense network of causal dependencies. As the evaluation will point out, these domain models allow for benchmark problems of various characteristics.

Since a complete discussion of all the details for each domain model is clearly not feasible with the given space constraints for this thesis, we focus in each section on what we believe are the most relevant facets of the models.

As a final technical note we would like to point out that we preferred the domains of IPC3 [171] and before over more recent developments (IPC4 [137]) for two reasons: Firstly, the older domain models were less fine-tuned to newer PDDL language features of versions 2.2 [80] and 3 [111] with their semantic particularities. This includes time windows, derived predicates, continuous effects, transformational compilation schemes between language levels, and the like, which to translate into our semantic frame is out of the scope of this thesis. These concepts do, secondly, for their largest parts not substantially differ from our supported HTN features. For instance, we ask the reader to compare the arguments along the discussion in [84] on HTN expressivity with the introduction of the artificial concepts like "struts", "clips", and "maintenance goals" into PDDL 2.1 [99, 100].

5.2.1 Satellite

Introduction

The *Satellite* domain has been introduced in its original form in the 2002 planning competition. It is inspired by space-applications that are a first step towards the "Ambitious Spacecraft" as described by David Smith at the AIPS 2000 conference [239]. It involves planning and scheduling a collection of observation tasks between multiple autonomous satellites, each equipped in slightly different but possibly overlapping ways. The equipment consists of observation instruments with different characteristics in terms of data productions, so-called *modes* like thermal images, x-ray, etc., and appropriate calibration targets. Satellites are mobile and can be pointed at different targets by slewing them between different attitudes/directions. A benchmark problem in this domain is consequently given by an initial state that describes the satellite configurations and phenomenon positions, while the goal state specifies of which observation targets an image has to be taken in which mode.

Sorts, Relations, and Functions

We adapt the respective STRIPS/ADL variant of the IPC benchmark suite (there also exist models including temporal and resource reasoning). However, for a proper formal description of the domain we cannot refer to the IPC domain repository because none of the current PDDL versions supports hierarchical modelling concepts. We will therefore present the *Satellite* domain, including our modifications to it, in the formalism that we have introduced in Chapter 2, that means we are defining a decomposition domain model $D = \langle \mathscr{M}, \Delta, \mathrm{T} \rangle$ (see Sec. 3.3.1). The domain model in turn is based on a logical model $\mathscr{M} = \langle \mathbb{D}, \mathscr{I} \rangle$ and a language $\mathscr{L}_{\mathrm{Sat}}$. The former is assumed to be self-explaining: The carrier sets refer to the real-world objects that the rigid constants are named after (satellites, phenomena, etc.) and the interpretation of the rigid symbols (used in satellite configurations, instrument capabilities, etc.) is given in the problem specification as usual. Finally, a language has to be defined by an appropriate tuple $\langle \mathscr{Z}_{\mathrm{Sat}}, \leq_{\mathrm{Sat}}, \mathscr{R}_{r_{\mathrm{Sat}}}, \mathscr{R}_{f_{\mathrm{Sat}}}, \mathscr{F}_{r_{\mathrm{Sat}}}, \mathscr{F}_{f_{\mathrm{Sat}}}, \mathscr{V}_{\mathrm{Sat}}, \mathscr{T}_{p_{\mathrm{Sat}}}, \mathscr{T}_{c_{\mathrm{Sat}}}, \mathscr{E}_{\mathrm{Sat}} \rangle$. We will incrementally develop some representative pieces from all of the above in the following sections. Although this showcase will be comparatively small with its 6 sorts, 8 relations, 8 task schemata, and 8 method declarations, we do not present all details here and refer the interested reader to the previously mentioned material web-page.

The first thing to do when translating the *Satellite* domain into the PANDA formalism is to introduce a sort hierarchy for describing the occuring concepts. While the PDDL encoding defines the types for satellites, directions, instruments, and modes, we feel that this flat sort hierarchy does not capture an essential feature of the problem instances: the defined directions are obviously divided into the actual observation phenomena that are of scientific interest and attitude points that are only used for calibration purposes. This aspect is incorporated in the following sort hierarchy by providing appropriate sub-sorts of `Direction`.

$$\begin{aligned}\mathscr{Z}_{\text{Sat}} &= \{\texttt{Instrument}, \texttt{Satellite}, \texttt{Mode},\\ &\qquad \texttt{Calib\_Direction}, \texttt{Image\_Direction}, \texttt{Direction}^A\}\\ \leq_{\text{Sat}} &= \{(\texttt{Calib\_Direction}, \texttt{Direction}^A), (\texttt{Image\_Direction}, \texttt{Direction}^A)\}\end{aligned}$$

The sort-annotation A indicates that we consider directions to be abstract entities and that no constant declaration of that sort is allowed. The rationale for this language feature is to identify sorts that are intended to be exclusively used for structuring the application domain concepts. This supports our modelling tools to argue about problem and domain consistency.

When it comes to specifying the relation symbols for expressing facts about the world state, a PDDL model explicitly provides the relation symbols' signatures in its declaration header. It does however not specify whether actions are allowed to manipulate the respective interpretations, that means, whether it is flexible, or if it is a rigid attribute. Studying the model's documentation suggests the following partitioning of relation symbol declarations:

$$\begin{aligned}\mathscr{R}_{f_{\text{Sat}}} &= \{\texttt{pointing}_{\texttt{SatelliteDirection}^A}, \texttt{power\_avail}_{\texttt{Satellite}}, \texttt{power\_on}_{\texttt{Instrument}},\\ &\qquad \texttt{calibrated}_{\texttt{Instrument}}, \texttt{have\_image}_{\texttt{Image\_DirectionMode}}\}\\ \mathscr{R}_{r_{\text{Sat}}} &= \{\texttt{on\_board}_{\texttt{InstrumentSatellite}}, \texttt{supports}_{\texttt{InstrumentMode}},\\ &\qquad \texttt{calibration\_target}_{\texttt{InstrumentCalib\_Direction}}\}\end{aligned}$$

The `pointing` relation is used for expressing that a satellite platform, and with it all on-board instruments, aim at a given direction. Slewing the satellite therefore controls the orientation of the desired instrument. The `power_avail` and `power_on` symbols reflect that energy is a limited resource on the observation platform and that instruments consequently have to be switched off before other instruments can be activated. On-board observation systems typically have to take reference images for calibrating the sensors and if an instrument is ready for taking images, its status changes into `calibrated`. In a state in which the image of a phenomenon is finally taken a respective atom over `have_image` is supposed to hold.

The elementary operations for this language, $\mathscr{E}_{\text{Sat}}$, can be directly obtained from these flexible relation symbols.

The rigid symbol set includes the relation for modelling which instruments a satellite carries, which kind of sensor the instrument provides, and what the reference object for the instrument is.

In this version of the *Satellite* domain, no function terms occur in the actions' preconditions and effects and therefore $\mathscr{F}_{r_{\text{Sat}}}$ and $\mathscr{F}_{f_{\text{Sat}}}$ are empty sets. We omit the variable symbols here, since they are generated on the fly during plan generation.

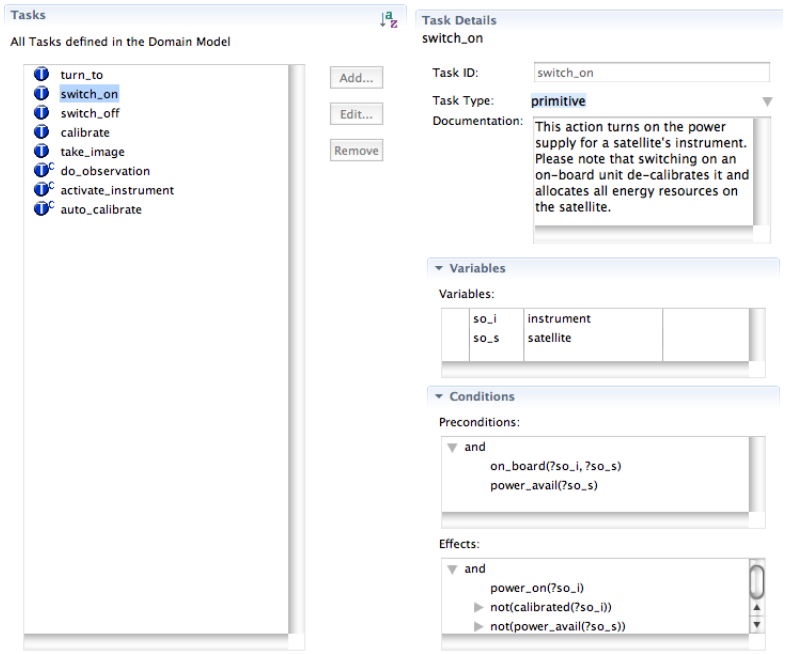


Figure 5.7: The definition of the `switch_on` operator schema (screenshot taken from our prototypical domain model editor).

Tasks and Methods

The final language elements are the operator and task symbols. The former can be directly taken from the PDDL description:

$$\begin{aligned}\mathcal{T}_{p_{\mathrm{Sat}}} = \{&\texttt{turn\_to}_{\texttt{Satellite}\,\texttt{Direction}^A\,\texttt{Direction}^A}, \texttt{switch\_on}_{\texttt{Instrument}\,\texttt{Satellite}},\\ &\texttt{switch\_off}_{\texttt{Instrument}\,\texttt{Satellite}}, \texttt{calibrate}_{\texttt{Satellite}\,\texttt{Instrument}\,\texttt{Calib\_Direction}},\\ &\texttt{take\_image}_{\texttt{Satellite}\,\texttt{Image\_Direction}\,\texttt{Instrument}\,\texttt{Mode}}\}\end{aligned}$$

The intended meaning of these five operators is apparent and needs no in-deep explanation. The associated schema definitions T of the domain model are also straight-forward adaptations of their PDDL counterparts, for example as the operator that is depicted in Fig. 5.7. It is the primitive task schema `switch_on` and represents the operator for feeding energy to an instrument on the observation platform. The action is executable in a state in which the satellite has energy available, therefore no two instruments can be used in parallel. The other precondition for the action is that the desired instrument is on board the given satellite (which is a rigid state feature). Readers that are familiar with IPC's model suite will notice that we do not use conditional effects where the competition's "ADL" variants do. Note that a conditional effect can be easily translated into a complex task with an appropriate implication postcondition and two methods with an operator each that introduce the effects for the two cases. Our framework is therefore able to cover this effect representation, but the intended semantics for the PDDL description – if a state feature does not hold yet, it does so now – are perfectly covered by our action semantics (see relation update-functions in Sec. 2.3).

When we analyze the IPC benchmark problems in this domain and their documentation, it occurs to us that there obviously exists an *intended procedure* for taking satellite images and that all solutions follow that pattern with minor deviations: Making an observation for a given sensor mode and phenomenon firstly consists of choosing a suitable instrument, which indirectly determines the satellite that performs the observation. In a second step, the instrument has to be routed energy to and properly calibrated. The satellite finally slews in the direction of the target phenomenon and takes the image.

This procedure is plausible enough to be considered not as a specification artefact that is introduced by the competition initiators but as an underlying principle in the *Satellite* domain and consequently a clue

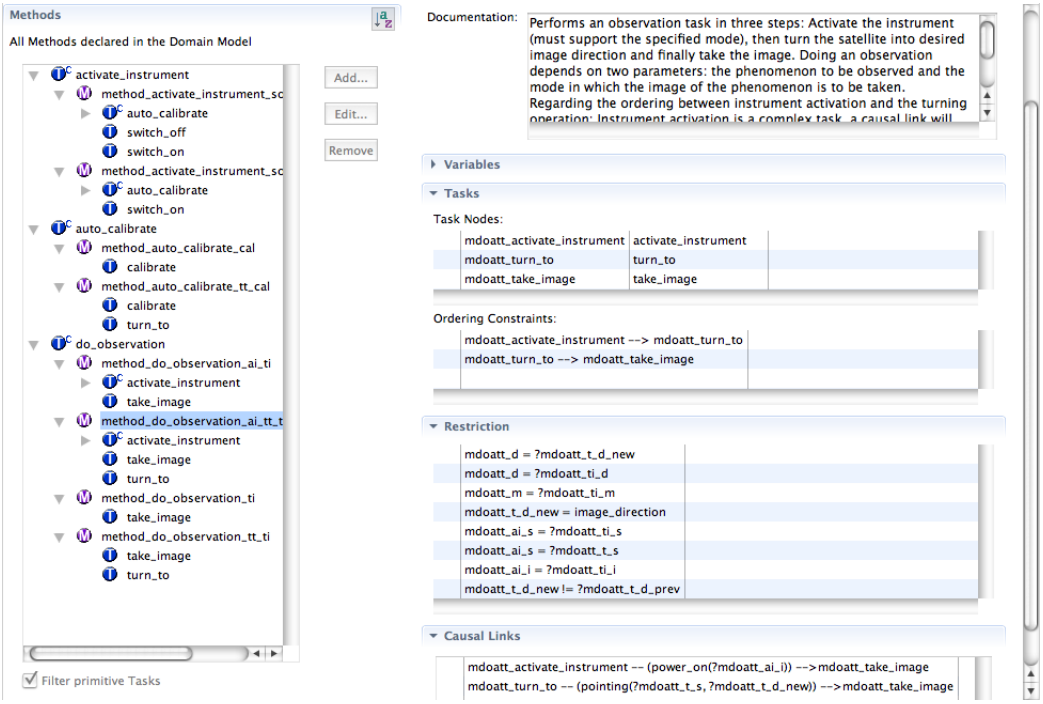


Figure 5.8: The decomposition hierarchy of the *Satellite* domain (left) and an example for a method declaration (right).

for a well-reasoned action abstraction. An apparent structure is to build an abstraction for each of the two phases: preparing the instrument and taking the picture becomes an abstract task `do_observation` with parameters for the desired phenomenon to observe and the mode to support. The preparation phase seems to require an abstraction hierarchy on its own, in order to encapsulate the different ways of getting the sensory system on-line (the instrument is already on and calibrated, some other instrument has to be turned off first in order to raise the energy level properly, etc.). We therefore introduce an abstract action for activating the instrument and for dealing with the calibration. The complex task symbols of the *Satellite* domain are consequently the following three:

$$\mathscr{T}_{c_{\mathrm{Sat}}} = \{\texttt{do\_observation}_{\texttt{Image\_DirectionMode}}, \texttt{activate\_instrument}_{\texttt{SatelliteInstrument}}, \texttt{auto\_calibrate}_{\texttt{SatelliteInstrument}}\}$$

We intend to define the complex task schemata $\mathtt{T}$ in the domain model in the fashion of ABSTRIPS operator reductions. That means, we do not employ state abstraction axioms but simply generalize the preconditions and effects. We may assume that all variables in the following definition are provided by $\mathscr{V}_{\mathrm{Sat}}$ and of the appropriate sort.

$$\begin{aligned}
\texttt{do_observation}(id_{do}, m_{do}) &= \langle \top, \texttt{have_image}(id_{do}, m_{do}) \rangle \\
\texttt{activate_instrument}(s_{ai}, i_{ai}) &= \langle \texttt{on_board}(i_{ai}, s_{ai}), \texttt{power_on}(i_{ai}) \rangle \\
\texttt{auto_calibrate}(s_{ac}, i_{ac}) &= \langle \texttt{on_board}(i_{ac}, s_{ac}) \wedge \texttt{power_on}(i_{ac}), \texttt{calibrated}(i_{ac}) \rangle
\end{aligned}$$

Given these complex and primitive tasks, the methods of the domain model set up a decomposition hierarchy that implements the different observation procedures. Figure 5.8 displays this hierarchy on the left. The first level in the tree-like representation are the complex task schemata (a blue disc labelled with "T" and a superscript "C"). Each sub-tree stands for the declared decomposition methods in the domain, symbolized by violet "M" icons. For example, the selected method in the figure is named `method_do_observation_ai_tt_ti`, which is the mnemonic for "implementing the observation task by an activation, a turning operation, and taking an image". This method is partly displayed on the right hand side of the figure. Its formal specification is the following (variable symbols and plan step prefixes are simplified for better readability):

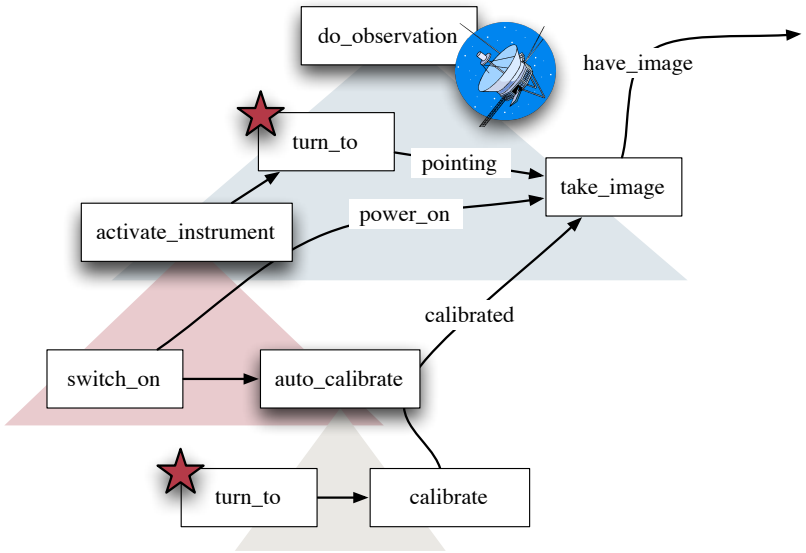


Figure 5.9: A three-level decomposition of the complex task `do_observation`.

$$
\begin{aligned}
m_{\texttt{do\_observation\_ai\_tt\_ti}} = \langle & \texttt{do\_observation}(i,m),\\
\langle\ & \{\texttt{a}:\texttt{activate\_instrument}(a:s,a:i),\ \texttt{b}:\texttt{turn\_to}(b:s,b:to,b:from),\\
& \ \texttt{c}:\texttt{take\_image}(c:s,c:dir,c:i,c:m)\},\\
& \{\texttt{a}:\texttt{activate\_instrument} \prec \texttt{b}:\texttt{turn\_to},\ \texttt{b}:\texttt{turn\_to} \prec \texttt{c}:\texttt{take\_image}\},\\
& \{i \doteq b:to, i \doteq c:dir, m \doteq c:m,\\
& \ b:to \dot{\in} \texttt{Image\_Direction},\\
& \ a:s \doteq b:s, a:s \doteq c:s, a:i \doteq c:i, b:from \dot{\neq} b:to\},\\
& \{\texttt{a}:\texttt{activate\_instrument} \xrightarrow{\texttt{power\_on}(a:i)} \texttt{c}:\texttt{take\_image},\\
& \ \texttt{b}:\texttt{turn\_to} \xrightarrow{\texttt{pointing}(b:s,b:to)} \texttt{c}:\texttt{take\_image}\}\rangle\rangle
\end{aligned}
$$

The first line of variable constraints thereby relates the abstract task's parameters with the expansion network, the second and third establish a consistent parameter binding. Figure 5.9 depicts one way of decomposing an abstract observation task. The light blue triangle thereby represents the just presented method. The curved arrows indicate an "inherited" causality, that means, the `have_image` state feature is an effect of the complex `do_observation` but it is eventually provided by the operator `take_image`. Instrument activation is decomposable into switching on and performing an abstract calibration procedure (red expansion network). The last complex task in the decomposition hierarchy is finally decomposed into an operator for slewing the satellite into the appropriate direction and then calibrating the instrument on that phenomenon. Note that it takes only three decompositions to produce a consistent plan that implements an observation, a plan with 6 primitive actions that are completely causally linked and the parameters of which are consistently (non-) co-designated. The red stars in the figure mark the preconditions of the `turn_to` operators, which introduce the most important source of combinatorial problems in this domain, namely the question how to establish a `pointing` state feature. If the depicted decomposition is pursued, an observation is self-contained such that the turning operation after the calibration step is properly causally linked and fully supported from within the surrounding network steps. If a plan, however, contains a number of observation operations that are only developed to the "blue" level, there is typically some confusion about causal support with respect to the orientation of the satellite. The following table displays the possibilities of implementing an observation in the presence of other observations.

A problem in the *Satellite* domain is typically given by abstract observation tasks. The hybrid domain

model offers four implementation variants, the applicability of which depends on the observation context:

1. First, the instrument is activated, then the satellite turns into the direction of its scientific interest and takes the image. This is the base case for isolated (sub-) problems as explained above and showed in Fig. 5.9.
2. The instrument might have been used in a previous observation and is still calibrated. In this case it is sufficient to slew the satellite and take the image.
3. Problems with many jobs may take advantage of decompositions that provide an activation-imaging skeleton for which the slews can be filled in later by task insertion modifications. This method therefore provides the causal information that connects activation and usage of the sensor.
4. If the configuration supports task insertion (like in the previous case) or if we have to deal with exceptional situations in which more than one image is required of a phenomenon, the fourth variant solely consists of a direct translation into taking the image.

Activating an instrument is implemented by two alternative methods: The first is to switch the required instrument on and to initiate the calibration procedure, while the second method covers those cases in which energy has to be re-routed from a currently on-line instrument before.

The last two methods implement the calibration process, which is either atomic in the context of other observation tasks or has to perform a preparatory slew into the calibration direction.

Concluding Remarks

It is certainly an inconvenience of the *Satellite* domain that it induces refinement spaces that obviously contain a large number of isomorphic plans. While it is a general property of hybrid domain models that their decomposition methods are necessarily reproducible via task insertion, this model is furthermore focused on exactly one complex task. The different methods do thereby not provide *alternative* ways of performing the task but define the configuration of the observation process: with or without calibration, with or without a previously used instrument, with or without preparation slews, and the like.

On the other hand, the *Satellite* domain has several nice properties due to which it qualifies as an interesting demonstration and benchmark domain for our hybrid planning configurations.

The first advantage is the intuitive simplicity of the application domain and the underlying principles. This also holds for model extensions like incorporating temporal information, addressing energy consumption, etc. Any modification can easily be explained and its effects on the solution generation process investigated. It is also a relatively simple task to formulate problems, to validate solutions, and to judge problem complexity as well as solution quality. In contrast to other simple benchmark scenarios, planning for satellite observations exhibits significantly more variety and extension options (cf. stacking of coloured boxes).

A second characteristic is the applicability of hierarchical modelling concepts. The *Satellite* domain does not only naturally motivate task abstraction but it also offers several reasonable ways to implement that. The presented decomposition hierarchy is of course not the only plausible model and it would be an interesting research question to quantify the impact of changing that hierarchy. While there are obviously alternative method configurations with respect to the arrangement of tasks, there is also the possibility of "deepening" the hierarchy and that of "flattening" it. The latter implies to reduce the number of pre-defined task implementation alternatives and to leave closing the preconditions solely to the respective modification generators.

A last observation on the *Satellite* benchmark problems anticipates our experimental evaluation (Chap. 6). This domain allows to control problem complexity, for example in order to determine the scaling behaviour of a strategy, in several dimensions. For example, the mildly sophisticated "Copy'n'Paste" complicating, that means, duplicating of observation jobs and scientific equipment, leads to an increasing number of self-similar sub-problems. Although this may be the intended scientific focus, it has to be taken into account that this kind of complexity may not favour a generally well performing search strategy. It makes solving ten

times more observations as "more difficult" than stacking ten times more boxes. In contrast, we can define benchmark problems with an increasing number of observation jobs that require an increasing number of modes and instruments on a stable number of satellites. This induces an increasing number of interacting sub-goals and causal interferences. As a consequence, the refinement space grows but its solution density declines. A third aspect is the amount of overlapping target requirements, respectively instrument capabilities: if a set of observations can be performed by single platforms sequentially as well as by multiple platforms in parallel, optimality of the solutions becomes more and more an issue.

5.2.2 UM Translog

Introduction

The *UM Translog* domain describes an urban transportation and logistics scenario that has been first introduced as a challenging benchmark problem for the UMCP planning system at the University of Maryland - hence the name. It covers scenarios in which different types of transportation goods are delivered to customer locations by various means (trucks, trains, etc.) via appropriate infrastructures (roads, railroad lines, transport centers, etc.). Among the obvious tasks of planning for loading the good, finding a path through the infrastructure, and finally managing the delivery, additional requirements come into play: the particular goods demand particular transportation means and procedures. Hazardous goods demand specific (un-) loading protocols, valuable item transports have to be secured and insured, livestock needs to be fed, etc. A problem specification in this domain consists typically of a number of delivery jobs of a specific good from one customer location to another. The more jobs are given, the more becomes solution quality an issue.

In its original version it is specified as an HTN domain model and the authors of [8] argumented convincingly in favour of such somewhat realistic, more knowledge-rich scenarios for comparing planning systems' performance. Furthermore, Erol used it in his thesis [83] as a showcase for the expressivity of his approach and for the adequacy of an HTN representation. Nonetheless, the IPC community decided to adhere to the commonly agreed, non-hierarchical PDDL standards and therefore a PDDL-conforming version has been issued for IPC 2002 [290]. While the hierarchical version uses HTN methods to capture the specific transportation procedures when decomposing an abstract delivery task, the "flat" PDDL version synthesizes causal chains for delivery goals.

Our *UM Translog* interpretation as a hybrid domain model consequently starts from the HTN specification of the UMCP distribution and complement it step-wise by task preconditions and effects. If we call the above modification of the *Satellite* domain a hierarchization then this section describes a "causal grounding" of an HTN model.

Sorts, Relations, and Functions

The HTN formalization of *UM Translog* as well as its PDDL translations support a simple type system in order to express sub-sort relationships. Both variants, however, try to employ the notion of a type hierarchy by additional predicates that simulate type information, similar to the representation of a sorted logic by predicate logic with designated sort predicates. There is, for example, a defined sort `Route` and a predicate $\texttt{Rail} - \texttt{Route}_{\texttt{Route}}$ for representing the type of railroad lines (the former is also a so-called *primary type*). In our formalism, this implicit type information becomes an explicit sort declaration; we will discuss the consequences of such a translation in Sec. 5.2.2. We defined sub-sort relationships according to the usage of the non-primary type predicates in the expansion network. Along the decomposition hierarchy, a transportation vehicle is, for example, first specified to be a train car and in the specialized train-car handling methods it is assigned to be the argument to a liquid transport type predicate. From this information we deduce that there has to be a sort for categorizing these objects, name it `Tanker_Traincar`, and place it in the sort hierarchy accordingly.

The *UM Translog* sort hierarchy is however too large to be displayed here, Fig. 5.10 therefore depicts a simplified model fragment that contains about two thirds of the 96 sorts in the complete version (with a

maximum depth of 7). Regarding the geometric shapes, sort names in ellipses denote abstract sorts while sort names in boxes represent sorts for which constants can be provided.

The coloured areas symbolize four key areas of the model: The blue rectangle at the bottom backdrops all sorts concerning the modelling of transportation routes and locations. Note the use of multiple super-sorts: the `Hub` concept represents the property of transportation centers to serve as a central distribution facility with storage rooms, etc. Blue triangles mark the two prominent top-level sorts for all kinds of transportation vehicles and packages (transportation goods). Finally, red triangles are the entry points in the hierarchy for all sorts that represent packages and vehicles with a given physical or "special" property. For instance, the physical property `Regular` denotes objects that are approximately cuboid, stackable, and do not need a special treatment. But it is also used for those transportation devices and vehicles that are able to deal with these objects. This peculiarity has been adopted from the original domain description.

Let us run through a small example for declaring two object classes in the *UM Translog* domain: the automobile transport. For providing the transportation means, we define the sorts `Thing`, `Object`, and `Vehicle`. A physical property of such a vehicle are the specifics of automobile transports, therefore we introduce the sort-path with `Thing`, `Object`, `Physical`, and `Auto`. The common sub-sort of `Vehicle` and `Auto` is given by `Auto_Vehicle`. This class represents all kinds of automobile transports, for instance `Auto_Truck`.

Concerning the relation symbols, the full domain includes a total of 29 of which 8 are rigid. Like we did for the sorts, we will focus on a representative fragment of the complete set of definitions. Let us begin with the state-independent relations. They basically cover two static aspects of the logistics scenarios: the environment topology and the relationships between transportation means, their infrastructure, and the transported items.

$$\begin{aligned}\mathscr{R}_{r_{\mathrm{UMT}}} \supset \{&\mathtt{Connects}_{\mathtt{RouteLocationLocation}},\\ &\mathtt{In\_City}_{\mathtt{City\_LocationCity}}, \mathtt{In\_Region}_{\mathtt{CityRegion}},\\ &\mathtt{Serves}_{\mathtt{TCenterLocation}},\\ &\mathtt{At\_Equipment}_{\mathtt{EquipmentEquipment\_Position}},\\ &\mathtt{PV\_Compatible}_{\mathtt{PackageVehicle}}, \mathtt{RV\_Compatible}_{\mathtt{RouteVehicle}}\}\end{aligned}$$

The first four relations thereby deal with abstract transportation routes between the different location types. An airport will, for example, be connected in this sense to another airport via an appropriate `Air_Route` instance. Relations over the symbols `In_City`, `In_Region`, and `Serves` define the geography of a *UMT Translog* scenario. Fig. 5.11 depicts an example for such a topology: The ellipses stand for three representatives of the largest geographic entities, the regions. A region organizes its associated cities (boxes), which in turn group the customer sites (stars) and local transportation centers. The latter are represented as airport and railway icons at the edge of one of the city symbols. The triangle symbolizes a transportation hub, which is not located in a specific region or city but which is responsible for several regions. The intended structure is that transportation centers "serve" one or more cities within a region, while the hubs deal with the inter-region traffic. While `Connects` specifies the inter-city and inter-region routes, all locations within a city are assumed to be implicitly interconnected. The `At_Equipment` is rigid, because the equipment in the *UM Translog* context is stationary, for instance, cranes or conveyor belts. Since these devices can be mounted only in specific places, an extra location sub-sort has to be defined.

The relations for expressing compatibility finally relate package objects with vehicle objects that can handle the package (`PV_Compatible`), respectively routes with vehicles. In the original model, secondary types were specific constants assigned via an appropriate predicate to the respective object. A compatibility can therefore be expressed as an atomic fact on the level of secondary types. Our language does not support comparable expressions on the sort level and therefore our initial design choice, namely to use multiple inheritance for expressing secondary types, appears to be some drawback: it requires compatibility statements for every respective pair of objects. We will take up this issue again in the discussion at the end of this section.

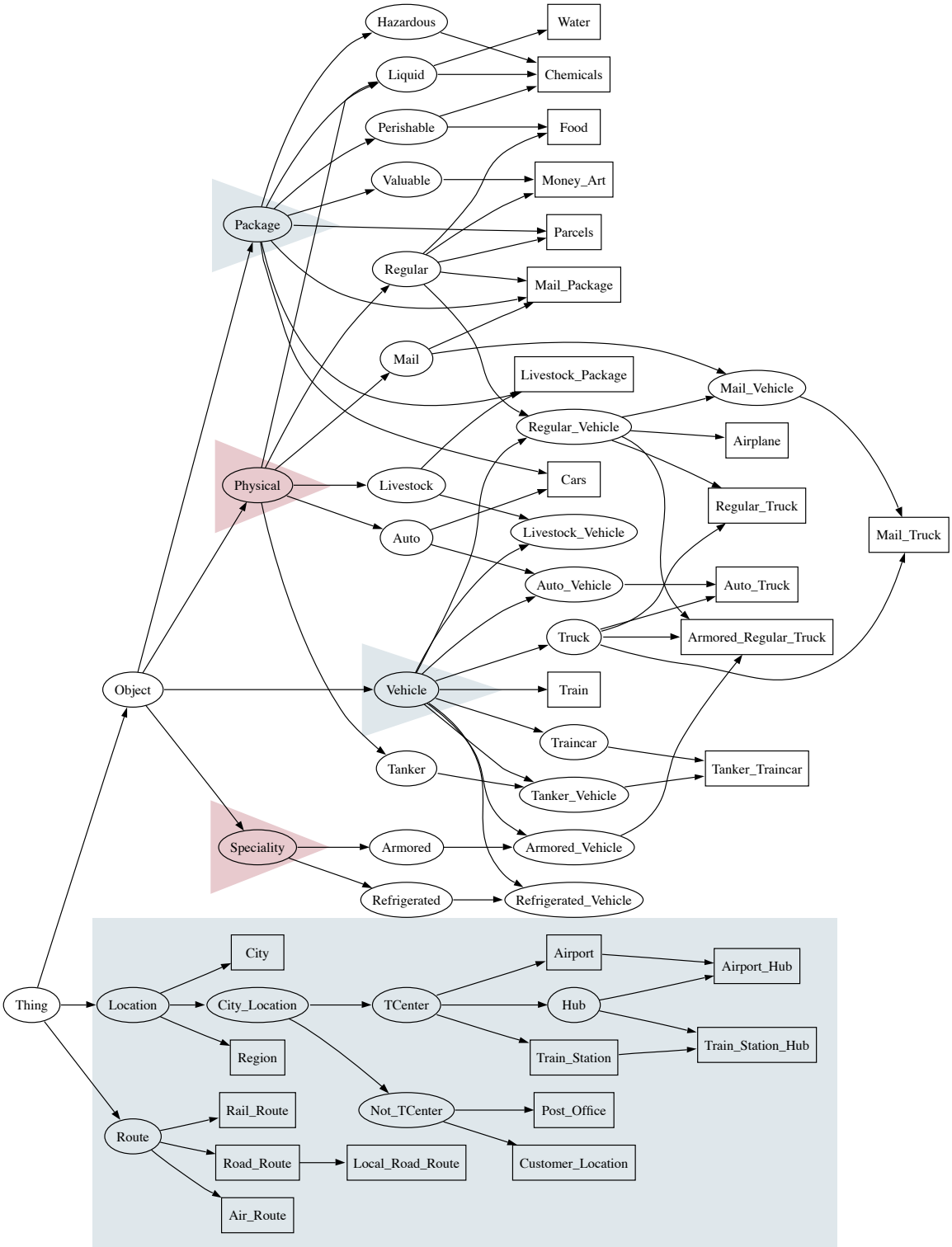


Figure 5.10: A fragment of the *UM Translog* sort hierarchy.

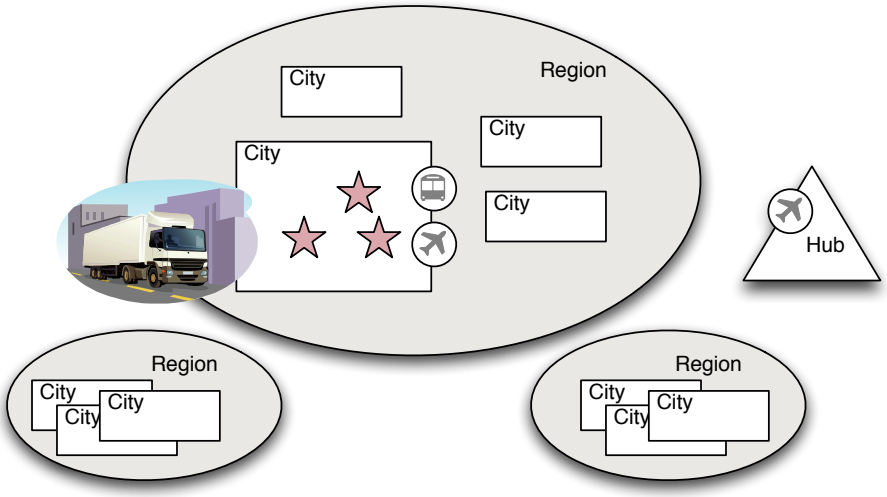


Figure 5.11: The geographic elements of a *UM Translog* scenario: regions, cities, transportation hubs, train stations, and airports. The red stars represent relevant locations within the city limits, for example customer sites.

Concerning the dynamic state attributes, the following flexible relation symbols are an assortment of properties of packages:

$$\mathscr{R}_{f_{\text{UMT}}} \supset \{\texttt{At\_Package}_{\texttt{Package}\,\texttt{Package\_Storage\_Position}}, \texttt{Delivered}_{\texttt{Package}}, \texttt{Fees\_Collected}_{\texttt{Package}}, \texttt{Insured}_{\texttt{Valuable}}, \texttt{Have\_Permit}_{\texttt{Hazardous}}\}$$

The first two relations describe the most important facts about packages, namely where the package is currently located and whether it has reached its destination location and is thereby `Delivered`. During their transport, some more package properties are monitored: An atom over the `Fees_Collected` relation documents whether the customer has paid the fees for the transport, valuable goods have to be additionally insured, and hazardous material requires a specific permission for its transport.

The remaining flexible symbols are used to describe properties of the transportation vehicles. It is worth noting that the original *UM Translog* version did not model any dynamics in the environment's locations like loading staff, etc. A small fraction of the defined properties are the following:

$$\begin{aligned}\mathscr{R}_{f_{\text{UMT}}} \supset \{&\texttt{At\_Vehicle}_{\texttt{Vehicle}\,\texttt{Vehicle\_Position}}, && \texttt{Connected\_To}_{\texttt{Traincar}\,\texttt{Train}},\\ &\texttt{Decontaminated\_Interior}_{\texttt{Vehicle}}, && \texttt{Guard\_Inside}_{\texttt{Armored}},\\ &\texttt{Ramp\_Connected}_{\texttt{Plane\_Ramp}\,\texttt{Airplane}}, && \texttt{Clean\_Interior}_{\texttt{Vehicle}},\\ &\texttt{Trough\_Full}_{\texttt{Livestock}}, && \texttt{Warning\_Signs\_Affixed}_{\texttt{Vehicle}},\\ &\texttt{Hose\_Connected}_{\texttt{Tanker\_Vehicle}\,\texttt{Liquid}}, && \texttt{Valve\_Open}_{\texttt{Tanker\_Vehicle}}\}\end{aligned}$$

Among more general properties like vehicle positions and train configurations, these relations represent safety conditions of special transport. This includes the decontamination for hazardous substance transports, the proper handling of livestock, or the operating of liquid loads. The latter, for instance, requires that the respective valve is not opened until a suitable hose is connected in order to fill the tanker. Finally, there is one rather "technical" relation, a flexible symbol that is used in many contexts. The relation `Available` : `Thing` represents a semaphore for allocating infrastructure and equipment, as well as it denotes temporarily unreachable locations, blocked routes, broken equipment, etc.

Since we re-modelled the original (purely symbolic) HTN domain, this model does not contain any functions. The more recent *UM Translog* versions are however resource enhanced and take into account temporal information as well as storage limits of vehicles and transportation centers. Upgrading our domain model in this direction is part of future work.

Tasks and Methods

The central topic in *UM Translog* is bringing packages from one location to another and while doing so, always complying to the required security protocols. The sort and relation definitions we presented so far already suggest that the action repertoire of this domain model will be very rich and quite realistic. The complete model contains 51 primitive operators and 21 complex tasks, for which in turn 51 decomposition methods are provided. The thereby induced decomposition hierarchy is up to 10 expansion steps deep. We will detail some parts of this hierarchy in order to demonstrate the model design principles.

There is a noteworthy observation to be reported from our work of translating the domain: From the point of view of knowledge engineering, we started out from an HTN decomposition hierarchy that ends in primitive task nodes. From these complex task implementations we re-engineered the appropriate abstract preconditions and effects that are consistent with the pre-defined primitive task networks, a process that resembles the automated ALPINE abstraction computation [153]. At the same time, we experimented with intuitively plausible conditions for the existing complex tasks and tried to propagate them in a top-down manner. Both resulted in an iterative process in which causal information is propagated up and down the decomposition hierarchy. However, it turned out that this method has its limitations, since the original domain model was not set up according to plausible implementations that finally constitute something in the sense of our refinement notion. HTN hierarchies rather tend to focus on structuring the search space and as a consequence the re-engineered complex task conditions of many "implementations" turned out to become empty. Whenever that happened, we tried to partition the methods according to plausible implementations and introduced intermediate helper tasks. If that did not produce a satisfactory hierarchy, as a last resort, we modified the operator description and repeated the re-engineering process.

Let us therefore begin with most important complex task that is the entry point in the decomposition hierarchy with respect to an HTN problem definition: the abstract transportation process. For variables p of sort `Package` and o and d of sort `Location`, a transport is defined as it is common for movement-like schemata as follows:

$$\begin{aligned}&\texttt{transport}(p_{\texttt{Package}}, o_{\texttt{Location}}, d_{\texttt{Location}}) = \\ &\quad \langle \texttt{At\_Package}(p,o), \neg\texttt{At\_Package}(p,o) \wedge \texttt{At\_Package}(p,d) \rangle\end{aligned}$$

The intended semantics of this schema is of course that the package is to be transported from one (customer) location o to another (customer) location d. On this level of abstraction, neither the transportation means nor a path through the urban infrastructure is relevant, the only significant state change concerns the package's position. We provide exactly one refinement, in which the transport activity is decomposed into a task network with three sub-task: the package has to be picked up at the customer's site (taking order, preparing transport), is then physically moved ("carrying") to the target destination and finally officially delivered (handing over, documenting receipt, follow-up procedures). In our concrete modelling language, the method is defined as follows:

$$\begin{aligned}m_{\texttt{transport\_pi\_ca\_de}} = \langle &\texttt{transport}(p_{\texttt{Package}}, o_{\texttt{Location}}, d_{\texttt{Location}}), \\ \langle \quad &\{\texttt{a:pickup}(a:p), \texttt{b:carry}(b:p, b:o, b:d), \texttt{c:deliver}(c:p)\}, \\ &\{\texttt{a:pickup} \prec \texttt{b:carry}, \texttt{b:carry} \prec \texttt{c:deliver}\}, \\ &\{p \doteq a:p, p \doteq b:p, p \doteq c:p, o \doteq b:o, d \doteq b:d, d \doteq c:p, o \dot{\neq} d\}, \\ &\{\texttt{a:pickup} \xrightarrow{\texttt{Fees\_Collected}(a:p)} \texttt{c:deliver}\} \rangle \rangle\end{aligned}$$

All the technicalities concerning the various procedures are delegated to the implementations of the subtasks; the co-designation constraints thereby ensure that the package identity is passed properly. The last non-codesignation constraint may seem superfluous, but remember that we cannot enforce parameter inequality. On the other hand, we could define an alternative transport implementation this way that handled transports within a single location. This made sense either in order to model personnel activity at

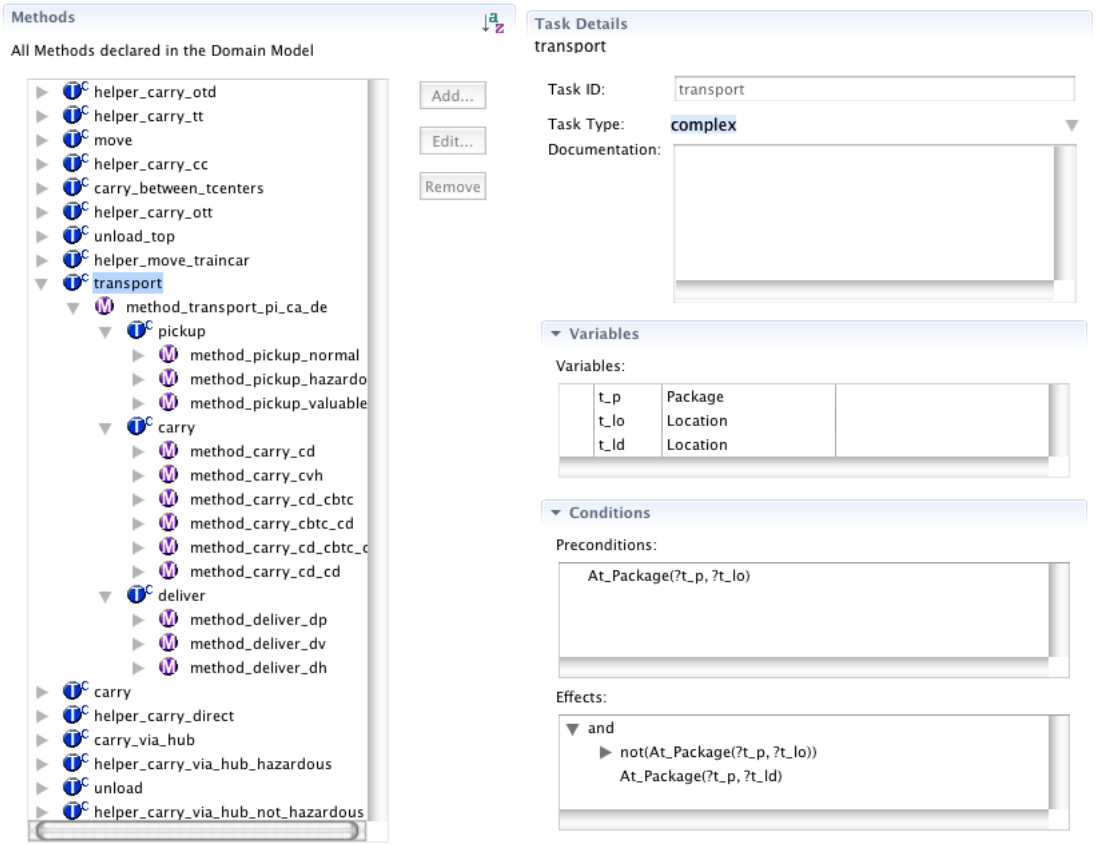


Figure 5.12: The decomposition hierarchy for the complex task schema `transport` and its definition.

a customer's site or in order to define termination networks for decompositions that are specified recursively.

Figure 5.12 shows the PANDA domain editor with the transport task's schema definition on the right and the decomposition hierarchy on the left. The latter shows for each of the three transport sub-tasks, `pickup`, `carry`, and `deliver`, their respective methods. While the implementations of the handover procedures, for which `pickup` and `deliver` stand, are organized according to the package classification, the actual transport procedure, encoded by `carry`, is primarily structured along the possibles trajectories of the package within the defined infrastructures.

The `pickup` and `deliver` task schemata are simply defined as follows:

$$\texttt{pickup}(p_{\texttt{Package}}) = \langle \neg\texttt{Fees\_Collected}(p), \texttt{Fees\_Collected}(p) \rangle$$
$$\texttt{deliver}(p_{\texttt{Package}}) = \langle \texttt{Fees\_Collected}(p), \texttt{Delivered}(p) \rangle$$

In our logistics context, these two tasks implement the business rule: get payed in advance and do only deliver payed packages. The methods depicted in Fig. 5.12 put this rule into context for valuable and hazardous transport goods. That means, after the fees have been collected, an insurance has to be contracted and a transit permission has to be obtained, respectively. The appropriate delivery procedures require these facts as their preconditions and thereby make sure that the package is insured or registered during the whole transport activity.

Following the tradition of the original *UM Translog* domain models, the methods that are provided for the complex task `carry` cover three possibilities to traverse a given infrastructure. The first is available if there exists a direct route to the target location, the second involves intermediate transport centers, and the third handles cases that require inter-region hubs. Our design for the methods for the `carry` task is sketched in the screenshot depicted in Fig. 5.13. On the first two levels of decomposition, the alternative for direct routes,

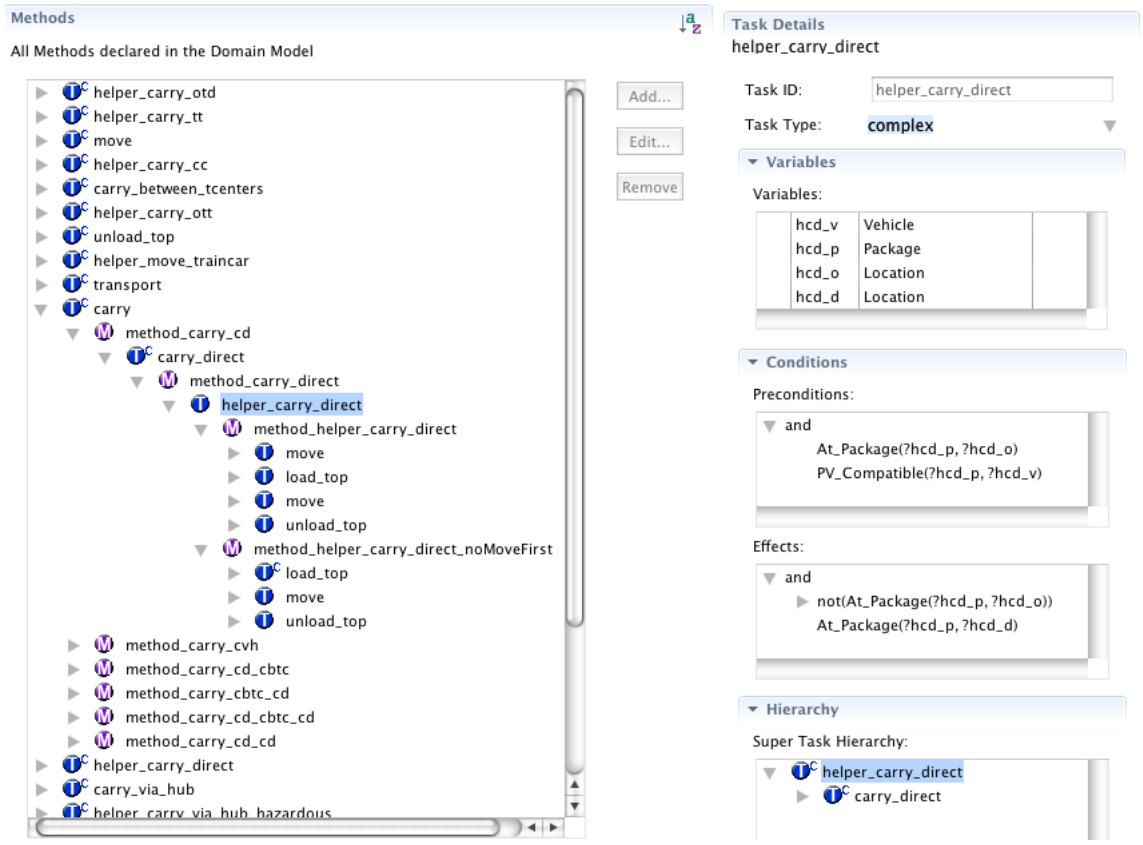


Figure 5.13: The schema definition (right) and decomposition hierarchy for the complex task schema `carry_direct` (left).

`carry_direct`, is transferred into an auxiliary task schema, which introduces the transport vehicle and checks for compatibility between package and vehicle. For the auxiliary schema two implementations are provided: either the package is loaded in the vehicle (still abstract), then the vehicle travels to the destination location, and finally the package is unloaded, or the vehicle has to be moved to the customer location first.

Abstract loading and unloading tasks have a precondition/effect structure like an action for movement, since the package is relocated inside and outside the vehicle:

$$
\begin{aligned}
&\texttt{load}(p_{\texttt{Package}}, v_{\texttt{Vehicle}}, l_{\texttt{Location}}) = \\
&\quad \langle \quad \texttt{At\_Package}(p,l) \wedge \texttt{At\_Vehicle}(v,l) \wedge \texttt{PV\_Compatible}(p,v), \\
&\qquad \neg\texttt{At\_Package}(p,l) \wedge \texttt{At\_Package}(p,v) \quad \rangle \\
&\texttt{unload}(p_{\texttt{Package}}, v_{\texttt{Vehicle}}, l_{\texttt{Location}}) = \\
&\quad \langle \quad \texttt{At\_Package}(p,v) \wedge \texttt{At\_Vehicle}(v,l), \\
&\qquad \neg\texttt{At\_Package}(p,v) \wedge \texttt{At\_Package}(p,l) \quad \rangle
\end{aligned}
$$

The decomposition hierarchy for the loading procedure is displayed in Figure 5.14. All tasks on this expansion level are primitive and reflect the different kinds of loading activities, ranging from loading regular packages in trucks to filling liquids in tanks and driving livestock into transport vehicles. A simple example for the primitive loading operators is also depicted in the figure. Flatbed transport vehicles, for instance, have to be loaded with the assistance of a crane. This specific `Equipment` is used for bulky "packages" like wood, etc. Readers who are familiar with robotic application scenarios will immediately recognize the

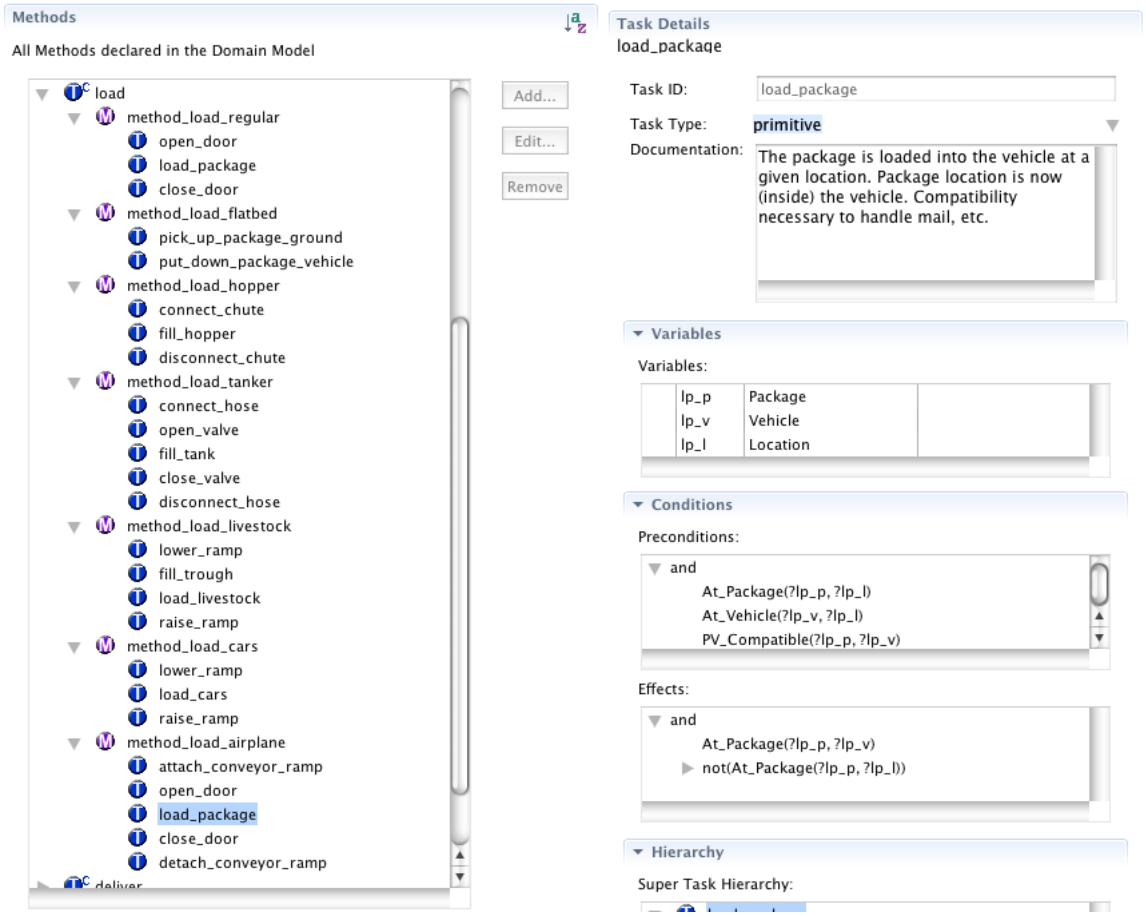


Figure 5.14: The decomposition hierarchy for the complex task schema `load` (left) and the operator schema definition `load_package` (right).

usual operator structure.

$$\begin{aligned}&\texttt{pick\_up\_package\_ground}(p_{\texttt{Package}}, c_{\texttt{Crane}}, l_{\texttt{Location}}) = \\ &\quad \langle \quad \texttt{Empty}(c) \wedge \texttt{Available}(c) \wedge \texttt{At\_Equipment}(c,l) \wedge \texttt{At\_Package}(p,l), \\ &\qquad +\texttt{At\_Package}(p,c)\ -\texttt{Empty}(c)\ -\texttt{At\_Package}(p,l) \quad \rangle\end{aligned}$$

It is also easy to verify that with this loading procedure and a symmetrical unloading operator, the depicted method `method_load_flatbed` implements the abstract loading task.

Let us finally inspect the tasks for actually moving the transport vehicles. The complex task `move` is thereby abstracting away the concrete location connections:

$$\begin{aligned}&\texttt{move}(v_{\texttt{Vehicle}}, o_{\texttt{Location}}, d_{\texttt{Location}}) = \\ &\quad \langle \quad \texttt{At\_Vehicle}(v,o), \neg\texttt{At\_Vehicle}(v,o) \wedge \texttt{At\_Vehicle}(v,d) \quad \rangle\end{aligned}$$

Its decomposition hierarchy is depicted in Fig. 5.15: from the concrete movement point of view, all transportation vehicles behave essentially the same way and are therefore commonly represented in the following primitive operator.

$$\begin{aligned}&\texttt{move\_vehicle\_no\_traincar}(v_{\texttt{Vehicle}}, o_{\texttt{Location}}, r_{\texttt{Route}}, d_{\texttt{Location}}) = \\ &\quad \langle \quad \texttt{Connects}(r,o,d) \wedge \texttt{Available}(v) \wedge \texttt{Available}(r) \wedge \texttt{RV\_Compatible}(\texttt{r},\texttt{v}) \wedge \\ &\qquad \texttt{At\_Vehicle}(v,o), \\ &\qquad +\texttt{At\_Vehicle}(v,d)\ -\texttt{At\_Vehicle}(v,o) \quad \rangle\end{aligned}$$

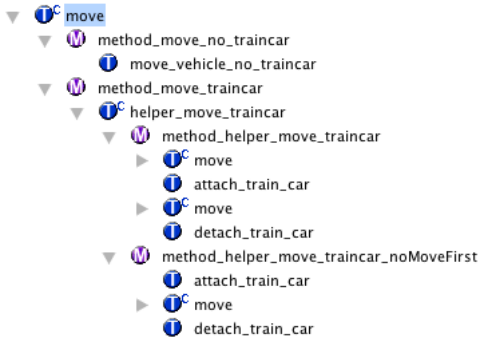


Figure 5.15: The decomposition hierarchy for the complex task schema `move`.

The only exception are train cars, which are moved indirectly by the movement of their pulling train. A train car "appears" at its destination location at the moment it is detached from its train. An implementation of this movement is specified by the following method:

$$
\begin{aligned}
&m_{\texttt{method\_helper\_move\_traincar}} = \\
&\langle\, \texttt{helper\_move\_traincar}(tc_{\texttt{Traincar}}, t_{\texttt{Train}}, o_{\texttt{Location}}, d_{\texttt{Location}}), \\
&\langle \{\texttt{a:move}(a{:}v, a{:}o, a{:}d), \texttt{b:attach\_train\_car}(b{:}t, b{:}tc, b{:}l), \\
&\quad \texttt{c:move}(c{:}v, c{:}o, c{:}d), \texttt{d:detach\_train\_car}(d{:}t, d{:}tc, d{:}l)\,\}, \\
&\quad \{\texttt{a:move} \prec \texttt{b:attach\_train\_car}, \texttt{b:attach\_train\_car} \prec \texttt{c:move}, \\
&\quad\ \texttt{c:move} \prec \texttt{d:detach\_train\_car}\,\}, \\
&\quad \{tc \doteq b{:}tc, tc \doteq d{:}tc, t \doteq a{:}v, t \doteq b{:}t, t \doteq c{:}v, t \doteq d{:}t, \\
&\quad\ o \doteq a{:}d, o \doteq b{:}l, o \doteq c{:}o, d \doteq c{:}d, d \doteq d{:}l, \\
&\quad\ o \dot{\neq} d, a{:}o \dot{\neq} a{:}d\,\}, \\
&\quad \{\texttt{a:move} \xrightarrow{\texttt{At\_Vehicle}(b:t, b:l)} \texttt{b:attach\_train\_car}, \\
&\quad\ \texttt{a:move} \xrightarrow{\texttt{At\_Vehicle}(c:v, c:o)} \texttt{c:move}, \\
&\quad\ \texttt{b:attach\_train\_car} \xrightarrow{\texttt{Connected\_To}(d:tc, d:t)} \texttt{d:detach\_train\_car}, \\
&\quad\ \texttt{c:move} \xrightarrow{\texttt{At\_Vehicle}(d:t, d:l)} \texttt{d:detach\_train\_car}\,\}\,\rangle\,\rangle
\end{aligned}
$$

The task expression `a:move` is intended to find the appropriate train to which the transporting train car is attached by `b:attach_train_car`. After that, the train moves to the given destination location d, where the train car is finally detached. The variable constraints thereby fix coherent bindings and the causal links provide a appropriate commitment within this implementation. During the movement, the train car is not at any specific location and changes from a state of *being* somewhere into a state of *being connected to* some train. The attachment operators in the following definitions detail that:

$$
\begin{aligned}
&\texttt{attach\_train\_car}(t_{atc} : \texttt{Train}, tc_{atc} : \texttt{Traincar}, l_{atc} : \texttt{Location}) = \\
&\quad \langle\ \texttt{At\_Vehicle}(t_{atc}, l_{atc}) \wedge \texttt{At\_Vehicle}(tc_{atc}, l_{atc}) \wedge \neg\texttt{Connected\_To}(tc_{atc}, t_{atc}), \\
&\qquad +\texttt{Connected\_To}(tc_{atc}, t_{atc}) - \texttt{At\_Vehicle}(tc_{atc}, l_{atc})\ \rangle \\
&\texttt{detach\_train\_car}(t_{dtc} : \texttt{Train}, tc_{dtc} : \texttt{Traincar}, l_{dtc} : \texttt{Location}) = \\
&\quad \langle\ \texttt{At\_Vehicle}(t_{dtc}, l_{dtc}) \wedge \texttt{Connected\_To}(tc_{dtc}, t_{dtc}), \\
&\qquad +\texttt{At\_Vehicle}(tc_{dtc}, l_{dtc}) - \texttt{Connected\_To}(tc_{dtc}, t_{dtc})\ \rangle
\end{aligned}
$$

The remaining parts of the task model deal with alternative implementations of the presented procedures. In general, there is a large number of switching and antagonistic action schemata, that means, action definitions that solely negate their precondition. Examples are collecting fees, obtaining permits for hazardous

transports, connecting and detaching equipment, and affixing warning signs. All these tasks basically have the form of the following operator schema for opening and closing vehicle doors:

$$\texttt{open\_door}(v_{od} : \texttt{Regular\_Vehicle}) = \langle \neg\texttt{Door\_Open}(v_{od}), +\texttt{Door\_Open}(v_{od}) \rangle$$
$$\texttt{close\_door}(v_{cd} : \texttt{Regular\_Vehicle}) = \langle \texttt{Door\_Open}(v_{cd}), -\texttt{Door\_Open}(v_{cd}) \rangle$$

With this definition, we conclude our presentation of the action model. Space limitations made it necessary to focus this section on 8 complex (5 methods) and 7 primitive task representatives, which is only one sixth of the model. For the complete domain specification, again, we refer the reader to the technical documents on our website.

Concluding Remarks

As it has been intended by the authors of the original versions [8, 290], the *UM Translog* domain model is certainly a step into a direction that leads away from playing with toy-problems in toy-domains onto a path towards real-world planning applications. It is a reasonable benchmark because it allows to define a variety of completely different problem structures, ranging from singular transports to groups of jobs, from a sparse infrastructure to large, highly interconnected environments. Its diversity is one key argument to prefer this domain over small-sized benchmarks that define their challenge quality over mere sub-problem replication. It is also very demanding in terms of the minimum plan length; a simple problem that requires one regular package to be delivered by a truck has a minimum length of 11 plan steps and if, for instance, an armored transport is involved a solution consists of at least 16 steps. Consequently, an interaction of multiple delivery jobs typically occurs over relatively long paths of actions, either as a conflict situation or as a mutual dependency due to joint resources.

The commonly agreed advantage of the the *UM Translog* domain is that it features a mixture of complexity and accessibility. First, it is impressively large in terms of number of action schemata and object types while it is at the same time reasonably structured as regards to independent transportation methods, infrastructure, and geography. Second, the model is close to realistic but at the same time it remains easily adjustable with respect to the supported domain features, for example, temporal information can be introduced, enhanced loading procedures can be added, and the like. However, we feel the definitive need for reconsidering these advantages within the AI planning community. A fairly obvious counter-argument against the value of "being large" is the question of how to define difficulty for planning benchmarks in general and for *UM Translog* in particular. As Helmert already showed in [132], the IPC version of this domain is on the borderline between NP and P, depending on whether or not movements are resource-constrained. According to this result the "operator base" of our translation, which does not define resources yet, is of a lower-order polynomial time complexity with respect to the problem of plan existence and becomes NP-complete for finding optimal plans with limited resources. As useful as such complexity considerations may be, they do not take into account the impact of procedural knowledge in the respective domain models (for instance, [84] only presents a general classification of HTN planning in terms of expressivity). In addition, the abstractions made for a complexity classification largely prevent that the respective result can be applied to a difficulty assessment of concrete problem instances: for example, it is not clear whether mixing an increasing number of transport types introduces more effort for finding a solution than increasing the distance of the target locations for a stable number of jobs. For the most part, the question of problem difficulty (within a certain class of problems) remains open, or, to put it more provocatively, it is harder to define a set of progressively more challenging *UM Translog* scenarios than just dropping more blocks on the table.

In contrast to the conciseness of the previously presented *Satellite* domain model, consistency and plausibility become salient issues for *UM Translog*. With making the translated sort hierarchy visually accessible by our editing tools, it becomes apparent that the originally intended typing system needs some re-work. As it has been noted above, the knowledge engineering process behind the domain definition for the UMCP system led to a task hierarchy that evidently has been designed (probably in a top-down manner) to the purpose of an efficient task decomposition and *not* in the view of implementation abstractions. That means that, for instance, the ad-hoc typing predicates are only used consistently along a decomposition path and not cross-method-wise. Using the networks' `formula` constraints as a suggestion for the tasks' intended

preconditions and effects[8] also revealed the problem that many causal commitments on the abstract level cannot be reproduced by the operators in the respective implementations. As a result, practically every task network's complex task can only be defined with trivial preconditions and effects. We consequently had to re-organize, for example, the `carry` definitions accordingly. It has also to be noted that the weak semantic support for UMCP models obviously could not even prevent typographical errors in the specification files such that we occasionally encountered task conditions that used predicates with the wrong signature, and the like. There also exists one translation into the OCL_h formalism [182] but these authors did not report to have encountered any semantic problems. We must assume that they re-built the model solely from the textual documentation and the operator schema definitions.

Regarding the plausibility or adequacy of a domain model, we have to admit that the current version of the *UM Translog* model is not completely satisfying. The first unresolved issue is some sort of obscurity in the sort hierarchy with respect to modelling "specialities" and "physical properties" of packages and vehicles. Some of the latter have to be moved into the special feature category in order to make the model conceptually clearer. In any case, the model should provide a more coherent relationship between "physically" implied compatibilities of vehicles and packages. For example, liquid packages require a tanker transport, while being a car package and being able to transport them is represented by the literally identical secondary type. Similar observations have been made by the translation into the OCL_h formalism [182].

It is also questionable whether or not a *property* should be modelled as a sort in the first place. The original model suggested this large number of respective type predicates and was thereby able to verify sort relationships explicitly in precondition formulae and task network constraints. Although our translation into the sort hierarchy lost that flexibility of annotating type information and required the somewhat clumsy work-around with atomic compatibility formulae in the initial state for each individual, our modelling framework however provides an extremely valuable and effective mechanism for verifying the model's consistency. Concerning the compatibility information in the initial state, note that although it is defined on a per-instance basis, determining the concrete information is computationally tractable because the respective relation symbols are rigid ones. As a "meta-remark" on the issue of properties versus types, we believe that there is a mis-balance between the sort hierarchy and the relation definitions: Specifying three times more sorts than relations indicates to us that the former need to be reduced in favour of the latter.

The most apparent implausibility could be fixed, though. The *UM Translog* in the UMCP distribution was not able to solve problem instances in which two transport tasks had their origin in the same location, or in which packages had to be picked up in the context of an overlapping jobs. Also, vehicles were not allowed to be positioned on the pick-up location and that made follow-up jobs impossible to plan for. Since we intended this domain to be processable in a reasonable way by an HTN planning configuration (see evaluation chapter 6), we introduced appropriate methods that perform just the loading procedure, methods that make the vehicle drive to the customer location before loading, and so forth.

The final conclusion on the *UM Translog* domain is twofold: the logistics scenario is a very appealing application environment and provides many challenges. The problem structure is very rich and very complicated "real-world" situations can be specified. On the other hand, the domain is not very clearly laid out and there are many errors a model or problem writer can introduce, from minor inadequacies to substantial inconsistencies.

5.2.3 CrissCross

Introduction

While the above domain models are inspired by real-world application scenarios, the *CrissCross* domain model describes an artificial environment that has been specifically designed to provide a clean experimental frame for evaluating hybrid planning strategies. Its rationale is the following: What makes problems particularly challenging in the context of hybrid planning is a decomposition hierarchy in which a large

[8] In the UMCP modelling formalism, atomic conditions can be defined to hold between task expressions or before or after a task expression. We derived from these sort of constraints what the intended causal structure in the logistics scenarios has to be.

number of causal interactions between subsequent decompositions occur. It is the cross-wise dependencies and influences that gave the domain its name. The induced search trees typically contain many decision situations that are very sensitive to early commitment like a premature task insertion (cf. [298]) or an inadequate conflict resolution. Any strategy that relies too much on one paradigm during plan generation, namely POCL or HTN-like refinements, will therefore fail to find a solution efficiently: Precautionary disregarding POCL modifications misses opportunities to deal with causal interactions in the more manageable abstract plans, and eagerly addressing fixing the causal structure may unnecessarily introduce inconsistencies with respect to some decomposition refinements. Successful strategies hence perform a *balanced* mixture of both.

The structure of the *CrissCross* domain is very simple in order to be easily readable for the human experimenter and also to allow for an adjustable level of "difficulty" in an evaluation setting.

The Language

A sort hierarchy imposes in general a strong structure on the refinement space in terms of separating solutions from inconsistent plans. This is mostly because decomposition refinements are often not only defined as "alternative" implementations for an abstract task but also as implementations for a specific (sub-) class of objects. We therefore restrict $\mathscr{L}_{\text{CC}}$ to a single-sorted language with the respective $\mathscr{Z}$ containing only one symbol: `Char`, standing for "character". The language furthermore includes flexible unary relation symbols from `A` to `F`, no rigid relation symbols, and no function symbols. The *CrissCross* language is consequently defined by the following tuple:

$$\mathscr{L}_{\text{CC}} = \langle \mathscr{Z}_{\text{CC}}, \leq_{\text{CC}}, \mathscr{R}_{r_{\text{CC}}}, \mathscr{R}_{f_{\text{CC}}}, \mathscr{F}_{r_{\text{CC}}}, \mathscr{F}_{f_{\text{CC}}}, \mathscr{V}_{\text{CC}}, \mathscr{T}_{p_{\text{CC}}}, \mathscr{T}_{c_{\text{CC}}}, \mathscr{E}_{\text{CC}} \rangle$$

with

$$\begin{aligned} \mathscr{Z}_{\text{CC}} &= \{\texttt{Char}\} & \leq_{\text{CC}} &= \emptyset \\ \mathscr{R}_{r_{\text{CC}}} &= \emptyset & \mathscr{R}_{f_{\text{CC}}} &= \{\texttt{A}_{\texttt{Char}}, \texttt{B}_{\texttt{Char}}, \texttt{Char}_{\texttt{Char}}, \texttt{D}_{\texttt{Char}}, \texttt{E}_{\texttt{Char}}, \texttt{F}_{\texttt{Char}}\} \\ \mathscr{F}_{r_{\text{CC}}} &= \emptyset & \mathscr{F}_{f_{\text{CC}}} &= \emptyset \\ \mathscr{V}_{\text{CC}} &= \{\ldots\} & \mathscr{E}_{\text{CC}} &= \{\ldots\} \end{aligned}$$

As usual, the elementary operations for this language, $\mathscr{E}_{\text{CC}}$, can be directly obtained from the flexible relation symbols. Furthermore, we will generate variable symbols on demand.

We provide symbols for four primitive task schemata `p`, `q`, `r`, and `s`, and a set of complex tasks for building abstractions of operator combinations like `pANDr`, `qANDs`, etc.

$$\begin{aligned} \mathscr{T}_{p_{\text{CC}}} = \{&\texttt{p}_{\texttt{Char Char}}, \texttt{q}_{\texttt{Char Char}}, \\ &\texttt{r}_{\texttt{Char Char Char}}, \texttt{s}_{\texttt{Char Char Char}}\} \end{aligned} \qquad \begin{aligned} \mathscr{T}_{c_{\text{CC}}} = \{&\texttt{pANDr}_{\texttt{Char Char Char Char}}, \texttt{qANDs}_{\texttt{Char Char Char Char}}, \\ &\texttt{pANDq}_{\texttt{Char Char Char Char Char}}, \\ &\texttt{rANDs}_{\texttt{Char Char Char Char Char Char Char}}, \\ &\texttt{addPR}_{\texttt{Char}}, \texttt{addPQR}_{\texttt{Char}}, \texttt{addAll}_{\texttt{Char Char}}\} \end{aligned}$$

Tasks and Methods

The *CrissCross* domain model contains four primitive task schemata. The schemata `p` and `q` are merely adding a state feature depending on the presence of another one and therefore stand for "rule-like" tasks, for example: "for any character x, if $\texttt{A}(x)$ holds then also does $\texttt{C}(x)$". Furthermore, operators `r` and `r` represent the typical "feature switching" schemata that undo (part of) their own precondition.

$$\begin{aligned} \texttt{p}(pA_{\texttt{Char}}, pC_{\texttt{Char}}) &= \langle \texttt{A}(pA), +\texttt{C}(pC) \rangle \\ \texttt{q}(qB_{\texttt{Char}}, qD_{\texttt{Char}}) &= \langle \texttt{B}(qB), +\texttt{D}(qD) \rangle \\ \texttt{r}(rB_{\texttt{Char}}, rC_{\texttt{Char}}, rE_{\texttt{Char}}) &= \langle \texttt{B}(rB) \wedge \texttt{C}(rC), +\texttt{E}(rE)\ -\texttt{B}(rB) \rangle \\ \texttt{s}(sA_{\texttt{Char}}, sD_{\texttt{Char}}, sF_{\texttt{Char}}) &= \langle \texttt{A}(sA) \wedge \texttt{D}(sD), +\texttt{F}(sF)\ -\texttt{A}(sA) \rangle \end{aligned}$$

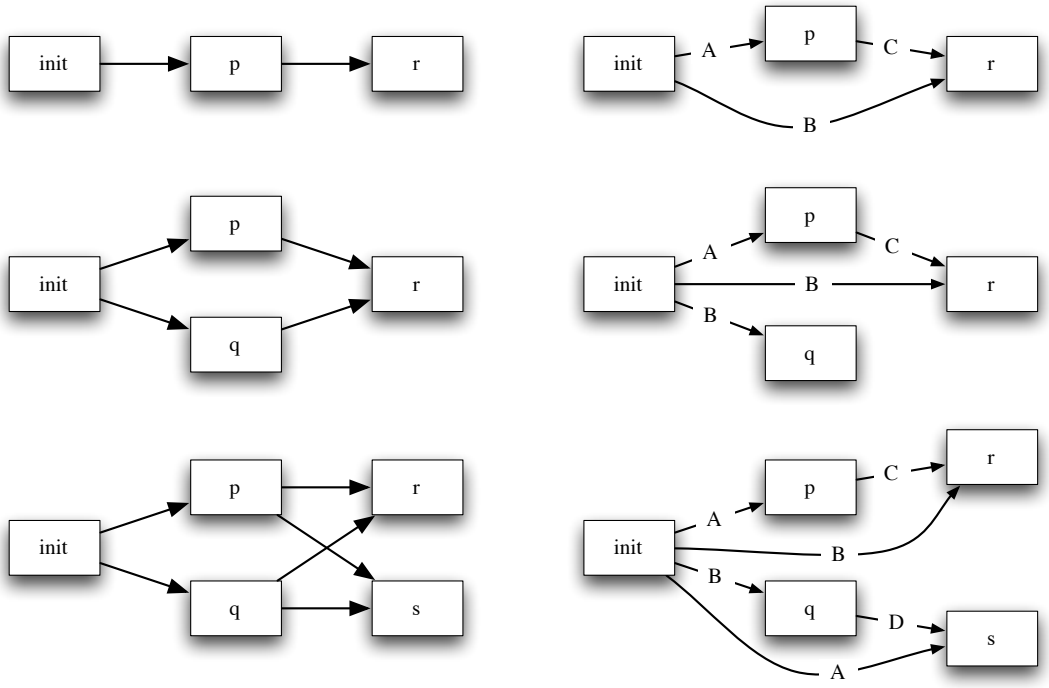


Figure 5.16: The temporal (on the left) and causal (on the right) structure of solutions to benchmark problems in the *CrissCross* domain.

The key problem with respect to early commitment lies apparently in the cross-wise dependencies between the operators `p` and `r` on the one side and the pair `q` and `s` on the other: While `p` provides a part of `r`'s precondition, `r` in turn falsifies `q` which is needed for `s`, and vice versa. Based on these definitions, Figure 5.16 depicts three scenarios for the structure of primitive plans that represent isolated standard situations in practically any planning process: (1) The situation shown at the top is a linear sequence of actions, in which a necessary causality is passed by the middle action. Any threat resolution that addresses the middle action implicitly affects the tail action and stretches the passed causal link which in turn becomes more vulnerable to interactions. (2) The plan in the middle stands for basically the same situation, is however somewhat more complicated due to the double use of a condition and the fact that the parallel task `q` cannot be placed after `r` because it undoes `q`'s precondition. (3) The situation at the bottom of Figure 5.16 often occurs unintendedly when two parallel strands turn out to share resources, etc. It resembles the "Gift of the Magi" paradox for demonstrating the significance of the upward solution property in hierarchical planning (for instance, see practical planning section in [223]).

All three cases have in common, that the temporal aspects of the solutions are not mirrored by the causal occurrences: neither are the causal chains separated as the temporal parallelism suggests, nor is the extension of causal dependencies limited to subsequent execution layers. Please remember that these are standard plan generation situation, in the *CrissCross* domain however, they are emphasized and examined in isolation. The remaining domain model components are designed such that any solution to appropriately defined problems necessarily consists of the above structures. Furthermore, it is the mutual threat situations in the causal chains that makes every *CrissCross* problem having exactly one solution (modulo isomorphisms).

The operator repertoire is now to be orchestrated in methods' task networks such that the respective complex tasks are going to fool a naive strategy on the more abstract plan level. We are consequently building abstractions of these tasks that obfuscate the described causal conflict by concealing negative effects of the subsumed operators:

$$\texttt{addPR}(aprE_{\texttt{Char}}) = \langle \top, \texttt{E}(aprE) \rangle$$
$$\texttt{addPQR}(apqrE_{\texttt{Char}}) = \langle \top, \texttt{E}(apqrE) \rangle$$

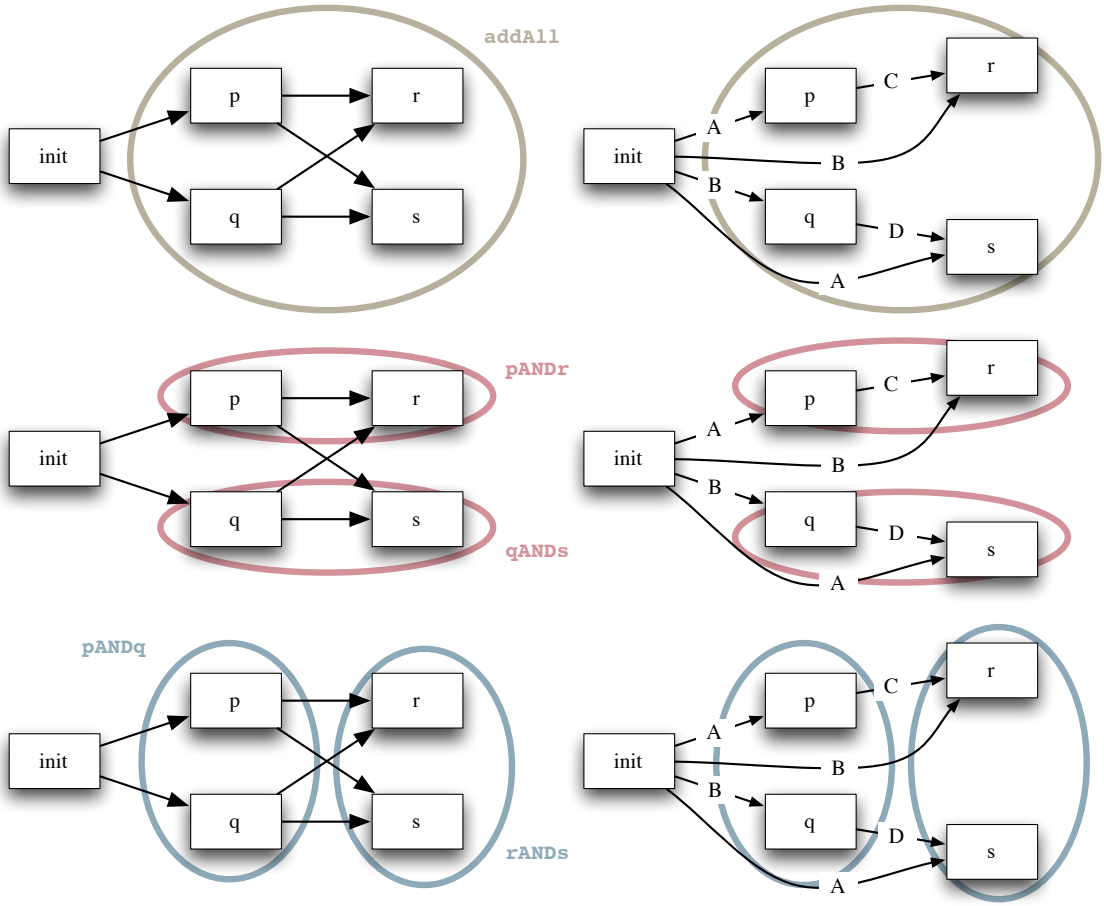


Figure 5.17: How the decomposition hierarchy relates to the temporal (on the left) and causal (on the right) structure of *CrissCross* solutions. Sand colour represents the instant introduction of all four operators, red the causal chain-wise decomposition, and blue the operator type-oriented method.

The `addPR` task is provided with a method that decomposes the task into a sequence of operators `p` and `r`, while `addPQR` does so for the appropriate triple. Both complex task schemata emphasize `r`'s effect specification with respect to the `E` relation but hide the negation of `B`. If more than one instance of these complex tasks occurs in one plan, the co-designation of respective parameters on the abstract level inevitably produces un-resolvable threats on more concrete levels.

Another dimension of analyzing hybrid planning domain models is that of how the decomposition structure influences the system's performance. For the *CrissCross* domain, we introduce a top-level complex task `addAll` that is eventually to be decomposed into the set of all four operator instances. A strategy can thereby choose between the application of three methods, each of which implies different requirements that the strategy has to fulfill in order to be able to finalize plan generation successfully. Let the initial task network of a problem contain several `addAll` instances. The first defined method instantly introduces all four primitive task instances, which leaves the strategy in a situation in which it has to deal with (numerous) causal conflicts on the primitive task level. A second and a third method establish an intermediate abstraction level with the two complex tasks `pANDr` and `qANDs`, respectively the actions `pANDq` and `rANDs`. The intermediate tasks group either the causal chains or the temporal layers. Figure 5.17 shows the three decomposition alternatives.

Concluding Remarks

The *CrissCross* domain is an artificial domain model that is geared to providing a controllable level of difficulty to hybrid planning strategies. Its challenging features arise from causal interactions on the primitive level that are, firstly, not visible on the abstract level and that, secondly, are divided up into different complex tasks. In this way, the *CrissCross* domain provides an experimental evaluation with benchmark problems that inflict a penalty on unbalanced planning strategies that pertinaciously follow a pure POCL or HTN schema and are hence unable to take opportunities. As our experiments will prove in Chap. 6, *CrissCross* problems are in fact hard to solve for many strategies. The challenge is apparently not generated by the size of the domain in terms of number of object types, attributes, or available task schemata. It is moreover the structural characteristics of the domain that are responsible for the problems' difficulty.

5.3 Discussion

5.3.1 Related Planning Architectures

The advantages of a strong modularization for code generation and maintenance are common sense in software development. With planning and scheduling technology being embedded in an increasing number of mission-critical applications, research focuses on longer exclusively on algorithmic issues but more frequently on architectural features that are relevant for this kind of software. Traditionally, multiagent techniques have been proven to provide an appropriate architectural basis for "AI applications". We understand that recent technical advances moved from the agent to the "service" paradigm. Various service related metaphors came into fashion ranging from "Grid Computing" and "Web Services" to "Service Oriented Architectures". Since service discovery is not involved in our configurations, these types of architecture are subsumed by our notion of multiagency.

An established planning system of this kind is certainly the O-PLAN architecture [261], which features some characteristics that are similar to our system design. Its blackboard (*plan state*) is modified by combined detector-modificator agents (*knowledge sources*). They announce their highest ranked flaws on a common agenda upon which the strategy (*controller*) works. Assistant agents (*support modules*) are used interactively by the knowledge sources and answer questions about current plan features. The strategy selects the highest prioritized flaw and triggers the associated knowledge source, which in turn performs its highest prioritized modification. Key differences are here (1) the agent granularity is not variable, that means, there is exactly one knowledge source for each flaw/modification class and (2) the strategy options are formulated in terms of the supported flaws. Thus, it lacks the sort of flexibility that is the main characteristic of our approach. We can furthermore simulate this kind of controller by selecting modification steps according to modificator names for which a one-to-one mapping to the flaw types should be provided. O-PLAN has been recently extended by a workflow-oriented infrastructure, called the I-X system integration architecture [256]. A plug-in mechanism serves as an interface to various application-tailored tools. However, the planner itself is still a monolithic system structure.

The multiagent Planning Architecture MPA [286] relies upon a very generic agent-based approach. It executes an agent society in which designated coordinators decompose the planning problem into sub-problems. These sub-problems are solved by subordinated groups of agents, which may again decompose the problem into smaller units. The individual agents return their solutions to their associated manager agent, which synthesizes the overall solution of its sub-agents. Communication of queries and results is based on the highly abstract KQML formalism. To our knowledge, no (standardized) middleware functionality has been incorporated. This architecture is considerably more general than our presented system, which gives additional overhead to the agent design for a specific planning system incarnation. Another problem is certainly to find a sensible decomposition technique for a planning problem and a proper methodology of synthesizing and combining the partial solutions, including the necessary capabilities for supporting meaningful communication of both. Besides that, the main difference to the proposed approach is that all solutions are prepared on the lowest agent level and then have to be synthesized on higher levels. Note that a comparable system behavior can be reached if we allowed for sequences of modification steps to be proposed and executed.

MPA's use of views corresponds to our use of XML as communication base, which allows for selective parsing, and the presence of *Service Agents*.

Some approaches try to present planning functionality as services to a larger application environment: The SIADEX architecture [70] uses XML-RPCs[9] for building a distributed planning environment that is accessible via standardized protocols – the architecture thereby decouples a (monolithic) planning server, knowledge base management tools, and an execution monitoring. In order to address connectivity issues, some planning systems offer their functionality as web service. Examples are the CGI-based[10] O-PLAN interface [257] and the approach in [267], where a planner uses SOAP for communication and the Web Services Description Language for presentation of the service. Although this view helps in enhancing the accessibility of planning software, the system (development) itself is not directly supported.

Focusing again on architectural issues, appearingly similar frameworks mostly deal with the connection of a deliberative high level planning component with reactive or rule-based execution monitoring and control modules [19, 33]. This is especially the case for the integration of plan generation and plan execution. We sometimes find several special-purpose planning systems deployed concurrently, for instance, the "planning experts" in the New Millennium Remote Agent architecture [194]. They contribute to different aspects, respectively sub-problems of the overall planning problem. To our knowledge, the planning or deliberation components in all these approaches are monolithic "standard" systems with no specific development, configuration, or distribution support.

There also exist application frameworks or class-library toolboxes for building planning applications. Representatives for this concept are ASPEN (Automated Planning Scheduling and Planning Environment) [102] and work in the context of the Planform project [217]. They provide planning specific infrastructural data structures, classes of supportive inference mechanisms for common planning sub-problems, and algorithmic templates for rapidly building planning applications "out-of-the-box". These frameworks concentrate on delivering easy to use or configure pieces of a planning software independent from the architecture, while our view is more centered on providing an architecture in which arbitrary parallel planning system designs can be realized.

By supporting the designing process of planning software from the point of view of software engineering, [293] tries to meet several challenges in search-based applications. The most important is certainly that of having a reusable search engine. As we described above and in previous chapters, our planning agents can be easily re-used and tailored in our framework. An important point is the flexible composition of search techniques (this topic is addressed in detail in Sec. 4.1). A last challenging feature is that of the "obscurity of module boundary". In the presented system, the decomposition into agents is done along the flaw detection process and plan modification functionality, which seems rather intuitive and adequate to us.

5.3.2 Lessons Learned

The Software Framework

An important question is that of our justification to deal with an agent oriented system at all, namely: Did we fall into an "agent pitfall" [289], believing that agent technology is a "silver bullet"? We are confident that neither our agent design is too generic for our purpose (cf. pitfall 3.3) nor that we do see agents everywhere (7.1). On the contrary, we use them more in the way of a naturally modularized software system in which a least commitment planner can easily be decomposed. The previous sections also give evidence that we are aware of related technology and existing architectures and standards (5.1, 6.1, and 8.2), since we build on established formats and architectural features (for instance, the messaging mechanism is compatible to the JAVA Message Service).

[9] XML-RPC is a specification and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet. It uses HTTP as the transport and XML as the encoding language. See `http://www.xmlrpc.com`

[10] Common Gateway Interface: Information systems often use HTTP gateways for accessing for their data. The CGI is a standard for integrating such gateway scripts and programs. See `http://www.w3.org/CGI/`

Concerning performance issues of the agent framework, it has to be noted that in our first experiments the system needs tens of microseconds for handling the communication and coordination overhead per life-cycle for a society of twenty agents on one physical machine. This overhead can certainly be neglected. The current framework does not yet exploit concurrency on several computers (aka as pitfall 5.2), but this rather technical matter is part of our future work.

Despite the numerous advantages of our knowledge-based middleware, we encountered some problems regarding the inclusion of semantic web components. The semantic web is a relatively young and very active field of research. During our developments, this implied that DAML has been superseded by OWL[11] as the de-facto standard representation language for semantic web content. Although OWL shares many features with DAML and the relevant subset persisted into OWL, the practical implications of this situation were quite problematic. It became increasingly difficult to maintain a set of tools and libraries that agreed on the same language: OWL was not finally defined yet and in addition divided into layers of increasing expressivity. When DAML support finally started to disappear, it caused an extreme amount of work to re-build the project.

Another problem is the integration of planning domain knowledge in the description logic formalism of DAML/OWL. To our knowledge, no satisfactory solution has been proposed in the literature yet. The main discrepancy lies of course in the dynamics of planning models that cannot be reflected in standard description logics. The current work-around of using DAML/OWL ontologies as a dictionary for the planning domain vocabulary is feasible, constitutes however no additional value for the efforts of maintenance.

Domain Model Specifications

As it has been stated in the introductory section, a planning and scheduling system goes hand in hand with its domain models and problems. Insights gained with the respect to one aspect have an immediate impact on understanding the other. What started out as an effort to quickly produce a number of benchmark domains for an empirical evaluation turned into a lesson about what a formally based modelling method like ours can achieve in terms of model consistency. The contribution of our formal framework is a sound and therefore reliable basis that helps in writing consistent domain models as well as it assists in finding the errors in flawed ones. The implications of being tied to such a clear model semantics during specification must never be underestimated, and the surprises in our translation effort of the *UM Translog* domain give evidence to that.

This experience also convinces us to believe that the relevance of a domain model suite for the planning community must not be reduced to the question of how well it is suited to measure *planning systems'* performance but also take into account the *knowledge engineering* issues it raises. To our surprise, this topic has been largely ignored and not yet been pursued with the impetus we believe it requires, though the lately installed *International Competition in Knowledge Engineering for Planning* is certainly an important step towards addressing it.

Another lesson learned is the discovery of numerous "anomalies" and "clumsy parts" of the models. But although it is a trivial fact that anyone who specifies domain models gains experience in recognizing flawed formalizations and writing "better" models, we are not aware of *any research* about the issue of *quality* of procedural knowledge in the area of planning. In terms of our framework, this includes the question of how strong the defined refinements separate the search space and how fast they narrow down possible implementations to a set of solutions. In this view, knowledge quality is not only a matter of aesthetics but has direct implications on the re-usability of model components on the one side and on the performance of search strategies on the other.

[11] Web Ontology Language, see `http://www.w3.org/TR/owl-ref/`

5.3.3 Perspective

Future versions of our software framework will not only keep the agents but also the planning state and agent messages in the system ontology and ABox in order to extend the verification capabilities of the system, including multiple ABoxes to enable multi-session planning. This will make the PANDA system even more robust by reasoning in real-time over the dependencies between system states, possible actions and sent messages that trigger state transitions. To achieve this goal, knowledge about communication (message performatives, sender, receivers, etc.) and interaction (FIPA-protocols and the planning process model) will be incorporated into the description logics representation of the system.

Extracting the hard-coded planning process model and describing it in a declarative manner makes the process model interpretable, that means it can be executed on a generic engine that uses the description as a process template. Since changes to the planning process do not involve a change of code anymore, this allows for an easy tailoring of the planning system to applications. The process model itself an then be verified by transforming it into a petri net representation [197].

As we have pointed out in the presentation of the formal framework, the granularity of flaw detection and modification generating functions allows for defining highly specialized entities. This is obviously supported by a knowledge-based architecture: modifications and their generators are more specific than others and may therefore be (automatically) preferred in proposing a flaw resolving refinement. Such ontologies may even be domain specific such that, for example, a route-planning subsystem can be deployed on-demand – relating domain model features with flaws or modifications is an easy task for the presented architecture.

Regarding our application domains, apart from the obvious need for more well-elaborated formalizations, all presented domain models are of course to be extended to temporal and resource reasoning. For the *Satellite* domain, for instance, we can adopt the respective aspects from the IPC-4 benchmark suite [138]. The *UM Translog* enhancements can be inspired by the appropriate IPC definition [290] and the corresponding demonstration scenarios that have been translated for the SHOP2 system.

5.4 Summary and Conclusion

We have presented in this chapter two facets of fielding the PANDA system, namely the software architecture aspects and the modelling characteristics of the framework.

The software-related presentation focuses on our multiagent architecture in which we realized the implementation of our formal framework. The exploitation of the previously introduced semantics for system components and configurations enables us to use representation and inference techniques of description logics for dealing with knowledge about the PANDA system itself. This extends the capabilities of the architecture significantly in both a functional and a non-functional manner. High level configuration verification can be performed and the system becomes more flexible and configurable as previously implicitly encoded knowledge is made explicit. With the use of application server technology and standardized communication protocols, we have laid the foundation for fielding our system in real-world application scenarios.

Our brief excursus on specifying planning domain models provided us with some insight into the knowledge engineering issues that arise in hybrid planning. We presented the relevant aspects of our model repertoire and gave background information on the rationale behind their design. The set includes a POCL model that has been added procedural information, an HTN benchmark that has been extended to hybrid planning, and an artificial experimentation domain. We detailed their particularities, explained the applied modelling techniques, and shared our experience by discussing the encountered modelling problems. The increased expressivity and flexibility with respect to plain POCL and HTN models apparently comes with a burden to construct the formalization very carefully and sometimes very verbosely. On the other hand, the declarativeness and clear semantics of the framework's formalism allow for a *structured* and *tool-supported* knowledge engineering process for *consistent* planning domain models.

We will use the presented domain models in the successive Chap. 6 as benchmarks for evaluating some of the planning strategies that we have developed in Chap. 4.

6 An Empirical Evaluation of Planning Strategies

THE central question after more theoretically oriented chapters is usually how the approach works out in practice. As a response to that question we decided to conduct a series of systematic experiments on the performance of planning strategies, more precisely on their efficiency, stability, and reliability. The study's primary objective is to analyze these characteristics of a number of instances from our strategy component portfolio as it has been presented in Sec. 4.1. Our goal is to achieve a more general understanding of the search mechanisms and component interdependencies, as well as obtaining deeper insights into the application domains that we deploy (Sec. 5.2). As a secondary objective, the experimentation simultaneously evaluates the practicability of our approach itself, as we gain empirical evidence for the following claims:

1. The presented approach is an effective framework for setting up planning systems with appropriate strategic components,
2. our architecture defines a useful experimental frame of yet unmet flexibility, and
3. we have developed a set of very efficient planning strategies with interesting characteristics.

The first two claims are addressed by the various system configuration instances that participate in the evaluation, which are conceptually individual planning systems. This flexibility of aggregating planning functionality can be used to systematically alter some of the components – in our experiments these are the strategic selection functions – and to evaluate the alternative combinations. The evidence for the third claim are the results that we obtain from the experiments: On the one hand, we will realize direct comparisons to already proven and tested classical strategies, on the other hand, the analyses will provide a more general context for positioning our findings.

The chapter is structured as follows: Section 6.1 describes in detail the aims of the conducted experiments. We begin with defining precisely what we mean by "performance measures" and formulate accordingly our study objective and research questions that guide experimentation and analysis. Sec. 6.2 then gives information about the concrete experimental frame, that means, the participating strategies and the evaluation environment. In particular, it describes all test problems and their essential properties.

The major part of the chapter is devoted to the presentation of the individual results in sections 6.3 to 6.5, where we evaluate the strategies' performance, analyze our findings, and consider inferences on the characteristics of the involved domains and problems. The chapter continues with a comprehensive discussion in which we lift our results from the domain-specific level to a more global perspective (Sec. 6.6.1). We also share our experiences with the technical aspects of the experiments, followed by a short survey of evaluation efforts that have been reported in the literature.

In the perspective section (6.6.4) we show how the described experimentation can and should be continued. We sketch the next steps as well as the issues that are implied as upcoming research topics. Sec. 6.7 summarizes the results and concludes the chapter.

6.1 Study Objective and Research Questions

The primary objective of the experimental evaluation is to assess the *performance characteristics* of a set of planning strategies. More precisely, we want to examine the properties called *efficiency*, *stability*, and *reliability* of selected combinations of strategy components from our portfolio. We define these characteristics as follows:

efficiency: The ratio of obtained solution quality to the required efforts.

stability: A measure for deviations in solution quality over several planning episodes. It covers deviations over multiple runs on the same problem as well as inter-problem variations of quality. The typical scale is defined inversely as $[0,\infty)$ with 0 representing a completely stable strategy and higher values indicating that the strategy produces increasingly unpredictable solution qualities.

reliability: A measure for the success-rate of a strategy to find a solution (of a desired quality) at all. As for stability, reliability is often defined inversely as the ratio of tackled problems to successful runs.

There are of course many candidates for a solution quality metric that may be considered in such experiments, ranging from computation real-time to plan-optimality metrics like the number of plan steps, and the like. We have chosen the size of the search-space that is explored by the strategy, because we want to have a simple measure as well as a qualitative one that reduces the influence factors on the data interpretation. If we considered, for example, a real-time dependent metric, side effects of the implementation would come into play and the results would massively interfere with the computational resources (available hardware, competing processes, etc.). On the other hand, we intend to define solution quality as a binary result of the plan generation process – a solution is either found or not found. We therefore do not have to take into account more complicated interferences between strategies or any other side-effects of optimization processes.

Since the precise meaning of these terms is essential for interpreting our findings, we have to define them in more detail. Please note that the notions of performance characteristics for strategies are inseparably connected to the configurations into which they are embedded.

Definition 6.1 (Efficiency of System Configurations)**.** The *(search-) efficiency* of a system configuration $\mathfrak{C}$ with respect to a set of problems $\{\pi_1,\ldots,\pi_n\}$ is defined via the average number (arithmetic mean) of plans that are analyzed for flaw detection by that configuration in all terminating runs of the refinement planning algorithm (Algorithm 2.2 on page 70) on that problems. •

According to this definition, a strategy is more efficient the smaller its efficiency value is. It is also left open whether the domains underlying the problem set are identical or not; in our experiments, we consider intra-domain characteristics if not explicitly stated otherwise. Please note that it is unspecified in the above definition whether or not a solution to the given problem actually exists.

Definition 6.2 (Stability of System Configurations)**.** The *(search-) stability* of a system configuration $\mathfrak{C}$ with respect to a set of problems $\{\pi_1,\ldots,\pi_n\}$ is defined in terms of the statistical dispersion (sample variance) of the number of plans that are analyzed for flaw detection by that configuration in all terminating runs of the refinement planning algorithm on that problems. •

Given the previous definition, a configuration is stable if its stability value is very low (ideally 0). The computation of stability values thereby relies on the efficiency results.

Definition 6.3 (Reliability of System Configurations)**.** The *(search-) reliability* of a system configuration $\mathfrak{C}$ with respect to a set of problems $\{\pi_1,\ldots,\pi_n\}$ is defined according to the ratio of problems on which the refinement planning algorithm does not terminate with the given configuration to the overall number of problems n (failure ratio). It has to be considered that termination may depend on external factors like total run-time or an upper bound for the size of the search space. •

Reliability is thereby given as 0% for solving all problems in a given set, while 100% represents a complete loss of all runs. This means in particular, that the system did not terminate even with a *fail* result.

We are now ready to map the notions of the specific performance characteristics from the previous configuration-based definitions onto the involved strategy tuples. Since we are interested in analyzing the strategies' search space organization (and not in some solution quality), we restrict our evaluation criteria on those strategy tuples that deploy the identical solution selection function[1]. We may assume that it is a function "first" with

[1] As it has been stated before, this restriction is not an essential one. Adding the solution selection function merely produces an additional dimension in the experimental space.

$f^{solSel}_{\text{first}}(P_1 \ldots P_n) = \text{true}$ for any sequence of plans $P_1 \ldots P_n$, that means, it always accepts the first solution that is found (see Def. 2.39 on page 68).

Definition 6.4 (Performance Characteristics of Planning Strategies)**.** A combination of a modification selection function f_a^{modSel} and a plan selection function $f_b^{planSel}$ is more *problem-specific (search) efficient* on a given problem π than a combination $f_{a'}^{modSel}$ and $f_{b'}^{planSel}$ with respect to two given system configurations

$$\begin{aligned} \mathfrak{C}_1 &= \langle \mathfrak{Det}, \mathfrak{Mod}, \mathfrak{Inf}, (f_a^{modSel}, f_b^{planSel}, f^{solSel}) \rangle \text{ and} \\ \mathfrak{C}_2 &= \langle \mathfrak{Det}, \mathfrak{Mod}, \mathfrak{Inf}, (f_{a'}^{modSel}, f_{b'}^{planSel}, f^{solSel}) \rangle, \end{aligned}$$

if configuration $\mathfrak{C}_1$ is more (search) efficient on the problem set $\{\pi\}$ than $\mathfrak{C}_2$.

The pair of strategy functions a and b can be called more *domain-specific (search) efficient* than a' and b' if for a given set of problems $\{\pi_1, \ldots, \pi_n\}$ over a domain model D the selection functions f_a^{modSel} and $f_b^{planSel}$ are more problem-specific efficient than $f_{a'}^{modSel}$ and $f_{b'}^{planSel}$. If a strategy combination is more domain-specific efficient than another over a given set of problems from a given set of domains, it becomes a more *configuration-specific (search) efficient* pair of strategy functions, and if is finally a *universally (search) efficient* combination, if it is more configuration-specific efficient in a given set of system configurations. The sets of problems, domains, and configurations to consider are also called the *scope* of the experiments.

Analogously, a combination of modification and plan selection functions is regarded more *stable* and *reliable* than another combination on the problem, domain, and configuration level. •

The introduced performance characteristics define a three-dimensional space of strategy quality for which we now have to decide what it means for a strategy to be "better" than another. We will analyze the strategy performance from different points of view: The first kind of evaluation is primarily interested in strategies that are producing a result, and among their top performers, we identify the most efficient ones. A slightly different value system may prefer consistently efficient strategies.

Definition 6.5 (Performance Aggregation)**.** An *efficiency-centered performance evaluation* of a set of planning strategies is a dominance analysis according to (1) efficiency, (2) stability, and (3) reliability.

A *reliability-centered performance evaluation* is defined over (1) reliability, (2) efficiency, and (3) stability.

The success criterion for a strategy is in both evaluation types to be undominated in the according schema. •

The definition implies for the efficiency-centered schema, that a strategy is considered successful

1. if it is a member or the set of all strategies that are undominated with respect to efficiency (we may call this set A),
2. if it is a member of the set of strategies that are undominated by a member of A with respect to stability (this set may be called B), and
3. if it is a member of the set of strategies that are undominated by a member of B with respect to reliability.

Success in the reliability-centered evaluation schema is interpreted analogously.

Please note the subtlety of defining performance not in terms of being better/faster than any competitor (cf. discussion section below) but in terms of none of the competitors being systematically better/faster. While the former puts an emphasis on finding the best – already existing – in the field, the latter adopts our research perspective of identifying any subject of which we may assume that it is possible to be developed into one of the best in the field. Such a conservative definition takes into account that as long as there is no generally better designed strategy, the mechanism has to be considered as a worthwhile candidate for improvement (under the optimistic assumption that improvement is possible in general). Furthermore, our view of performance ultimately coincides with the competition-like view, namely when only one candidate remains, which implies that it dominates all remaining candidates.

It has to be stressed that the performance characteristics are defined in terms of a certain scope, that means, in terms of *actual experimental evidence* only. We will have to address the issue of how some of our results can be generalized to a wider class of problems, domains, and finally system configurations, but the empirical approach is not able to produce some kind of general proof. For example, a strategy combination that is found to be more domain-specific efficient than some other combination does not necessarily *dominate* that strategy in the broader Artificial Intelligence sense unless we can *prove* that one strategy is more problem-specific efficient on *any* problem in that domain.

While it would be most interesting to be able to create universally efficient, stable, and reliable strategies within the widest possible scope, their existence is, however, very unlikely and the ambition to build them is therefore of limited significance. For example, we do not expect any strategy combination to be universally efficient on all system configurations that we presented in this thesis, because that would imply to find a perfect search controller for any kind of planning paradigm and domain structure. Building universally stable strategies is possible in a fixed "first-solution" selection setting but obviously impossible in an open environment in which other plan quality metrics become relevant. This is because we want to build non-trivial stable heuristics,[2] which would be forced to produce a (non-optimal) result of constant quality in an any-time fashion.

Our experimentation is targeted at identifying those components from our portfolio, which constitute domain-specific efficient, stable, and reliable strategies for hybrid planning configurations. We will focus on the role of causality with respect to task expansion and therefore *disable task insertion* (cf. Sec. 3.3.1), which helps us to relate our findings to the (few) results that are available in the literature. Our domain models of Sec. 5.2 will thereby serve as reference applications for efficiency- and reliability-centered performance evaluations.

Concerning the participating strategy-component candidates, an exhaustive, pairwise experimental evaluation of all strategy combinations is obviously not feasible. With the hybrid planning flaw and modification classes, Chapter 4 provides 40 modification selection functions plus another 40 obtained by applying the "inversion" strategy. There are 70 plan selection functions defined, 20 of which are applicable to "clustering", and, again, all of those can be inverted. That means, there is a total candidate set of $(40+40)\times(70+20+70+20) = 14.400$ possible strategy combinations that are based on unary components alone, and for binary modification and plan selection compositions, we would have to evaluate $(80\times 79)\times(180\times 179) = 203.630.400$ strategies (not to think of selection triples, which would result in about $2,8\cdot 10^{12}$ combinations). Although these numbers contain a considerable amount of (sometimes trivially) useless combinations, the evaluation efforts are too high for an naive approach (see also future work section 6.6.4). We therefore make a start with a subjective selection of strategy combinations for being evaluated and try to find answers to a set of "leading questions" that will give us some insights into the inter-strategy mechanisms, synergy effects, and incompatibilities that may arise – and hopefully in the future some guidance through the vast space of strategy candidates.

Definition 6.6 (Leading Questions for the Experimental Evaluation)**.** The empirical data is interpreted according to the following set of research questions, which are formulated as an open issue on the domain level as well as the global perspective:

1. Which strategies dominate others? Is there a dominance pattern, for example a particular plan selection component? How does the individual performance change over the application domains?
2. Is there a confirmation, respectively difference between the theoretical expectations and empirical results with respect to antagonistic and synergistic strategy components? This includes in particular the widely discussed plan selection strategies that prefer most, respectively least constrained solutions (see the "ConstrPlans" plan selection function and its inversion, p. 159).
3. Regarding the role of strategy components:
   - Do prefix-sharing strategies produce similar results, and if not, can this be explained by subsequent components?
   - What is the exact performance relationship between strategies and their specializations? Examples are direct uniform and indirect uniform HotSpot components and the CL+OCA and PSA+OCA plan selections.

[2] A strategy is trivially universally stable if it prevents the algorithm from terminating.

- How do strategies with switched components perform, for example, if the primary plan selection is the secondary of the other and vice versa?

Are there completely different strategy combinations with identical results?

4. Related to the complete set of strategies tested: How do the four classical strategies
   - S+OC/CL+OC (A^* in terms of plan steps introduced and open preconditions to be solved, p. 162),
   - *UMCP* (conflict resolution on the primitive plan level, p. 174),
   - *SHOP* (expansion in execution order, p. 176), and
   - *EMS* (*expand-then-make-sound*, p. 175)

   perform? Regarding hybrid planning system configurations, it becomes in particular relevant whether or not an early conflict resolution pays off, that is to say, do *SHOP* and *EMS* dominate the *UMCP* strategy?

   In this context, it may also be interesting to ask which strategies are dominated by strategies deploying a depth-first plan selection? How do those strategies perform that share their modification selection with the "depth-first" ones?
5. Last, but not least, according to the empirical results, which problems are harder than others in each domain? Is the result plausible (expectable, if not trivial)? What can be deduced from this knowledge concerning the domain models?

•

Having defined the goals of the empirical strategy evaluation, we are now ready to specify the experimental frame.

6.2 Experimental Setup and Design

The following sections describe the design of the performance experiments: the strategy combinations that are evaluated, the application domains and problem instances that are used as a referential frame for the performance analysis, and the concrete experiment procedure.

6.2.1 Strategy Combinations

Our starting point for recruiting evaluation candidates lies in a small series of experiments that have been conducted in previous work on hybrid planning strategies, also in the context of hybrid system configurations that *do not insert new tasks* [231,234]. We extend the respective component collection such that it now includes most of the flexible HotSpot and HotZone strategies; the complete list of participating modification and plan selection functions is given in Table 6.1. From these components, we build systematically 93 strategy combinations (the table indicates which components were used in a primary and secondary position). The systematic approach for combination building allows us to deduce properties of shared or altered sub-components, for example, strategies with the same modification selection functions but different plan selections and vice versa.

This candidate set includes the three classical hybrid planning strategies *UMCP*, *SHOP*, and *EMS*, together with enhancements for two of them: we provided *UMCP* and *SHOP* with potentially synergetic plan selection functions. The evaluation set also covers a number of hybrid CL+OC and S+OC derivates, which have been used earlier in the context of non-hierarchical partial-order planning (cf. Def. (4.15) and (4.16)).

Primary Modification Selections	
EMS	expand then make sound (Def. (4.40))
HZone	HotZone (Def. (4.30))
LCF	least committing first (Def. (4.22))
Secondary Modification Selections	
LCF	
IndUniHS	indirect uniform HotSpot (Def. (4.24))
DirAdaptHS	direct adaptive HotSpot (Def. (4.27))
DirUniHS	
EMS	
HZone	
IndAdaptHS	indirect adaptive HotSpot (Def. (4.28))
ModBasedHS	modification based HotSpot (Def. (4.29))
Pref-$\mathbb{M}_{\texttt{ExpandTask}}$	prefer expansion modifications (Def. (4.4))

Primary Plan Selections	
CL+OCA	A^* heuristic (Def. (4.15))
DirUniHS	direct uniform HotSpot (Def. (4.25))
FewerHZones	fewer HotZones (Def. (4.32))
FewerModBHS	Modification based HotSpot (p. 169)
IndUniHS	indirect uniform HotSpot (Def. (4.26))
ConstrPlans	more constrained plans (Def. (4.3))
$ConstrPlans^{-1}$	inversion of ConstrPlans (Def. (4.34))
LeastHZone	least hottest zone (Def. (4.31))
PSA+OCA	expansion aware A^* heuristic (Def. (4.16))
$\mathbb{F}ITE$	smaller detection ratio (Def. (4.19))
Secondary Plan Selections	
Fewer-$\mathbb{M}$	fewer modifications first (Def. (4.18))
ConstrPlans	
$ConstrPlans^{-1}$	
FewerHZones	

Classical Strategy Compositions	
EMS	$f^{modSel}_{Addr-\mathbb{F}_{\texttt{Threat}}} \triangleright f^{modSel}_{Addr-\mathbb{F}_{\texttt{OpenPrec}}} \triangleright f^{modSel}_{Pref-\mathbb{M}_{\texttt{ExpandTask}}}$ with $f^{planSel}_{First}$
SHOP	$f^{modSel}_{\prec} \triangleright f^{modSel}_{Pref-\mathbb{M}_{\texttt{ExpandTask}}{}^{-1}}$ with $f^{planSel}_{First}$ as described in Def. (4.41) plus the variant $f^{planSel}_{\text{PSA+OCA}}$
UMCP	$f^{modSel}_{Pref-\mathbb{M}_{\texttt{ExpandTask}}} \triangleright f^{modSel}_{LCF}$ with $f^{planSel}_{First}$ and $f^{planSel}_{Addr-\mathbb{F}_{\texttt{AbstrTask}}{}^{-1}}$ as described in Defs. (4.38) and (4.37)

Table 6.1: All plan and modification selection functions that are participating as strategy components in the empirical evaluation.

6.2.2 Domains and Problems

The evaluation environment for our hybrid system configurations are the three application domain models that have been presented and discussed in Sec. 5.2: *Satellite*, *UM Translog*, and *CrissCross*. Since they represent three different styles of hybrid domain model building and include two former benchmarks[3], we believe that they are well-suited for our purpose.

The following sections introduce the planning problems that we defined for the performance experiments, grouped by the respective domain models. For each problem, we briefly discuss its structure and solution properties and indicate the best known solution as well as the best solution found by the system. The "best known solution" refers to the results obtained by a manual or in some cases semi-automatic planning tool and the minimum plan step number as well as the visited search space are upper bounds for the exact minima.[4] Its strategy cuts more or less all unselected plans and is therefore an incomplete process (cf. discussion in Sec. 2.8.5). The visited plan space is however considerably smaller since no alternatives are evaluated up to the point where the system "completes" the human plan generation suggestion and therefore roughly corresponds to the length of the solution plan path in the search space.

Problems in the *Satellite* Domain The *Satellite* domain is originated in non-hierarchical partial-order planning and therefore even in its hybrid formulation not very deeply structured (Sec. 5.2.1). In contrast to the known IPC problems, the hybrid problems are defined in terms of observation *tasks* that have to be *performed* and not in terms of observation *goals* to be *achieved*. The problems we deal with in the experiments are variations over the number of observation tasks, available satellites, and image properties. The basic choice in plan development is the decision of whether to perform the observations in a sequence on one satellite or to distribute the tasks on different satellites. All problems are solvable, that means, every required image mode is provided by at least one instrument of at least one satellite and every instrument has defined a calibration target. It has to be noted, that although the problem instances appear very small, the induced search space is surprisingly large. In fact, the satellite observation planing problems contain the most difficult of the whole evaluation.

1obs-1sat-1mod: One observation task, one satellite available, and the image is required in one mode. There is one unique solution in which the satellite is first slewed into a unique calibration direction, then calibrated, and finally slewed into the target direction in which the image is taken. The PANDA-System finds a solution with 7 primitive tasks (including initial and goal tasks) within 14 cycles in manual mode, the best automatic solution is found in 23.

2obs-1sat-1mod: Two observation tasks, one satellite available, both images are required in the same mode. There exist symmetric solutions in which a 1-1-1 solution is found for one target, followed by a slewing task into the other target direction. The second image can be taken without additional calibration. We can find a solution with 9 primitive tasks (including initial and goal tasks) after 21 cycles in manual mode; automated configurations needed at least 163.

2obs-2sat-1mod: 2 observation tasks are to be performed with 2 available satellites, images are required in one mode. There are two classes of solutions: Either a 2-1-1 solution on either satellite or two symmetric 1-1-1 solutions for each satellite. We can find manually one of the symmetric solutions with 12 primitive tasks (including initial and goal tasks) within 24 cycles and a solution with 9 primitive tasks in which two images are taken in series by the same satellite within 19 cycles. The best solution found by an automated configuration took 109 cycles.

2obs-2sat-2mod: 2 observation tasks, 2 satellites available, images are required in 2 modes. First type of solution is analogous to the 2-1-1 problem: one satellite has to sequence its observations, including necessary calibration procedures. Another option is a solution that consists of two 1-1-1 sub-solutions for each satellite, that means, images are taken in parallel. There is no further symmetry, because only one of the satellite carries the instruments that comply with the mode requirements. We can manually find a solution that consists of 13 primitive tasks after 28 cycles. This solution is obtained when one

[3] For some discussion about the domain models deployed in the *International Planning Competition* see page 191.

[4] The minimal sizes of the solution plans' task expression sets are known, however, the portion of the search space that is at least required to be traversed for encountering a solution has not been determined precisely. We believe that the numbers we obtained from an experienced user in a completely manual plan generation are a very close upper bound, though.

satellite takes two images in series. A parallel solution with 12 primitive tasks can be found after 25 cycles. The best automated search takes 135 cycles.

3obs-1sat-1mod: 3 observation tasks, one satellite available, images are required in one mode. Analogously to the 2-1-1 and 1-1-1 problems, there exist symmetric solutions with respect to the actual observation ordering. We can find a solution with 11 primitive tasks after 33 cycles in the manual search mode, the best known automated plan generation took 1.516.[5]

3obs-2sat-3mod: 3 observation tasks, 2 satellites available, images are required in 3 modes. The instruments on one satellite are capable of all relevant modes, the other satellite only covers two modes (similar to 2-2-2). We can distinguish three classes of solutions: (1) one satellite takes all the images, (2) one satellite takes two images, the second satellite performs one observation, and (3) the second satellite performs two observations. In addition, symmetries exist with respect to the observation ordering. A completely sequential solution on the primary satellite contains 19 primitive tasks (including initial and goal tasks) and can be found manually within 46 cycles. A solution in which both satellites are involved has 18 primitive tasks and takes 42 cycles. No automated configuration was able to solve the problem within the given time and search space horizon.

3obs-3sat-1mod: 3 observation tasks, 3 satellites available, images are required in one mode. The solutions can be characterized analogously to the 2-2-1 problem: Either every satellite performs exactly one observation task or one is taking over responsibility for additional observations. Symmetries occur concerning the choice of satellites. We can find a solution with 17 primitive tasks within 36 cycles in manual mode. For this solution, the three images are taken in parallel (one image per satellite). More "sequential solutions" include only 14 and 11 primitive tasks and are found manually after 31 cycles and 26, respectively. The best known automated strategy's performance is 336.

3obs-3sat-3mod: 3 observation tasks, 3 satellites available, images are required in 3 modes. This problem is basically an extension of the 3-2-3 scenario, in which satellites are equipped with an increasing number of instruments: one satellite is only capable of one mode, the second can deliver images in two modes, and the third finally covers all three required modes. The classes of solutions are analogous to the previous two-satellite problem and are consequently symmetric with respect to the ordering of observations in parallel threads. Manual efforts for solving this problem are 44 (19 tasks, images taken in series), 43 (18 tasks, two images in parallel), and 39 (17 tasks, all images in parallel) cycles. There is no result from the automated strategy candidates.

Problems in the *UM Translog* Domain The *UM Translog* domain originally is an HTN domain that has been transcribed into a non-hierarchical IPC benchmark (Sec. 5.2.2). It has deeply nested method structures (see also discussion in the respective section) and therefore serves as a test for the prediction of causal interactions down the decomposition hierarchy. Our problem set consists of a representative cross-section of the transportation means, ranging from air freight to specific chemical transports. The focus is thereby on identifying the suitable method(s) for the decomposition of transportation tasks, that means, on the choice of compatible transportation means and procedures, and not on the embedding path-finding subproblem that the urban infrastructure poses. All problem instances are solvable by all of the appropriate task implementations; there are no further constraints like limited transportation means, unavailability of transportation routes, and the like. The automated strategies perform very well on these benchmarks, so since the construction of a solution under the experiment-constraints (time and search space) seems to be a feasible task for our evaluation candidates, we interpret it as an performance indicator of much finer granularity than, for example, the satellite problems.

Since none of the following problems has multiple solution classes like the previous satellite problems, we will use a short notation for the minimal plan length (l_{min}, including initial and goal task), the number of plans that are visited by using a human-guided manual strategy (c_{man}), and the minimal number of cycles it takes for our set of automated evaluation candidates (c_{auto}).

[5] It is the first problem in this listing in which the automated process performs significantly worse than a human search (factor is 46). Significance has to be seen here in relation to an exponential growth of the search tree, which makes an additional search "overhead" of factor 10 not too surprising.

Airplane: A plan for transporting a regular package as air freight eventually consists of primitive tasks for loading the package onto the plane via a conveyor ramp, moving the plane to the destination airport, and unloading the plane. $l_{min}: 15 \quad c_{man}: 34 \quad c_{auto}: 87$

RegularTruck: In this basic package delivery task a truck is used to transport a regular, stackable good from one city to another. $l_{min}: 11 \quad c_{man}: 24 \quad c_{auto}: 60$

RegularTruck-3Locations: This is a variation of the regular truck problem above in which the decomposition methods are used to perform an additional route planning over an intermediate city. $l_{min}: 12 \quad c_{man}: 28 \quad c_{auto}: 82$

2-RegularTruck: In another variation of the regular truck problem we ask the planner to deliver two regular packages. With 242 cycles for the "fastest" automated plan generation, this problem exhibits the highest difference between human and automated strategy performance in this domain. $l_{min}: 20 \quad c_{man}: 48 \quad c_{auto}: 242$

TankerTruck: The tanker truck problem models a transport of liquid hazardous chemicals. Before the actual transportation can begin, a permission for the transportation route has to be obtained, the cities that are passed on the way have to confirm that they allow the truck to enter their districts with such goods, and specific warning signs have to be affixed to the truck and trailer. In addition, the tank filling and emptying procedures have to be carried out under security measures. $l_{min}: 19 \quad c_{man}: 48 \quad c_{auto}: 65$

HopperTruck: Substances like sand have to be transported in containers for bulk material. Such a transporter is a hopper, which is loaded and unloaded via a chute that has to be connected to the vehicle during the filling operations. $l_{min}: 11 \quad c_{man}: 24 \quad c_{auto}: 60$

FlatbedTruck: This is an example for transporting lumber; this involves a stationary equipment "crane" for loading a flatbed truck. $l_{min}: 9 \quad c_{man}: 32 \quad c_{auto}: 67$

AutoTruck: Delivering cars on a special truck does only require a specific trailer but no additional security measures. $l_{min}: 11 \quad c_{man}: 30 \quad c_{auto}: 88$

ArmoredRegularTruck: In this scenario, a valuable (regular) art object is transported, which requires an armored transportation vehicle as well as guards that keep the loading procedure under surveillance. $l_{min}: 16 \quad c_{man}: 26 \quad c_{auto}: 63$

MailTraincar: Mail is an instance of regular packages, in this case delivered by train. Choosing train transport implies providing a suitable car and a locomotive to which the car can be connected. $l_{min}: 13 \quad c_{man}: 32 \quad c_{auto}: 78$

RefrigeratedTraincar: The problem deals with a train transport of food, which is a regular good that has to be kept cool. $l_{min}: 13 \quad c_{man}: 32 \quad c_{auto}: 78$

AutoTraincar: This is a variant of the auto truck problem in which a train transport is used. The locomotive has to be moved to the customer site first. $l_{min}: 14 \quad c_{man}: 43 \quad c_{auto}: 152$

AutoTraincar-bis: This problem is an simplification of the auto train car scenario: the locomotive is already at the customer site's train station. $l_{min}: 13 \quad c_{man}: 38 \quad c_{auto}: 100$

Problems in the *CrissCross* Domain The *CrissCross* domain has been designed as a test environment for the HotSpot planning strategies (Sec. 5.2.3). It's decomposition hierarchy is relatively flat, however it introduces many non-trivial causal inter-dependencies across the expansions that may provoke unresolvable inconsistencies. The benchmark problems consist of multiple arrangements of the basic causality patterns as shown in Fig. 5.16 and 5.17 (p. 210 and 211). We thereby distinguish repetitive arrangement of the same pattern and mixtures of different patterns, with which we try to analyze scalability in terms of "size" (adding another block to the table) versus scalability in terms of "structural complexity". Again, all problem instances are solvable since there are sufficient character constants available to satisfy a task implementation. Since there are no substantial variations in the solution structure, the following descriptions use the previously introduced short notation for the solution characteristics.

P0: This problem is defined as an initial task network that corresponds to the situation shown in the first row of Fig. 5.16 (p. 210). It is a prototypical arrangement of tasks in which a condition "passes" an intermediate step. This problem can be solved without backtracking. $l_{min}: 5 \quad c_{man}: 7 \quad c_{auto}: 15$

P0-2: The problem combines two P0 instances, which may be placed in parallel. The causal chains within the P0 solutions may interfere, hence some search.

$l_{min}: 8 \quad c_{man}: 15 \quad c_{auto}: 36$

P0-3: Like in the previous problem, this one combines three P0 instances, which may pose threats to the causal sub-structures in the P0 chains.

$l_{min}: 11 \quad c_{man}: 24 \quad c_{auto}: 95$

P1: This problem corresponds to the situation shown in the middle row of Fig. 5.16. P1 extends P0 by an additional consumer of the state feature that is finally required by the task at the end of the P0 sub-chain. At the end of a P0 plan, that state feature is however negated and the planner has to ensure that the additional step is positioned before that threat. In this simplest possible scenario, there may be some backtracking over unsuccessful threat resolution tactics. $l_{min}: 6 \quad c_{man}: 10 \quad c_{auto}: 22$

P1-2: The problem combines two P1 instances, which may be placed in parallel. This arrangement combines threat resolution combinatorics in the style of the respective P0 instances and the P1 problem. $l_{min}: 10 \quad c_{man}: 21 \quad c_{auto}: 57$

P1-3: Like in the previous problem, this one combines three P1 instances which obfuscates causal interference detection by the number of combinations of hypothetical threats.

$l_{min}: 14 \quad c_{man}: 33 \quad c_{auto}: 139$

P2: This problem corresponds to the situation shown in the last row of Fig. 5.16. It is basically an integration of two mirrored P1 instances in which the order of expansion refinements determines the structure of the (visible) causal interactions. In the single P2 problem, this already produces search paths that cannot be trivially prioritized according to features like "integrity" or "stage of development" of the causal structure, causing branches in the search tree that are usually pursued in parallel.

$l_{min}: 7 \quad c_{man}: 13 \quad c_{auto}: 40$

P2-2: The problem combines two P2 instances, which may be placed in parallel. This is the first *CrissCross* problem that may induce a serious explosion of the search space.

$l_{min}: 12 \quad c_{man}: 28 \quad c_{auto}: 147$

P2-3: Like in the previous problem, this one combines three P2 instances. Regarding the minimal search-space exploration, this has been the most difficult problem (more than two times harder than the second hardest) in the *CrissCross* domain for the automated strategies.

$l_{min}: 17 \quad c_{man}: 45 \quad c_{auto}: 358$

P0-P1: This problem combines a P0 and a P1 problem instance, both sub-solutions can be sequenced or placed in parallel. $l_{min}: 9 \quad c_{man}: 18 \quad c_{auto}: 44$

P0-P2: This problem combines a P0 and a P2 problem instance, both sub-solutions can be sequenced or placed in parallel. $l_{min}: 10 \quad c_{man}: 22 \quad c_{auto}: 94$

P1-P2: This problem combines a P1 and a P2 problem instance, both sub-solutions can be sequenced or placed in parallel. $l_{min}: 11 \quad c_{man}: 25 \quad c_{auto}: 105$

P0-P1-P2: This problem combines a P0, a P1, and a P2 problem instance, the sub-solutions can be sequenced or placed in parallel. $l_{min}: 14 \quad c_{man}: 35 \quad c_{auto}: 152$

6.2.3 Experimental Frame

The above introduced problem definitions served as the experimental environment of our evaluation. We implemented an experiment software tool for automatically configuring and running the planning system as well as for retrieving and storing the produced data. With that tool, we set up a hybrid planning system configuration for every combination of the strategies candidates and problem instances and run 5 plan generation trials, resulting in $93 \times (8+13+13) \times 5 = 15.810$ data points where we determined the size of the explored search space in terms of the number of plans that have been analyzed for flaw computation. For each set of five runs on the same configuration and problem, we calculated the minimum value, the arithmetic mean, the sample variance, and the failure ratio, that means, the ratio of runs in that set that did not terminate on a solution. Every run of the planning system was limited to a real-time consumption of 150 minutes and an exploration of at most 5.000 plans; a run consequently failed if it exceeded these limits

and was counted as a non-terminating run and excluded from the mean and variance computation. We preferred the exclusion of failure runs over assigning a fictitious "worst-case" in order to capture the notions of efficiency and reliability separately.

All participating strategies were deployed in an aggregating "master strategy" that implements a cut operation with respect to what we call *unreachable modifications* (see p. 172). Furthermore, a decision procedure is prefixed to every to call of the modification selection functions: in the fashion of the least commitment selection (Def. (4.22)) the current plan is analyzed for singular modification proposals, that means, for flaws that are answered by exactly one modification. In this case, the proposed modification is selected and executed without entering the "regular" modification selection cycle and the resulting plan substitutes its predecessor in the fringe. In this way, necessary modifications are forced and some redundant plan space fragments are cut. But it has to be emphasized that despite these two heuristic operations, all evaluated configuration instances are complete planning systems in the sense that is discussed in Sec. 2.8.5.

6.3 Evaluation Results in the *Satellite* Domain

Efficiency-Centered Performance Evaluation

The *Satellite* domain and the proposed problem instances are very accessible; we will hence report our findings in this particular domain along with a detailed explanation of our applied analysis techniques and refer to these explanation in later sections. We begin with the efficiency-centered performance evaluation in the *Satellite* domain, the results of which are depicted in Figure 6.1.

The graph shows the aggregated dominance relationships between the strategies regarding their domain-specific search efficiency, stability, and reliability. The nodes are only a fraction of the participating candidates, they are those strategy tuples that are undominated in the domain-specific search-efficiency analysis of the domain. The node at the top of the figure, for example, stands for the pair of modification selection function $f^{modSel}_{EMS} \triangleright f^{modSel}_{LCF}$ and plan selection function $f^{planSel}_{\text{PSA+OCA}}$. The three numbers denote the number of strategies that are dominated by the strategy of this node in terms of efficiency (left), stability (middle), and reliability (right).

According to the efficiency-centered performance evaluation schema, green edges are added to the graph for every stability-dominance relationship between two strategies of the candidate set. After that, for any strategy that performs more domain-reliable than another, a corresponding blue edge is inserted if this does not induce a cycle on the edges, in other words, if the reliability result does not contradict stability (which in fact only occurs once in this candidate set and domain). In order to improve readability, the depicted graph is a reduced representation such that all transitive edges are removed, and all undominated nodes are coloured red.

The most prominent result of the first efficiency-centered performance evaluation schema is therefore that the "best" strategies in the *Satellite* domain are the following:

- $f^{modSel}_{EMS} \triangleright f^{modSel}_{LCF}$ with $f^{planSel}_{\text{PSA+OCA}}$
- $f^{modSel}_{EMS} \triangleright f^{modSel}_{LCF}$ with $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$
- $f^{modSel}_{LCF} \triangleright f^{modSel}_{DirAdaptHS}$ with $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$
- f^{modSel}_{SHOP} with $f^{planSel}_{First}$ (also known as the "classical" SHOP strategy[6])

While we expected the candidate set to include strategy combinations like $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ with $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$, which is a strategy that dominates more than 50% of its competitors in every performance characteristic, we were surprised by the presence of tuples like $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndUniHS}$ and $f^{planSel}_{DirUniHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$,

[6] f^{modSel}_{SHOP} is an abbreviation for $f^{modSel}_{\prec} \triangleright f^{modSel}_{Pref-\mathbb{M}_{\texttt{ExpandTask}^{-1}}}$, see page 176.

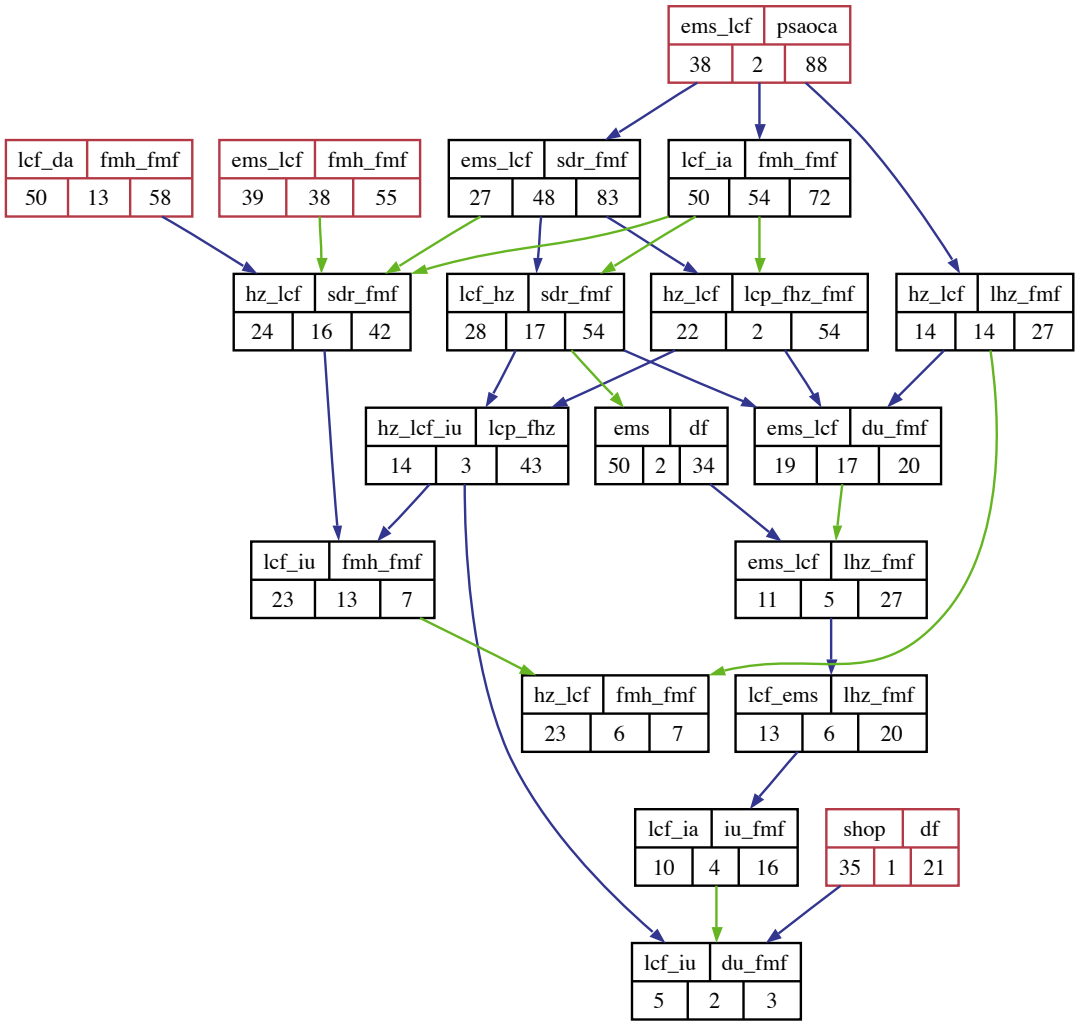


Figure 6.1: The efficiency-centered performance evaluation in the *Satellite* domain. Nodes are the most efficient strategies, edges represent stability (**green**) and reliability (**blue**) dominance. Node annotation shows number of strategy combinations dominated by the node in terms of efficiency, stability, and reliability (**red** = undominated strategy).

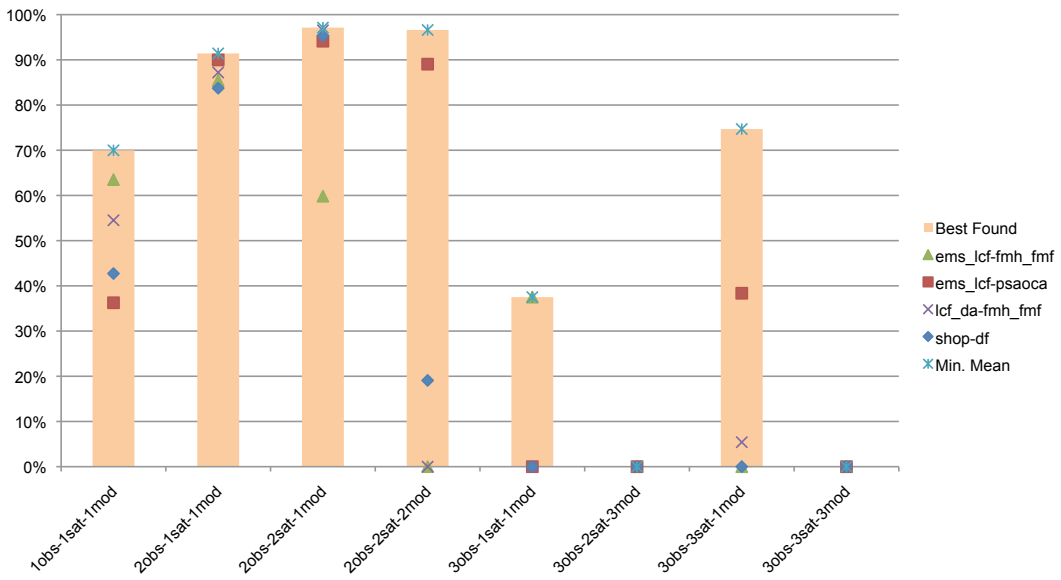


Figure 6.2: The *relative* search efficiency of the undominated strategies in the *Satellite* domain. 100% on the vertical axis corresponds to a "perfect" strategy that solves the problem immediately, 0% represents the worst efficiency value in the experiments.

because they hardly dominate any competitor. It is also remarkable that the classical strategies *EMS* and *SHOP* with their depth-first plan selection function are in the candidate set.

In order to get an idea of how "good" the "best" are, we put the efficiency-centered performance result into perspective of the range of efficiency values that we have measured for their competitors. Fig. 6.2 displays the efficiency values normalized with respect to the worst measured value. For example, on the 1-1-1 problem, the worst known efficiency is 87 cycles, which defines the 100% reference value for the performance window of that problem. The $(f_{EMS}^{modSel} \triangleright f_{LCF}^{modSel}, f_{FewerModBHS}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel})$ strategy ("ems_lcf-fmh_fmf") has an efficiency of 32 in that problem instance, which leads to a relative performance gain of 64%. What is intended to be seen in the figure is the relative distance between the individual winning strategies and the two best cases: the best efficiency on that problem ("Min. Mean") and the best known single run, represented by the orange "Best Found" columns; the latter also displays the range of the obtained efficiency results in relation to the worst case. It has to be noted that the two best cases do not coincide, they are merely very close on the large scales. While on some problems the average values of the best strategies are very close to the best known solutions, there is definitely room for improvement on others (not to speak of the unsolved instances).

That leaves us with the question of a dominance pattern: The performance results in Fig. 6.1 provide us with a set of good candidates for working in the *Satellite* domain, they do however not give a broader view on what *kind of* strategy one should try in that domain. To that end, we set up a matrix of strategy components that covers most of the combinations (80, plus 5 classical variations) and calculate the average rank of every combination within the efficiency values for each problem instance (with respect to the whole strategy universe of 93 combinations). Tab. 6.2 shows that matrix with the respective selection functions in alphabetical order. Since the bare numbers are certainly hard to overview, we provide a colouring schema that roughly displays the statistical rank of the strategy's not rounded average rank within the original data (cf. Tab. B.1).

- The **blue** cells are values that are within the first quartile of the rank values,
- **beige** coloured cells lie between the first quartile and the median,
- **white** indicates a value between median and third quartile, and
- the **red** color denotes those values that are above the third quartile, respectively the fact that the problem instance could not be solved by the strategy.

The rationale for looking after the rank of ranks lies in normalizing the performance characteristic over the statistical population and the differently difficult problem instances.[7] To put it simply: We can characterize how difficult is it to win with the given challenges and field of competitors. The ranks for completely

	ModSelection									
PlanSelection	**ems_lcf**	**hz_lcf**	**lcf_da**	**lcf_du**	**lcf_ems**	**lcf_hz**	**lcf_ia**	**lcf_iu**	**lcf_mod**	**lcf_pExp**
cloca	66	61	61	63	64	61	59	60	65	64
du_fmf	33	35	44	50	51	47	43	45	50	50
fhz_fmf	38	53	45	53	55	56	38	56	62	56
fmh_fmf	27	42	31	56	55	46	24	43	58	43
iu_fmf	37	42	45	54	53	41	37	56	46	55
lhz_fmf	37	36	47	49	36	47	46	46	42	39
psaoca	27	46	34	55	53	48	39	43	59	52
sdr_fmf	29	41	38	50	46	36	35	37	49	50

EMS	25		
UMCP	41	**UMCP+**	45
SHOP	23	**SHOP+**	49

Table 6.2: A strategy-component matrix of the average ranks of efficiency values in the *Satellite* domain (**blue** = within first quartile, **beige** = within median, **red** = beyond third quartile).

unsolved problems are set to the arithmetic mean of the first and last potential rank that could be assigned to that strategy, that means, for n competing strategies of which m did not solve a given problem, all failing strategies are assigned a rank of $(n-m)+\frac{m}{2}$. Since we cannot interpolate the efficiency beyond the termination criterion, we decided to give an "average bad rank" to the failing strategies. The relatively high number of unsolved problem instances is also a reason for the bad rank values; this also applies to the well-performing combinations. After all, an empirical evaluation can only be done relatively to a given universe.

There is however an important observation that can be made on the data in Tab. 6.2. It appears that efficiency is positively associated with the plan selection functions $f^{planSel}_{\mathbb{F}/TE} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ (row "sdr_fmf") and $f^{planSel}_{LeastHZone} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ and the modification selection functions $f^{modSel}_{EMS} \triangleright f^{modSel}_{LCF}$ and $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$. On the other hand, a negative influence on efficiency seems to emerge from plan selection $f^{planSel}_{\text{CL+OCA}}$ and modification selection $f^{modSel}_{LCF} \triangleright f^{modSel}_{ModBasedHS}$. Both tendencies are exceptionally evident at the respective crossing points in the matrix, so we may hypothesize that efficiency is an additive property of both selection function types, although it does not explain all phenomena (for instance, the fact of some depth-first combinations being undominated). We will come back on this issue later.

Reliability-Centered Performance Evaluation

We are now switching to the reliability-centered performance evaluation in the *Satellite* domain, and we can confirm our previous findings: Figure 6.3 shows the results that are obtained if reliability becomes the primary objective for a strategy. The candidate set is considerably smaller, since there are only four strategy tuples that are undominated according to the reliability analysis. We conjecture that this is also due to the small number of possible outcomes for the failure ratio calculation (five discrete values, cf. Tab. B.3), and that the set will therefore become richer populated in an experimental setup with substantially more runs per configuration and problem. The graph does not contain any edges, because no additional dominances are obtained based on efficiency and stability considerations. That means, all candidates qualify for the "best" strategy:

- $f^{modSel}_{EMS} \triangleright f^{modSel}_{LCF}$ with $f^{planSel}_{\text{PSA+OCA}}$
- $f^{modSel}_{EMS} \triangleright f^{modSel}_{LCF}$ with $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$
- $f^{modSel}_{LCF} \triangleright f^{modSel}_{DirAdaptHS}$ with $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$

[7] In fact, this presentation also normalizes the influence of the examined domain.

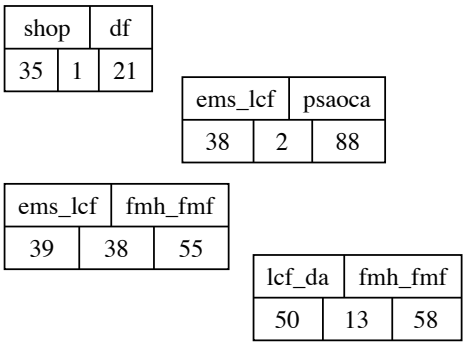


Figure 6.3: The reliability-centered performance evaluation in the *Satellite* domain. Nodes are the most reliable strategies; there exists no additional dominance result, hence all candidates are undominated according to this evaluation schema. Node annotation shows number of strategy combinations dominated by the node in terms of efficiency, stability, and reliability.

- f_{SHOP}^{modSel} with $f_{First}^{planSel}$

It has to be noted that the equivalence of the winner sets of the two performance evaluations is a pure coincidence, because the candidate sets from the first analysis steps (the undominated efficient versus undominated reliable strategies) are completely independently recruited and can be expected – with a certain probability – to be even disjunct.

What does it mean if a strategy is called *reliable* in the *Satellite* domain? The tabular overview in Tab. 6.3 puts the winning strategies into the perspective of the whole evaluation candidates.

Strategy	**Average Failure Rate**
Best value	43%
$f_{EMS}^{modSel} \triangleright f_{LCF}^{modSel}$ with $f_{\text{PSA+OCA}}^{planSel}$	43%
$f_{EMS}^{modSel} \triangleright f_{LCF}^{modSel}$ with $f_{FewerModBHS}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$	50%
$f_{LCF}^{modSel} \triangleright f_{DirAdaptHS}^{modSel}$ with $f_{FewerModBHS}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$	50%
1. Quartile	64%
f_{SHOP}^{modSel} with $f_{First}^{planSel}$	70%
Average value	72%
Median	75%
3. Quartile	80%
Worst value	88%

Table 6.3: Positioning of the most reliable strategies within the field of competitors in the *Satellite* domain.

We can see that the average reliability of our winning strategies is certainly not convincing on an absolute scale, one of them is even below the first quartile and only slightly above the average overall reliability. We believe that this is one more argument for an in-depth analysis of strategy behaviour, because a simplifying "adding things up" kind of data collection would have discarded some of our best strategies, by over-emphasizing single results and ignoring that actually no competitor is systematically superior.

Analysis of Strategy Components

We are closing the first performance analysis with a summarized comparison of the dominance results for strategy families that are built on the same primary plan selection function. It provides us with a brief overview of the experimental data as well as with a notion about the selectivity of the evaluation method

Primary plan selection	Total	Undom.	e	s	r	%	EC	RC
$f^{planSel}_{ConstrPlans^{-1}}$	2	2	1	–		50	–	–
$f^{planSel}_{First}$	3	2	1	1		44	1	1
$f^{planSel}_{FewerModBHS}$	10	5	3	2		33	2	2
$f^{planSel}_{DirUniHS}$	10	2	3	–		17	–	–
$f^{planSel}_{\mathbb{F}/TE}$	10	3	1	–		13	–	–
$f^{planSel}_{LeastHZone}$	10	3	1	–		13	–	–
$f^{planSel}_{IndUniHS}$	10	1	3	–		13	–	–
$f^{planSel}_{\text{PSA+OCA}}$	11	1	2	1		12	1	1
$f^{planSel}_{\text{CL+OCA}}$	10	–	–	–		0	–	–
$f^{planSel}_{FewerHZones}$	14	–	–	–		0	–	–
$f^{planSel}_{Addr-\mathbb{F}_{\texttt{AbstrTask}}{}^{-1}}$	1	–	–	–		0	–	–
$f^{planSel}_{ConstrPlans}$	2	–	–	–		0	–	–
Total:	**93**	**19**	**15**	**4**		**14**	**4**	**4**

Table 6.4: Primary plan selections in the *Satellite* domain: Number of undominated system configuration instances according to (e)fficiency, (s)tability, (r)eliability, efficiency-centered performance (EC), and reliability-centered performance (RC). % stands for the ratio of possibly to factually undominated.

in that domain (on that given set of problem instances). The following table (Tab. 6.4) shows the set of *primary* plan selections that participated in the evaluation, the total number of strategy combinations that have been built from that plan selection, and the number of combinations that were undominated with respect to efficiency (cf. Fig. 6.1), stability, and reliability (cf. Fig. 6.3). These figures are related to the total number of possible occurrences of instances being undominated, for example, two instances of the $ConstrPlans^{-1}$ family yielded three "undominated" findings of six possible, therefore resulting in 50%. Note that this ratio, if computed over all candidates, defines an average occurrence rate that we regard as a threshold for a strategy being "exceptionally undominated"; the table represents this threshold by a horizontal separator. The last two columns give the number of "best" strategies according to efficiency- and reliability-centered performance, respectively.

Although our findings have yet to be considered rather preliminary, these numbers give us the confidence that performance evaluations that are based on dominance considerations are strong enough to concentrate further experimentation on a "reasonably" shaped sub-set of candidates. "Reasonably" has thereby to be read in two ways: it reduces the experimentation efforts to dealing with a reasonable amount of subjects and at the same time, the candidate selection has been done in a comprehensible and plausible way (cf. remarks on Def. 6.5).

In order to gain further insight into the influence of strategy components on their performance characteristics, we build a component-based matrix of average reliability ranks, analogously to the efficiency ranks above. As we can see in Tab. 6.5, there also seems to be a correlation of selection functions and (un-) reliability, however the role of some individuals changed. The positive association with the plan selection function $f^{planSel}_{\mathbb{F}/TE} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ (row "sdr_fmf") and the modification selection function $f^{modSel}_{EMS} \triangleright f^{modSel}_{LCF}$ can be confirmed; the same holds for the negative performance of all combinations that involve a $f^{planSel}_{\text{CL+OCA}}$ plan and $f^{modSel}_{LCF} \triangleright f^{modSel}_{ModBasedHS}$ modification selection. New positive candidates are the $f^{modSel}_{HZone} \triangleright f^{modSel}_{LCF}$ family (four of them proved to be very efficient, cf. Fig. 6.1) and combinations with $f^{planSel}_{\text{PSA+OCA}}$. The cumulative effect of the components in the respective combinations seems to be evident, however not as definite as for the efficiency characteristic.

Let us finally have a look at a third component matrix, this time showing the stability values of the respective configurations' results. This characteristic that has not yet been examined by our previous analyses; we can see that Tab. 6.6 roughly corresponds to the above efficiency matrix, although there are not too distinctive row or column patterns visible. We would have expected to see stronger links also with the reliability results, since an instable strategy is intuitively more likely to fail and vice versa.

	ModSelection									
PlanSelection	**ems_lcf**	**hz_lcf**	**lcf_da**	**lcf_du**	**lcf_ems**	**lcf_hz**	**lcf_ia**	**lcf_iu**	**lcf_mod**	**lcf_pExp**
cloca	51	41	41	41	48	48	41	41	48	41
du_fmf	39	37	47	46	45	47	45	46	47	47
fhz_fmf	26	35	32	37	36	41	26	41	41	34
fmh_fmf	26	41	26	35	41	41	21	41	41	31
iu_fmf	39	38	47	46	46	38	40	51	41	47
lhz_fmf	34	29	45	46	39	46	43	45	40	40
psaoca	18	37	28	41	32	37	32	32	41	34
sdr_fmf	24	34	32	34	34	30	32	27	36	32

EMS	31		
UMCP	35	**UMCP+**	32
SHOP	32	**SHOP+**	41

Table 6.5: A strategy-component matrix of the ranks of reliability values in the *Satellite* domain (**blue** = within first quartile, **beige** = within median, **red** = beyond third quartile).

	ModSelection									
PlanSelection	**ems_lcf**	**hz_lcf**	**lcf_da**	**lcf_du**	**lcf_ems**	**lcf_hz**	**lcf_ia**	**lcf_iu**	**lcf_mod**	**lcf_pExp**
cloca	52	45	51	49	51	51	50	49	49	53
du_fmf	38	40	49	55	41	50	50	53	49	48
fhz_fmf	36	46	44	47	47	54	39	51	55	53
fmh_fmf	32	49	36	41	45	47	24	46	54	29
iu_fmf	37	40	50	54	54	51	46	60	44	52
lhz_fmf	43	34	51	51	45	47	52	53	44	44
psaoca	36	45	44	54	42	46	47	47	58	51
sdr_fmf	30	43	45	48	48	39	44	40	47	46

EMS	50		
UMCP	55	**UMCP+**	49
SHOP	41	**SHOP+**	53

Table 6.6: A strategy-component matrix for the ranks of stability values in the *Satellite* domain (**blue** = within first quartile, **beige** = within median, **red** = beyond third quartile).

We are now ready to examine some specific strategy combinations at a more detailed level. The strategy sub-set that is shown in Fig. 6.4 consists of prefix sharing combinations as well as combination rotations, that means, strategies in which the primary and secondary plan selection function are swapped. The set also includes two antagonistic plan selection preferences, the one that prefers the more constrained plans (`mcp` = $f^{planSel}_{ConstrPlans}$) and the less constrained plans (`lcp` = $f^{planSel}_{ConstrPlans^{-1}}$), respectively.

Our first observation is that there is hardly any stability dominance within the set of strategies. Since all combinations are equipped with the same primary and secondary modification selection functions, we take this and the above findings of a "missing" row pattern as a first evidence to a hypothesis that stability is determined by modification selection only. But it is probably also a side-effect of high failure ratios that produce trivial variances of 0 respectively *undefined*.

The performance characteristics of the antagonistic plan selection functions are positively in favour of the least constrained principle: most `lcp` combinations dominate most `mcp` combinations with respect to efficiency and reliability, without any converse result. The *Satellite* problem seems to be more adequately tackled by committing cautiously and *not* by trying to force an early failure. This argument is supported by the fact that one combination with the avoidance of too many HotZones (`hz_lcf_iu-lcp_fhz`) is among the final candidates in the efficiency-centered performance evaluation as well (Fig. 6.1). However, this finding is not very intuitive, because it has to be taken into account that prioritizing less developed plans induces a search space development that is similar to a breadth-first schema. On the other hand, the system avoids to deal with those plans that (prematurely) introduce too many causal dependencies.

In order to understand this behaviour of the strategy, consider Fig. 6.5; it shows the causal and temporal structure of a solution for the easiest *Satellite* problem. We can see that the temporal aspects of a *Satellite* plan are trivial: as we have shown in the problem descriptions, there is exactly one solution with seven tasks (including the initial and goal tasks) for the 1-1-1 problem and a corresponding number of parallel

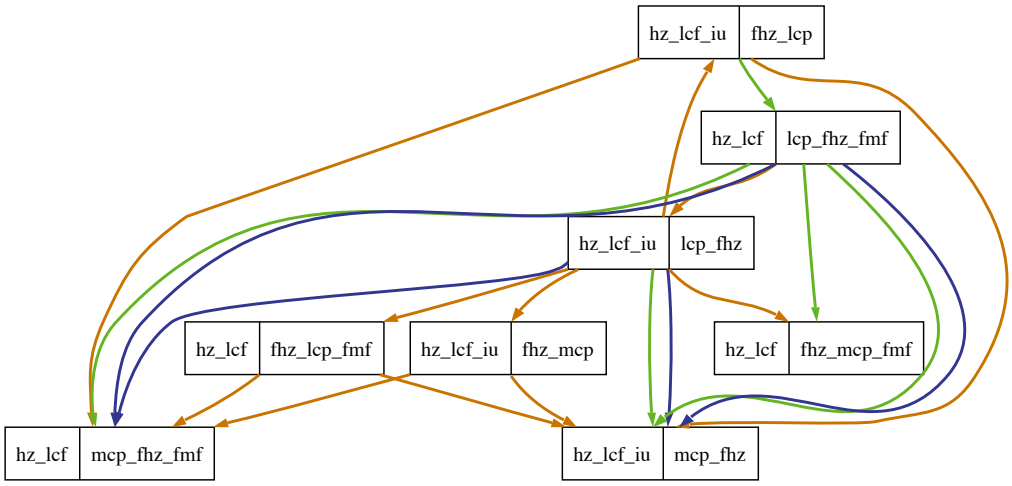


Figure 6.4: A detailed analysis of performance characteristics of related strategy combinations in the *Satellite* domain. The edges represent a dominance relationship with respect to efficiency (**orange**), stability (**green**), and reliability (**blue**).

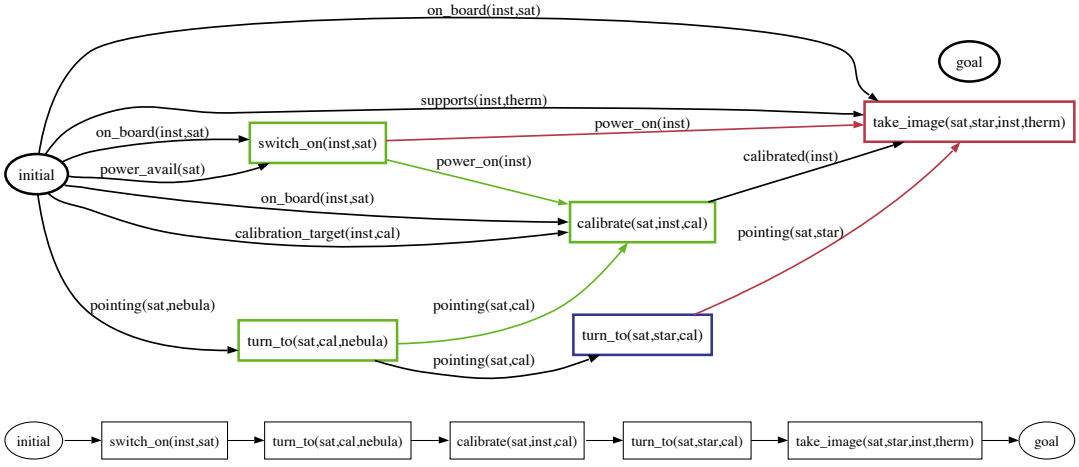


Figure 6.5: The causal structure (top) of a solution to the 1-1-1 problem in the *Satellite* domain and the induced temporal structure (bottom). The coloured task nodes and causal links indicate structures that are added to the plan in the context of one expansion each.

and interleaving chains in the more complex problems. However, even in the simplest problem, which is shown in the figure, it becomes obvious that the causal structure introduces a level of combinatorial complexity that can easily escalate. Most of the causal interactions are an easy task to organize for our hybrid planning configurations, because the causal support that is to be synthesized piecewise – depicted as black edges in the upper part of Fig. 6.5 – refers to rigid state features, that means the facts have to be deducible exclusively from the initial state description. The coloured causal links are introduced in the decomposition networks and are therefore only to be secured. The difficulty emerges from chosing the right decomposition method, as this determines whether or not the satellite platform is properly calibrated and orientated, and from adequately bridging the causal gap between the decompositions (in the figure, this means the black causal links between the two turning operations and the property of the instrument being calibrated). We may conjecture that this constellation produces a *certain* ratio of "inter-method" and "intra-method" causality which in turn makes an early causality establishment less useful or informative (cf. [144]).

Since the visited plan space is very small for the simplest *Satellite* problem, we are able to take a closer look at it and demonstrate the search behaviour that is described above. Fig. 6.6 displays the development of a solution for the 1-1-1 problem by a strategy that prefers less constrained plans to work on. The ellipses represent the examined partial plans, starting from the initial task network at the top, and are numbered in the order of their creation. The red ellipse marks a solution, blue stands for "open nodes" in the fringe that are yet to be developed, and light beige coloured plans are discarded ones (for they carry un-solvable flaws). Edges denote the respective plan refinements and are labeled with the applied plan modification class. The basic facts for this search space are the following: It consists of 62 visited nodes, has a branching factor between 1 and 5 (average approx. 1,4), and has a maximal depth of 17 (this is also the solution depth). We may also note that the ratio of open nodes is 5% and that of discarded plans 24%. It can easily be seen from the node generation order that the search pattern corresponds to a breadth-first schema – the figure highlights this by blue lines that show a characteristic sequence of fringes. It has to be noted that the emergence of such a regular search pattern is to some extent induced by regularities that are intrinsic to the *Satellite* domain model. If we take a closer look at the solution path, the measure of being constrained, that means, the ratio of the size of a plan's constraint sets to its number of steps, constantly increases from 1,3 to 7,3 by an amount of ca. 0,2 – except for the relatively elaborated very first task expansion which is not subject to strategic choice in this problem. This effect is more or less observable on all paths in the search space and therefore synchronizes the development of all refinement alternatives to proceed in strata.

When we look at the structure of the refinement space that is explored by the strategy with the antagonistic plan selection we immediately recognize a depth-first oriented search schema (Fig. 6.7). During the depicted plan generation episode – one of the better runs of that particular strategy – the system visits 51 nodes, has to face a branching factor between 1 and 5 (average approx. 1,5), and descends to a maximum depth of 18 nodes with the solution being available finally at a depth of 17. The ratio of open nodes is 4% and that of discarded plans 27%; the difference with respect to the previous example is obviously an effect of the depth-orientation which is known to have a lower space complexity[8] and to follow every path until it reaches a dead-end. The depth-first orientation of the strategy is caused by the above regularity argument, too, because the only task adding refinement (cf. experiment objective section) is that of apparently "constraint-balanced" expansions. This induces a monotonicity property for the values of the constraint ratio of successive refinements and the search has to become a depth-first one – irrespective of the deployed modification selection.

We also note that both strategies benefit from a late addressing of complex task flaws by the shared modification selection function. Remember that a HotZone-based modification selection focuses on the relationship of HotSpots in the plan structure, as it is defined by equation (4.30) (p. 169), and this avoids in our scenarios dealing with abstract tasks, which are typically flawed by open preconditions and open parameter bindings, too. If expansions occurred closer to the initial plan, this would be particularly disadvantageous to the unconstrained plan preference, because branching is highest in the decompositions, which means that many (futile) selection decisions have to be made repeatedly. On the 1-1-1 problem, for example, a simple expansion preference causes $f^{modSel}_{Pref-\mathbb{M}_{\texttt{ExpandTask}}} \triangleright f^{modSel}_{lcf}$ with $f^{planSel}_{ConstrPlans^{-1}} \triangleright f^{planSel}_{FewerHZones}$ to perform dramatically worse: it takes 116 plans to analyse, with a branching of 1 to 10 (1,6 on average). The open node ratio rises

[8] The fringe of a depth-first exploration will grow only factorial in size, in the order of *depth* × *branching factor*, while breadth-first schemata have to store $depth^{branching\ factor}$ many nodes.

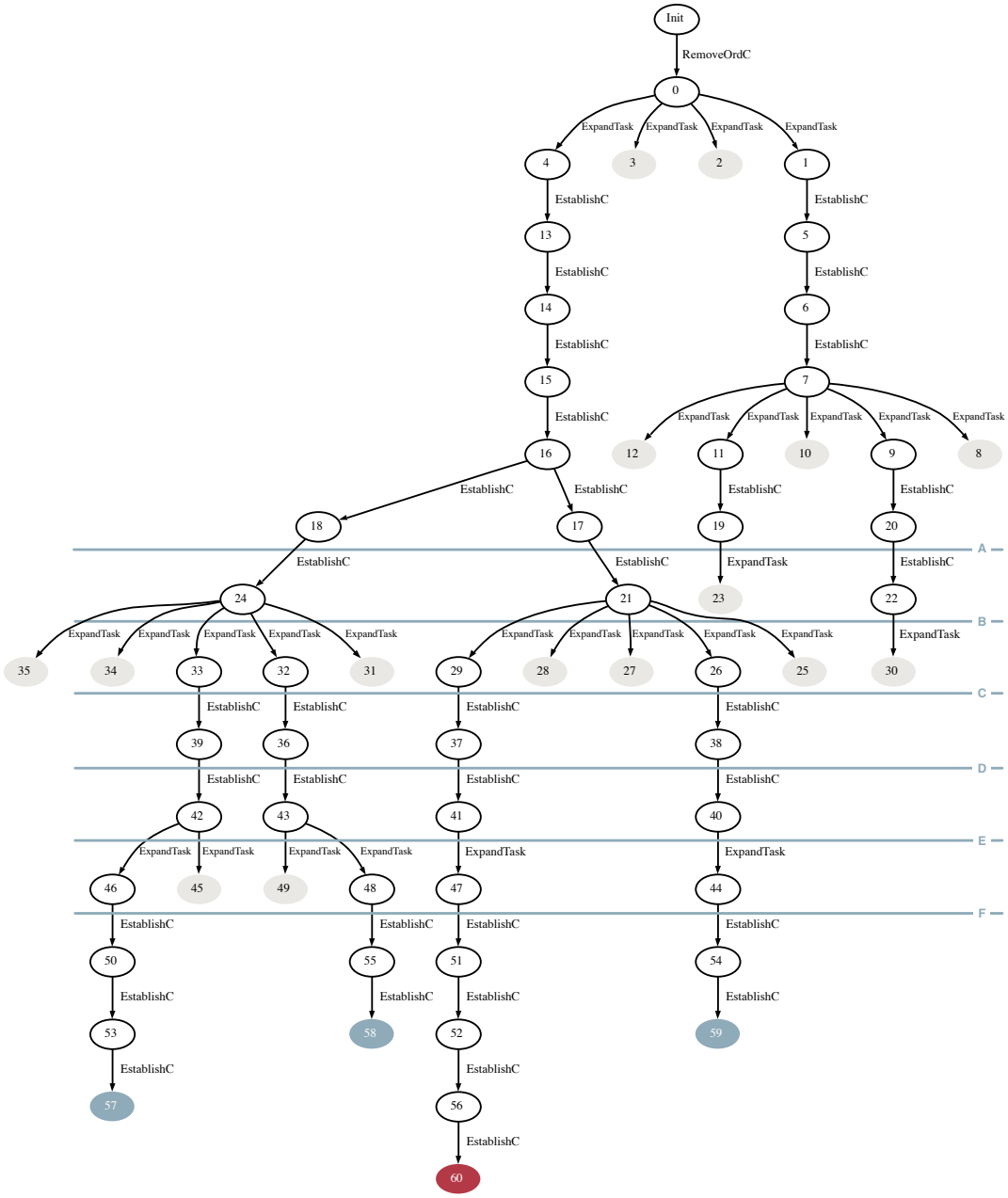


Figure 6.6: The spanned plan space by the modification selection $f^{modSel}_{HZone} \triangleright f^{modSel}_{LCF} \triangleright f^{modSel}_{IndUniHS}$ and plan selection $f^{planSel}_{ConstrPlans^{-1}} \triangleright f^{planSel}_{FewerHZones}$ with a hybrid planning system configuration on the 1-1-1 problem in the *Satellite* domain (**red** = solution, **blue** = open node, **beige** = discarded node).

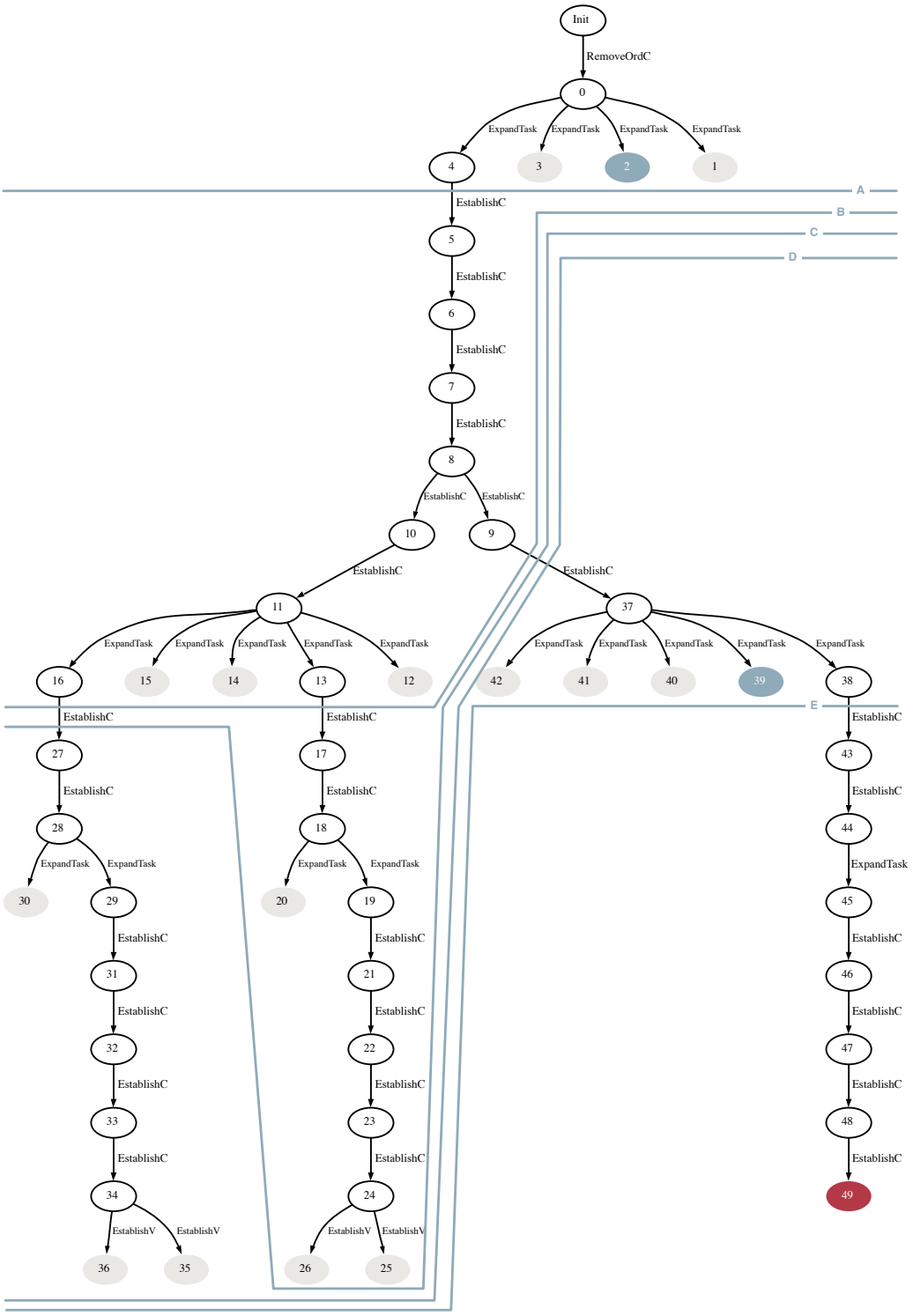


Figure 6.7: The spanned plan space by the modification selection $f^{modSel}_{HZone} \triangleright f^{modSel}_{LCF} \triangleright f^{modSel}_{IndUniHS}$ and plan selection $f^{planSel}_{ConstrPlans} \triangleright f^{planSel}_{FewerHZones}$ with a hybrid planning system configuration on the 1-1-1 problem in the *Satellite* domain (**red** = solution, **blue** = open node, **beige** = discarded node).

to 12% and 26% of the plans are discarded, which both is an effect of pursuing symmetric plan generation paths that lead to inconsistent plans.

All strategies in our detailed analysis are modulations of the two presented instances and basically show the same plan generation behaviour. In order to put this result into the perspective of our previous results, we consider for a moment an example for a strategy that is in a certain sense balanced between these two search schemata. It is a plan selection that primarily focuses on the modification HotSpots in the available plan modification set of a plan, the strategy that is built from $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ and $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ (cf. performance analysis in Fig. 6.1). Fig. 6.8 shows the explored topology; it takes the strategy 42 plans to analyze with a branching factor between 1 and 5 (average is 1,4). With an open node ratio of 1% and a discarding rate of 19%, the depicted sequence of fringes clearly indicates what we might call an opportunistic search schema. While it apparently does not pay off on the small problem instances, this schema turned out to be one of the most efficient ones in the *Satellite* domain.

Let us now continue with the performance characteristics analysis in Fig. 6.4: As expected, switching the primary and secondary plan selection functions affects the combination's performance significantly. The two strategies with the $f^{planSel}_{ConstrPlans^{-1}} \triangleright f^{planSel}_{FewerHZones}$ prefix dominate the $f^{planSel}_{FewerHZones} \triangleright f^{planSel}_{ConstrPlans^{-1}}$ combinations in terms of efficiency, the analogous constellation for the "prefer constrained plans" strategies however turns out to be less determined. The $f^{planSel}_{ConstrPlans} \triangleright f^{planSel}_{FewerHZones}$ combinations are not all dominated, but all are on most instances less problem efficient.

Adding an established strategy as a third plan selection function (preferring plans with fewer refinement options as a compensation for a third modification selection) increases the efficiency for one the $ConstrPlans^{-1}$ combinations but not their stability like we expected in the experiment design phase. This is of course only a very rudimentary result that needs substantially more experimentation with respect to which kind of strategies make beneficial decisions in those combinations, how many decisions actually can be made in the third position, and the like. But we believe this finding gives evidence to the conjecture that plan selection function $f^{planSel}_{Fewer-\mathbb{M}}$ is at least compatible[9] with the preference of less constrained plans and, which is of more general relevance, that stability is to a greater extent linked with the modification selection than with the plan selection functions.

The question of strategy stabilization by selection function sequencing is closely related to that of the performance relationship of strategies that are closely related from the methodological point of view. The relation of the plan selection functions $f^{planSel}_{\text{CL+OCA}}$ and $f^{planSel}_{\text{PSA+OCA}}$ has been discussed in Sec. 4.1.2 and the experimental results support our claim that a proper assessment of added plan steps is an essential factor in the A^* heuristic. In the end, a PSA+OCA derivate is among the best evaluated strategies, while the whole CL+OCA strategy family suffers from a very low stability and reliability, and produces the worst efficiency values.

Deploying a plan selection according to HotZone criteria is apparently a successful concept. For example, we see that five such strategy combinations are very efficient in the *Satellite* domain (see Fig. 6.1). But as we have observed before, the preference of fewer HotZone sections in a plan seems to be less effective than concentrating on the minimum HotZone values (cf. Tab. 6.4). However, a direct comparison gains a slightly different perspective on this issue: In fact, there is no single efficiency dominance result between two strategy combinations with primary $f^{planSel}_{FewerHZones}$ respectively $f^{planSel}_{LeastHZone}$. Furthermore, there are practically as many FewerHZones strategy combinations more stable than LeastHZone representatives (4 to 3), and the reliability result completely contradicts the global trend, because FewerHZones dominates LeastHZone in no less than 14 cases (and no counter example). Since we cannot provide yet a coherent interpretation of these findings that would explain this evident discrepancy, we suggest further experimentation in order to get a better picture of the two mechanisms.

The strategies *SHOP* and *UMCP* will be focused on later, at this point we are interested in the relationship between the two classical implementations and their derivates with plan alternative plan selections $f^{planSel}_{\text{PSA+OCA}}$ (`shop+` in tables 6.2, 6.5, and 6.6) and $f^{planSel}_{Addr-\mathbb{F}_{\texttt{AbstrTask}^{-1}}}$ (`umcp+`, idem). The *EMS* component is deployed

[9] We do not have yet a formal notion of compatibility. For the time being, we may consider two strategy components compatible if one of them benefits from the other as a subsequent strategy in terms of any performance characteristic. Compatibility is obviously an irreflexive, transitive relation that has to be redefined for any planning domain (and at least reconsidered for any major change of problem classes).

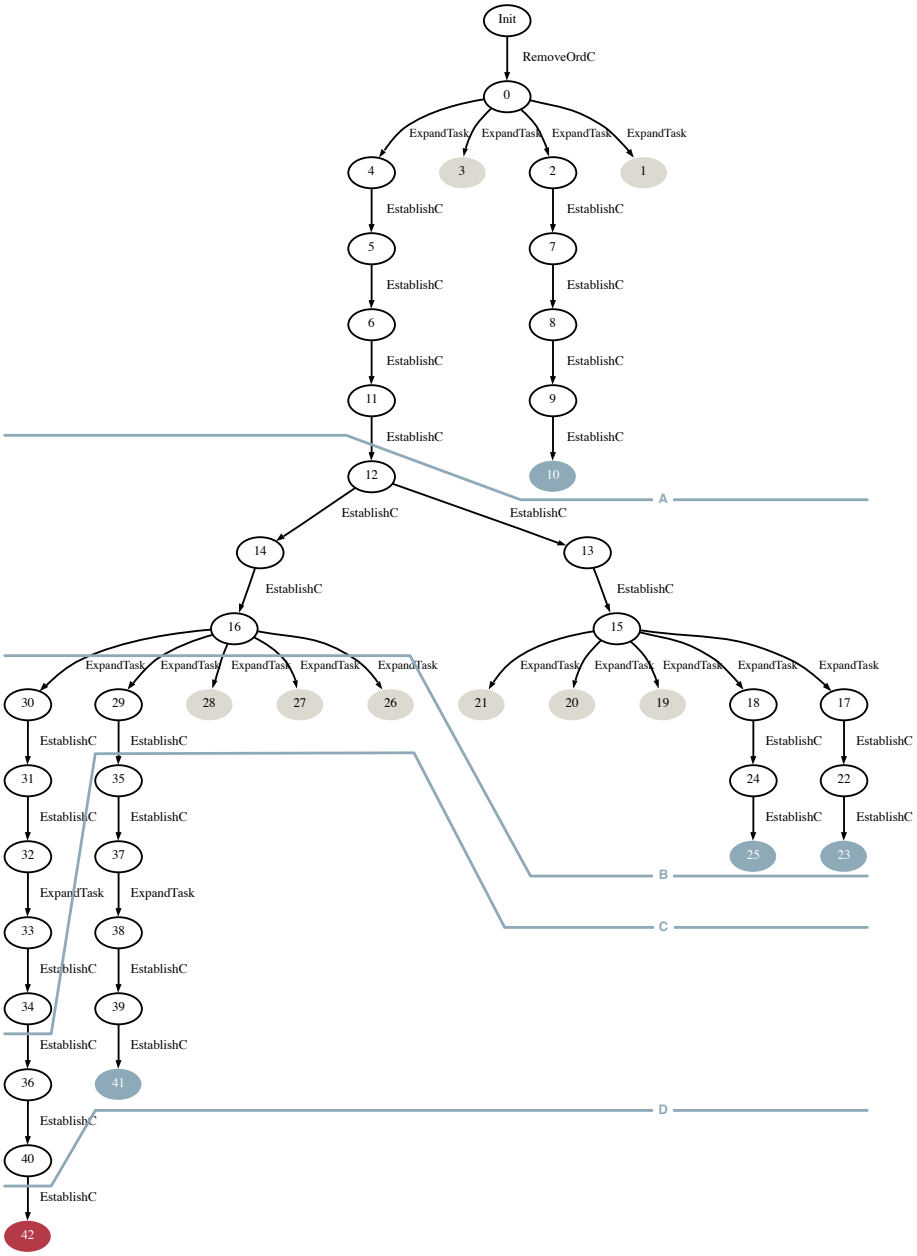


Figure 6.8: The spanned plan space by the modification selection $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ and plan selection $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ with a hybrid planning system configuration on the 1-1-1 problem in the *Satellite* domain (**red** = solution, **blue** = open node, **beige** = discarded node).

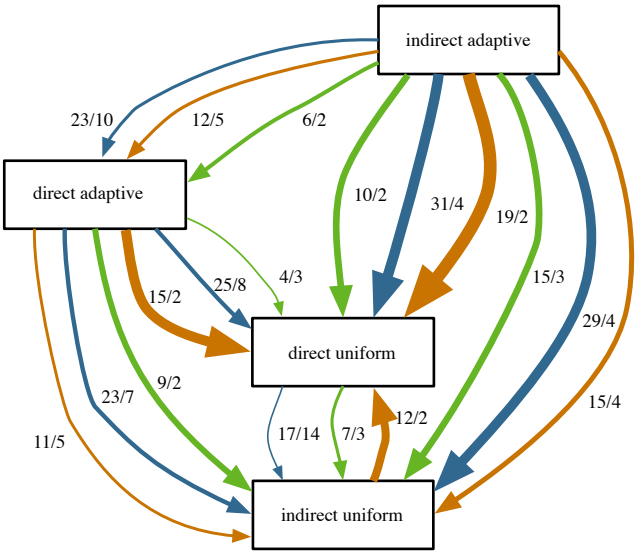


Figure 6.9: Quantitative dominance comparison between direct, indirect, adaptive, and uniform modification selection in the *Satellite* domain. Edge labels denote number of dominant instances in / contrary to edge direction; edge width corresponds to dominance ratio (**orange**=efficiency, **green**=stability, and **blue**=reliability).

solely only in its classical form (see below), richer combinations have already been treated above.
The rationale behind `shop+` is to combine the in all performance categories apparently very effective modification selection with a very well performing plan selection. The result is a bit disappointing, although not too surprising: the new `shop+` combination performs significantly worse and is even one of the least performing PSA+OCA combinations. The SHOP method is based on a specific temporal treatment of flaws in a plan (cf. Def. 4.41) and that refinement order cannot be preserved by the PSA+OCA A^* heuristic.
`umcp+` was intended as an example for repeating the modification selection principle in the plan selection and it improved the results for the classical variant with respect to reliability and stability at a minor loss of efficiency.

Our evaluation population contains a number of strategy combination instances that are built from direct uniform and indirect uniform HotSpot components (cf. strategy definitions (4.23) to (4.26) ff). When we compare the performance measures of the respective system configurations we find that in 26 cases those which are based on the *direct* uniform HotSpot plan selection ($f^{planSel}_{DirUniHS}$) dominate in terms of efficiency the ones that rely on the *indirect* uniform HotSpot function; the converse can only be found in 8 cases. Regarding stability, basically the same result holds: the direct variant beats the indirect plan selection $f^{planSel}_{IndUniHS}$ in 24 comparisons and looses in 7. In terms of reliability, both strategy types perform comparable (28 to 30). This result is not consistent with our hypothesis that the finer granular indirect HotSpot computation is better informed about the plan development status (the second conjecture, that this also depends on preceding and subsequent strategy components, cannot be addressed in this experimental setting). We do not have yet an explanation for this phenomenon and refer to corresponding analyses in other domains below.

Concerning the combinations that deploy the respective modification selections, our findings mostly confirm the expectations raised in the strategy sections. Fig. 6.9 shows the detailed numbers of dominance results in the *Satellite* domain. It is apparently almost always the case that an indirect HotSpot calculation is an advantage over a direct one for both adaptive and uniform modification selection functions: For example, a combination with a (secondary) indirect adaptive modification selection is in 23 cases more reliable, in 12 more efficient, and in 6 more stable than a combination with a direct adaptive component. The only exception to this clear superiority is the reliability parity and small stability inferiority for indirect uniform combinations. An unambiguous result is the relative dominance of the adaptive modification selections over the uniform ones. The biggest gain can be achieved in terms of efficiency, followed by reliability

	Efficiency		Stability		Reliability	
Strategy	dom.	dom. by	dom.	dom. by	dom.	dom. by
CL+OCA*	4	52	5	12	6	48
PSA+OCA**	20	9	1	8	37	19
UMCP	16	1	0	15	31	3
SHOP	35	0	1	0	21	0
EMS	50	0	2	5	34	2

* Average dominance values for 10 combinations.

** Average dominance values for 11 combinations.

Table 6.7: The dominance situation for the classical strategy implementations in the *Satellite* domain.

and stability. This encourages us to suggest future experimentation on that matter; further investigation on suitable initial weights and adaptation increments may improve the absolute result substantially (cf. definitions (4.27) and (4.28), Appendix A).

Since the general behaviour of the uniform plan selection seems to contradict that of the modification selection functions, let us take a closer look at the component performance tables 6.2, 6.5, and 6.6: An indirect uniform plan selection definitely suffers from the analogous modification selection, and vice versa, for the rankings of the $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndUniHS}$ $f^{planSel}_{IndUniHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ are the worst in the respective row and column. The situation is similar in the direct uniform cases, although the negative "amplification" is less significant in view of the column values, but for the direct uniform plan selection most other modification selections produce better results. Furthermore, mixing direct and indirect uniform selections does not improve the rankings either.

Other candidates for compatibility phenomena are the modification selection f^{modSel}_{HZone} and the two deduced plan selection functions that focus on plans with the "least hottest zone" or the fewer HotZone components: For both plan selections, the HotZone modification preference is better used as a primary selection function (the candidate set also contains an inferior combination with a prefixed least commitment function). According to the results, it is in particular the preference of fewer HotZones that benefits from a HotZone pre-selection.

Analysis of Classical Strategies

Let us now turn to the classical strategies that participated in the evaluation experiments. As we have seen before, the plan selections that are based on A^* schemata perform very well if the addition of plan steps is computed in terms of plan modifications (PSA+OCA) and not in terms of absolute numbers (CL+OCA). We have also documented the success of the classic strategies *SHOP* and *EMS*; both perform on a high level and although they are not in the same quartile for all solved problems, they seem to benefit respectively suffer from the same problem characteristics. The most visible difference is that the *SHOP* strategy is considerably more stable. One of our evaluation questions was where do these strategies, together with the classical *UMCP* strategy, stand in the whole field of competitors? The precise numbers of dominating respectively dominated strategy combination instances for each performance category are given in Tab. 6.7. While our performance evaluation concentrated on the property of being undominated, this table shows how many competing individuals are actually dominated and how many negative results exist. It has to be noted that although every CL+OCA combination dominates on average only 4 other combinations in terms of efficiency, the dominated strategy is in 90% of all cases a fellow CL+OCA; regarding reliability, this rate is still at 50%. The much more often dominanting PSA+OCA combinations, on the other hand, have established themselves in the evaluation population so that nine out of ten dominated strategies are based on a different plan selection function.

In the context of hybrid planning system configurations, it is particularly interesting whether or not an early conflict resolution is beneficial in the *Satellite* domain or not. This question can be answered by a direct comparison of the strategies *SHOP* and *EMS* with *UMCP*: do the former dominate the latter? Fig. 6.10 shows

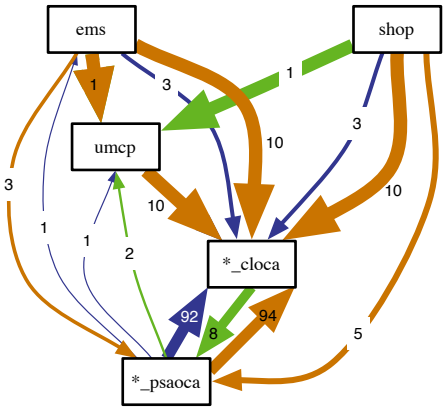


Figure 6.10: Quantitative dominance comparison between classical strategy combinations in the *Satellite* domain. Edge labels denote number of dominant instances; edge width corresponds to the ratio of dominance declarations to the total number of combinatorial possibilities (**orange**=efficiency, **green**=stability, and **blue**=reliability).

the dominance results for the subset of our classical strategies; the width of the arrows represents the ratio of actual dominance findings to the total number of combinatorial possibilities. For example, the orange arrow from strategy *SHOP* to CL+OCA stands for ten cases in which the former was found more domain search-efficient than the latter. Since there is one *SHOP* instance and ten combinations that deploy the CL+OCA plan selection, we have a full coverage and the arrow is assigned the maximum width. Besides the confirmed results concerning those combinations that deploy $f^{planSel}_{\text{CL+OCA}}$ and $f^{planSel}_{\text{PSA+OCA}}$, the figure also includes the relationships between our focused strategies.

Surprisingly, we cannot come to a clear judgement about the alternatives *SHOP*, *EMS*, and *UMCP*. If we compare our previous findings in

- Tab. 6.2 to 6.6: *SHOP* and *EMS* have superior rankings in all categories;
- Tab. 6.7: *SHOP* dominates twice as many competitors than *UMCP* via efficiency but 30% less via reliability, *EMS* dominates with its efficiency about three times more combinations than *UMCP*, while both perform more or less equivalently with respect to reliability;
- Fig. 6.10: The only dominance results in this context are *SHOP* and *EMS* dominating *UMCP* in terms of stability respectively efficiency.

In summary, the results indicate that UMCP is definitely not the first choice of strategy in the *Satellite* domain but the fact that some of the most efficient subjects in the experimental population do not dominate it leads to the assumption that probably (simple) derivate combinations of UMCP will perform satisfactory. We therefore conclude that with the exception of $f^{planSel}_{\text{CL+OCA}}$ **all** classical strategies are reasonably well performing candidates for further experimentation in the *Satellite* domain. This is in particular the case for the versatile $f^{planSel}_{\text{PSA+OCA}}$ strategy component. Since there is no clear evidence for the systematic superiority of an early (that means, abstract) threat detection, we recommend a focused exploration of extension strategies in order to obtain a broader data basis.

Problem Analysis

When we formulated the set of research questions that should be addressed by the experimentation (Def. 6.6), we also included the question of what information the performance results give about the experimental setting they are evaluated in. We regard a careful inference of difficulty characteristics from the empirical results as a legitimate extrapolation because of the number and variety of deployed planning strategies.

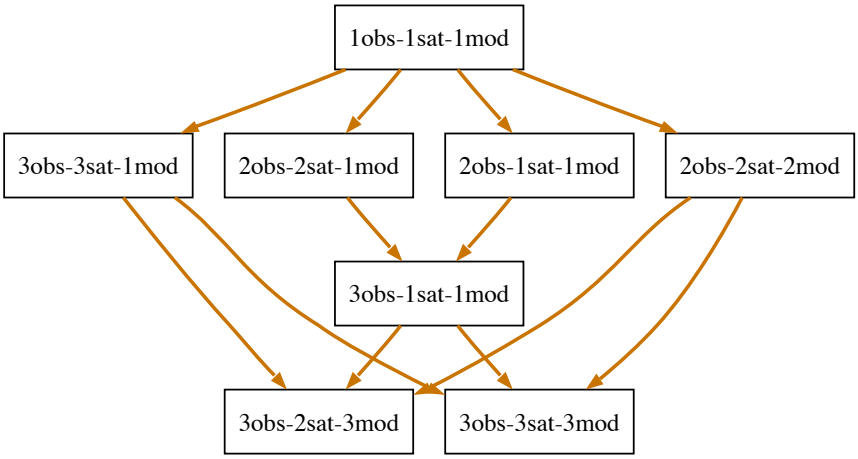


Figure 6.11: Difficulty of the problems in the *Satellite* domain in terms of efficiency.

To this end we applied to the problem instances the same data processing techniques that we used for the efficiency- and reliability-centered performance evaluations. In correspondence to previous definitions like 6.4, we understand the *Satellite* problem instances as subjects that apparently *can only be solved* by a number of individuals *with a certain efficiency*. In this way, a problem *a* can be seen more *efficiency difficult* than a problem *b* if all strategy individuals are less efficient on *a* than they are on *b*; stability and reliability can be formulated analogously (for better readability, we will use a boldface font for the categories in the following sections). Although all obtained results are solely valid in the context of the participating strategies, we may argue that if the strategy repertoire is representative or diverse enough then this empirical litmus test can categorize a given problem set into a "universal" difficulty schema, because there is obviously no[10] structural particularity in the problem that a strategy could exploit by some sort of "built-in trick" or bias.

The results of such an evidence based problem analysis in the *Satellite* domain are shown in the three diagrams Fig. 6.11 to 6.13. For example, the efficiency dominance between problem instances 1obs-1sat-1mod and 2obs-2sat-2mod means that all strategies that participate in the evaluation perform more efficient (on an absolute scale) within the 1-1-1 problem than on the 2-2-2 problem. We also interpret the fact that 1-1-1 is undominated in all three characteristics as an evidence for that problem being "the simplest" in the evaluation of that domain. Most of the observable dominance relationships are intuitively plausible and so is also the emerging of "strata": most problems in the *Satellite* domain are more difficult than other problems if they include more observations, more satellites, and more modes. This would be a trivial result – if it were not only *most* strategies that obey that rule. We will inspect some of these oddities.

As we have discussed in the experimental setup section, the number of required modes in which images have to be taken imposes restrictions on the solution due to the different capabilities of the satellites. These restrictions become no sooner visible than during the identification of a causal link producer for an open precondition on the `supports` predicate. At this point, the plan might already be committed by previous refinements to an unsuitable satellite object; consequently, the open precondition flaw cannot be answered in this case and the plan cannot be developed into a solution. This situation of a late inconsistency detection causes the planner to waste immense search efforts and is thereby obviously more influential on **efficiency** than the ambiguity implied by the number of available observation platforms (otherwise, 2-1-1 and 3-1-1 would dominate 2-2-1 and 3-3-1). The reason for that lies in the fact that it does not matter whether "redundant"[11] satellites are involved in the solution or not; most of the observation implementation options are consistent with unused satellites and even an unnecessary second calibration step is tolerable because it can be achieved within one complex task expansion. As a consequence, in the 2-2-1 scenario roughly six out of seven combinations of a satellite assignment and a primary decomposition method selection do yield

[10] or at least no obvious...

[11] Note that the evaluation candidates are not deployed in a resource-aware configuration; the described notion of redundancy would otherwise be considered as a relaxation of the problem.

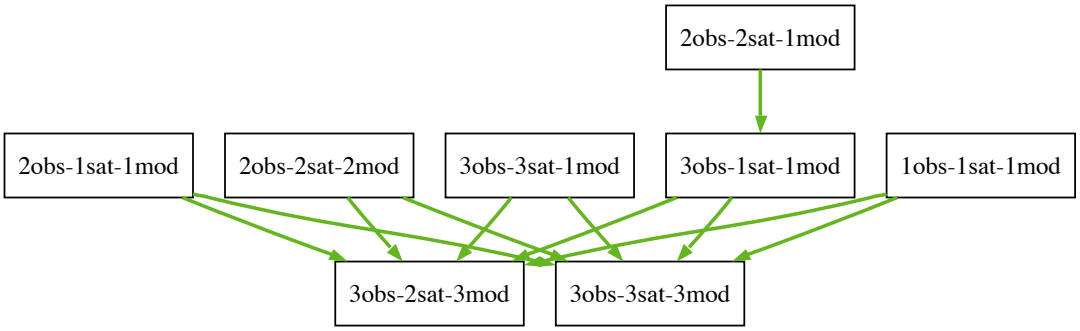


Figure 6.12: Difficulty of the problems in the *Satellite* domain in terms of stability.

a solution, which does not substantially challenge the planning strategies (for a comparison: in the 2-2-2 setting this ratio is only three out of seven). For these cases, a solution that is constructed in a sub-optimal way is obtained more **efficiently** than one that produces too many dead ends. If we furthermore assume minor "noisy" variations in the search space topology of our candidates, we can generalize this result such that *Satellite* problems of the form $x$obs-$y$sat-$z$mod are less **efficiency** difficult than problems x'obs-y'sat-z'mod with $x \leq x'$, $y \leq y' + \varepsilon$, and $z \leq z'$. So far, this matches our findings and is an explanation for 3-3-1 and 2-2-1 being easier than 3-2-3 and 3-1-1, respectively. The available data does however not indicate where the actual equilibrium between additional satellites ε and the differences of observations and modes may be located; the currently undecided pairs 2-2-1/2-1-1 and 3-3-1/3-1-1 suggests that ε may be larger than 2. We propose to conduct experiments with additional problem instances that include more (redundant) satellites and modes.

Having said that, an anomaly that we are not able to explain yet is lack of evidence for the 3-3-1 problem being more **efficiency** difficult than any of the problems dealing with two observations and in particular not more difficult than the 2-2-1 instance that is included as a sub-problem. It is an open issue, why of all strategies it is those few capable of solving the harder 3-x-y instances that are having exceptional difficulties with the 2-x'-y' ones. Furthermore, it is also not clear, why the 2-1-1 problem has not been solved by strategies that are able to solve those problems with more satellites and modes. We believe that these issues may arise from too small sample sizes and therefore we see the necessity for further experimentation at this point.

We will briefly look into the results for the other two performance characteristics. Fig. 6.12 shows the **stability** dominance of problems in the *Satellite* domain. Unfortunately, the obtained data cannot be reasonably analyzed because of the large number of unterminated runs that leaves a number of strategies with one single successful plan generation episode which in turn results in a stability value of 0 (cf. Tab. B.2). We have to conclude that **stability** is not generally suitable for analyzing problem characteristics, since its interpretability strongly depends on reasonably well performing strategy candidates as well as on large sample sizes. We will revisit this issue for the other evaluation domains.

For the time being, we note that **stability** is probably only useful as an additional difficulty measure; for the *Satellite* domain, it does neither contradict any previous result nor provide further insights.

Let us finally inspect the **reliability** dominance as it is projected on the planning problems (Fig. 6.13). It can be seen that this notion of difficulty corresponds perfectly to the results we obtained in the strategy analysis: In the *Satellite* domain **reliability** correlates with efficiency. This characteristic does not only confirm the previous findings, it also provides the "missing" dominance result between the 2-2-1 instance and 3-3-1 – the stratification is now more like expected. Given the above findings, we conclude with proposing a problem-difficulty measure that is based on reliability-centered performance (which is in this case equivalent to the **reliability** analysis). The necessity for this measure lies in the fact that only a primary focus on the failure ratios allows a proper data interpretation on inhomogeneously performing subjects in the strategy candidate set and in particular if the proportion of over-strained strategies is high (see the average failure ratio rates in the reliability compilation in Tab. B.3, the overall average is 72%).

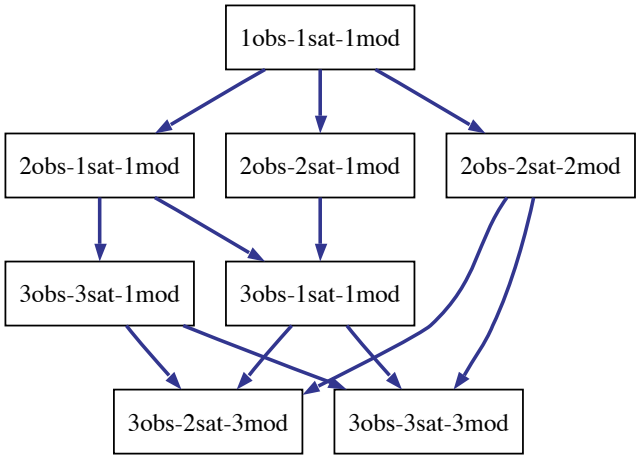


Figure 6.13: Difficulty of the problems in the *Satellite* domain in terms of reliability.

6.4 Evaluation Results in the *UM Translog* Domain

Efficiency-Centered Performance Evaluation

In the efficiency-centered performance evaluation of the *UM Translog* domain, we have to deal with almost twice as much candidate subjects as in the *Satellite* domain. Fig. 6.14 shows the results: the 33 nodes in the graph are those strategies that are undominated in this domain with respect to efficiency, green edges represent an additional domination in terms of stability, and blue edges record reliability domination. We note that we did not experience any contradictory result with respect to successive dominance relations.

As we can see, the efficiency-centered schema leaves four strategies undominated, that is to say, the following four strategies are regarded as the best performing ones:

- $f_{EMS}^{modSel} \triangleright f_{LCF}^{modSel}$ with $f_{\mathbb{F}/TE}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$
- $f_{LCF}^{modSel} \triangleright f_{DirAdaptHS}^{modSel}$ with $f_{FewerModBHS}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$
- $f_{LCF}^{modSel} \triangleright f_{IndAdaptHS}^{modSel}$ with $f_{FewerModBHS}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$
- $f_{LCF}^{modSel} \triangleright f_{ModBasedHS}^{modSel}$ with $f_{FewerModBHS}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$

Interestingly, the second strategy was also among the best performing ones in the *Satellite* experiments. It is also worth noting that reliability subsumed all but one stability relationship and is in the *UM Translog* domain an apparently more influential measure. This is most probably a side effect of a domain-wide smaller failure ratio; we will come to this point below.

Although the most effective strategies do perform better than before in the sense that the number of dominated competitors is significantly higher on average according to all characteristics, it is remarkable that the $f_{EMS}^{modSel} \triangleright f_{LCF}^{modSel}$ / $f_{\mathbb{F}/TE}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$ strategy with its only 7 efficiency- and 5 reliability-dominated rivals is finally one of the best. This evaluation also confirms the standing of the classical strategy combinations with *EMS*, *UMCP*, and *SHOP* participating at least in the undominated efficient candidate set.

Fig. 6.15 puts the absolute efficiency of our winning candidates into a statistical perspective of the whole set of participating subjects. As we can see, the winner's performance level in the *UM Translog* experiments is considerably better than before and only three or four problems seem to be relatively difficult. However, please note that these results are merely relative to the complete experimental series; reducing the competition on the efficient candidates and rerunning the evaluation will fan out the values on the given scale. But we can nonetheless record that for the time being our best performing strategies are not only un-beaten by other candidates but are very absolute-efficient as well.

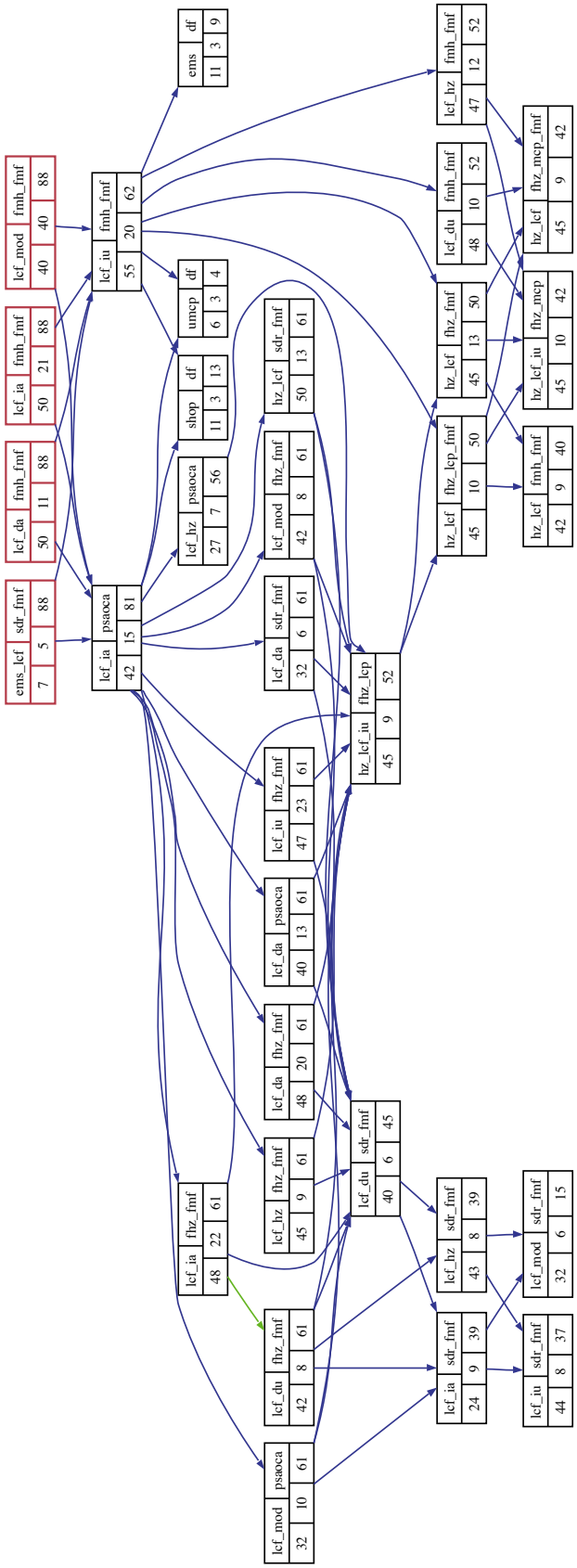


Figure 6.14: The efficiency-centered performance evaluation in the *UM Translog* domain. Nodes are the most efficient strategies, edges represent stability (**green**) and reliability (**blue**) dominance. Node annotation shows number of strategy combinations dominated by the node in terms of efficiency, stability, and reliability (**red** = undominated strategy).

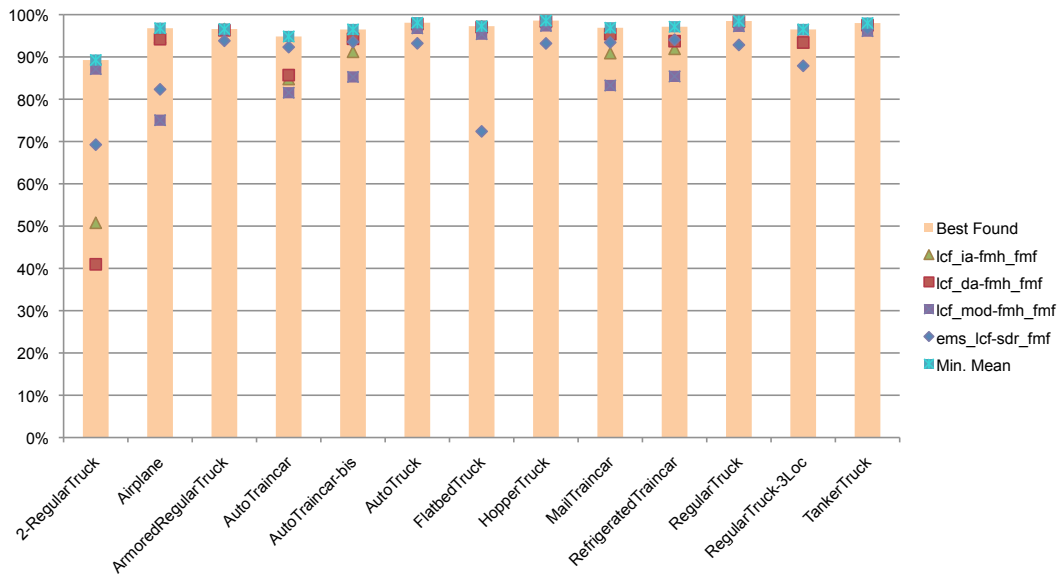


Figure 6.15: The *relative* search efficiency of the undominated strategies in the *UM Translog* domain. 100% on the vertical axis corresponds to a "perfect" strategy that solves the problem immediately, 0% represents the worst efficiency value in the experiments.

Let us now turn to the component-wise efficiency analysis that is shown in Tab. 6.8. The *UM Translog* domain does not share most of the tendencies that we observed in *Satellite*. The most obvious opposite results are the bad efficiency of the strategies that deploy the $f^{planSel}_{LeastHZone}$ primary plan selection (the row labeled "lhz_fmf") or the $f^{modSel}_{EMS} \triangleright f^{modSel}_{LCF}$ modification selection. On the other hand, the CL+OCA plan selection criterion is considerably better ranked than before.

	ModSelection									
PlanSelection	**ems_lcf**	**hz_lcf**	**lcf_da**	**lcf_du**	**lcf_ems**	**lcf_hz**	**lcf_ia**	**lcf_iu**	**lcf_mod**	**lcf_pExp**
cloca	84	44	47	51	65	51	43	46	47	64
du_fmf	84	68	65	70	78	69	62	64	65	78
fhz_fmf	62	21	22	27	35	27	20	23	19	38
fmh_fmf	82	21	12	14	27	14	11	8	22	30
iu_fmf	84	67	65	69	77	68	61	64	63	78
lhz_fmf	83	63	64	70	78	67	61	63	66	78
psaoca	82	35	30	41	44	34	23	40	27	48
sdr_fmf	40	27	48	75	38	14	27	25	34	22

EMS	43		
UMCP	48	**UMCP+**	31
SHOP	34	**SHOP+**	75

Table 6.8: A strategy-component matrix of the average ranks of efficiency values in the *UM Translog* domain (**blue** = within 1^{st} quartile, **beige** = within median, **red** = beyond 3^{rd} quartile).

A positive association with efficiency is slightly less obvious than before, but it seems to be the case for the (primary) plan selection functions $f^{planSel}_{FewerHZones}$ (fhz_fmf), $f^{planSel}_{FewerModBHS}$ (fmh_fmf), and $f^{planSel}_{\mathbb{F}/TE}$ (sdr_fmf), as well as for the modification selection functions $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ and $f^{modSel}_{LCF} \triangleright f^{modSel}_{DirAdaptHS}$. Negative efficiency associations can be observed with the before mentioned $f^{modSel}_{EMS} \triangleright f^{modSel}_{LCF}$ modification and $f^{planSel}_{LeastHZone} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ plan selection, but also with the plan selection functions $f^{planSel}_{DirUniHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ and $f^{planSel}_{IndUniHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$, which are mostly ranked in the last quartile of the data set. It is also worth pointing out that, ironically, the direct as well as the indirect HotSpot principle does work exceptionally well for modification selection but at the same time exceptionally badly for plan selection. This tendency holds conversely for the *Satellite* domain but not as clearly as it does here.

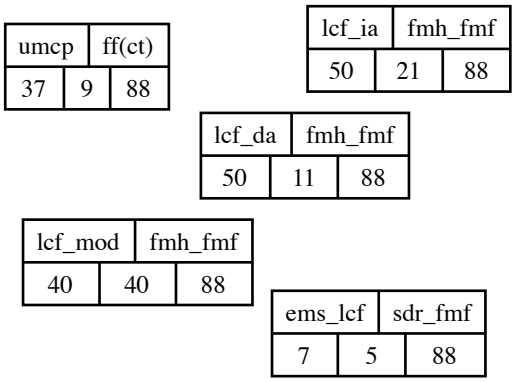


Figure 6.16: The reliability-centered performance evaluation in the *UM Translog* domain. Nodes are the most reliable strategies; there exists no additional dominance result, hence all candidates are undominated according to this evaluation schema. Node annotation shows number of strategy combinations dominated by the node in terms of efficiency, stability, and reliability.

Facts that particularly deserve our attention are the crossing points of selection functions with extra positive and negative association in the component matrix. An indeed, the data supports again the "accumulation hypothesis", since the positive crossing points are all in the first quartile's rankings (including the two best average rankings 8 and 11) and the negative ones in the last quartile, respectively (including two combinations with the worst average ranking of 84).

We believe that the latest analysis substantiate an implicit finding in the previous sections: The presented portfolio of examination methods is in fact necessary to judge the strategies properly, even more so if "performance" is understood in multiple ways. As we have argued repeatedly, performance in terms of any characteristic has to be seen in many dimensions if we are looking for *developable* strategy combinations. A good example is the above component analysis, which provides a more abstract view on the evaluation candidates but does not point explicitly to the winning strategies according to the efficiency-centered schema.

Reliability-Centered Performance Evaluation

Regarding the reliability-centered performance evaluation of our strategy set in the *UM Translog* domain, we make a similar observation as for *Satellite*; the candidate set that is derived from a reliability dominance analysis is considerably smaller than that we obtained from efficiency. But also, as Fig. 6.16 shows, these candidates cannot be ranked further by a subsequent characteristic, and the best performing strategies according to this schema are therefore the following five:

- $f^{modSel}_{LCF} \triangleright f^{modSel}_{DirAdaptHS}$ with $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$
- $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ with $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$
- $f^{modSel}_{LCF} \triangleright f^{modSel}_{ModBasedHS}$ with $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$
- $f^{modSel}_{EMS} \triangleright f^{modSel}_{LCF}$ with $f^{planSel}_{\mathbb{F}/TE} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$
- f^{modSel}_{UMCP} with $f^{planSel}_{Addr-\mathbb{F}_{\texttt{AbstrTask}^{-1}}}$ (the modified classic *UMCP+*)

With the exception of the *UMCP* derivate, all of these are also the best efficiency-centered performing strategies.

When we look at the reliability values of the best performing strategies, the results are much more closely placed on the statistical scale of the *UM Translog* domain (Fig. 6.9). This time, the winning strategies solved all posed problems, and also the other descriptive measures attest the evaluation candidates a reasonably

reliable behaviour, for example, a quarter of the strategies only failed on less than 10% of the runs. The only counter examples are those three strategies that actually terminated not on a single plan generation episode successfully.

Strategy	**Average Failure Rate**
Best value	0%
$f_{LCF}^{modSel} \triangleright f_{DirAdaptHS}^{modSel}$ with $f_{FewerModBHS}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$	0%
$f_{LCF}^{modSel} \triangleright f_{IndAdaptHS}^{modSel}$ with $f_{FewerModBHS}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$	0%
$f_{LCF}^{modSel} \triangleright f_{ModBasedHS}^{modSel}$ with $f_{FewerModBHS}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$	0%
$f_{EMS}^{modSel} \triangleright f_{LCF}^{modSel}$ with $f_{\mathbb{F}/TE}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$	0%
f_{UMCP}^{modSel} with $f_{Addr-\mathbb{F}_{\texttt{AbstrTask}}{}^{-1}}^{planSel}$	0%
1. Quartile	8%
Average value	23%
Median	17%
3. Quartile	31%
Worst value	100%

Table 6.9: Positioning of the most reliable strategies within the field of competitors in the *UM Translog* domain.

Analysis of Strategy Components

As we did above for the *Satellite* domain, we are now focusing on the dominance results per primary plan selection, thereby basically merging the rows of the qualitative ranking matrix (Tab. 6.8) into quantified occurrences of un-dominance. In contrast to those previous findings, we see that *UM Translog* does leave a greater percentage of strategies undominated (29%), presumably caused by a more diverse set of problem characteristics such that it becomes less probable for a single strategy to consistently perform superior to any other (which in turn produces more dominated strategies, cf. Tab. 6.10). As we can see, the best performing candidates do come from strategy families above the average threshold of 29%, that is to say, they are recruited from those strategies that are found undominated in all characteristic category above-average.

In order to examine the strategy components in more detail, we break up the results into component matrices – like we did above for efficiency – according to reliability and stability (Tab. 6.11 and 6.12). In contrast to the *Satellite* results, the tendencies are more visible and more coherent this time. The well-ranked plan selections, which were built on HotZone, modification HotSpot, and detection-ratio based primary selection functions are apparently the most reliable and stable ones, too. Also the negative results can be confirmed, that means, the direct and indirect uniform HotSpot selections and the least "hottest" HotZone heuristic.

Regarding observability of a trend, it becomes in particular obvious that in solving a *UM Translog* problem the proper modification selection becomes a key issue with respect to stability. Consequently, if we also take into account that in this domain all characteristics more or less correlate, we are inclined to propose that Tab. 6.12 alone gives us enough evidence to avoid at least modification selection that deploy the *EMS* and Pref-$\mathbb{M}_{\texttt{ExpandTask}}$ functions. However, we have to remember that one of the winning strategies has been constructed from such a negatively associated modification selection!

Let us now concentrate on that subset of the strategies that includes antagonistic combinations and cross-combinations, shown in Fig. 6.17. The more discriminating effect of the *UM Translog* problems becomes again observable, for instance, in a more balanced occurrence of characteristics dominance relations between the subjects (cf. Fig. 6.4). Since we also have more reliable strategies as in the previous *Satellite* domain, there are more distinct stability values available.

Primary plan selection	Total	Undom.	e	s	r	%	EC	RC
$f^{planSel}_{First}$	3	3	3	–	–	67	–	–
$f^{planSel}_{FewerModBHS}$	10	7	8	3	–	60	3	3
$f^{planSel}_{\mathbb{F}/TE}$	10	8	9	1	–	60	1	1
$f^{planSel}_{FewerHZones}$	14	11	9	–	–	48	–	–
$f^{planSel}_{ConstrPlans^{-1}}$	2	–	2	–	–	33	–	–
$f^{planSel}_{Addr-\mathbb{F}_{\texttt{AbstrTask}}{}^{-1}}$	1	–	–	1	–	33	–	1
$f^{planSel}_{\text{PSA+OCA}}$	11	4	4	–	–	24	–	–
$f^{planSel}_{IndUniHS}$	10	–	3	–	–	10	–	–
$f^{planSel}_{LeastHZone}$	10	–	2	–	–	7	–	–
$f^{planSel}_{\text{CL+OCA}}$	10	–	2	–	–	7	–	–
$f^{planSel}_{DirUniHS}$	10	–	1	–	–	3	–	–
$f^{planSel}_{ConstrPlans}$	2	–	–	–	–	0	–	–
Total:	**93**	**33**	**43**	**5**		**29**	**4**	**5**

Table 6.10: Primary plan selections in the *UM Translog* domain: Number of undominated system configuration instances according to (e)fficiency, (s)tability, (r)eliability, efficiency-centered performance (EC), and reliability-centered performance (RC). % stands for the ratio of possibly to factually undominated.

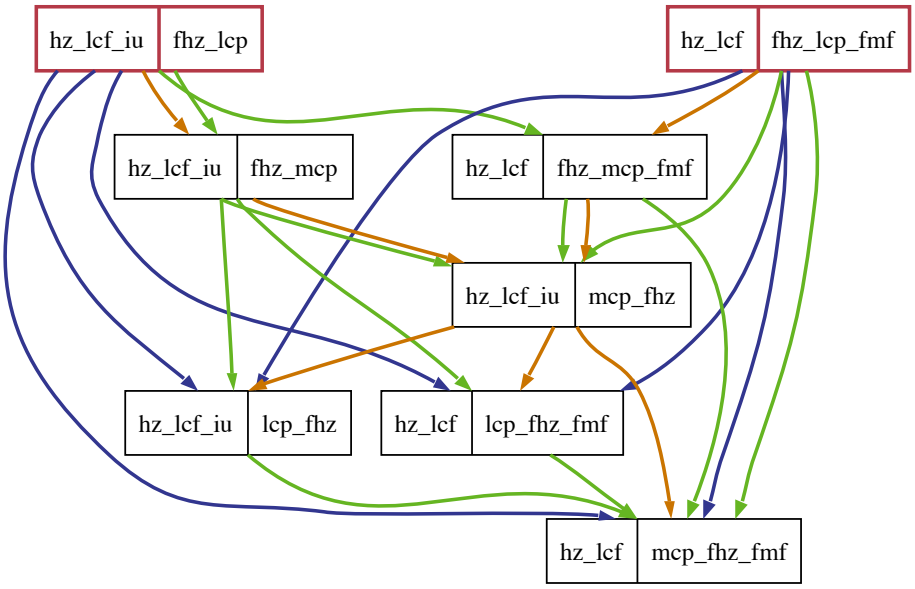


Figure 6.17: A detailed analysis of performance characteristics of related strategy combinations in the *UM Translog* domain. The edges represent a dominance relationship with respect to efficiency (**orange**), stability (**green**), and reliability (**blue**). **Red** boxes denote undominated strategies.

	ModSelection									
PlanSelection	**ems_lcf**	**hz_lcf**	**lcf_da**	**lcf_du**	**lcf_ems**	**lcf_hz**	**lcf_ia**	**lcf_iu**	**lcf_mod**	**lcf_pExp**
cloca	84	10	10	14	17	14	10	14	19	18
du_fmf	84	22	22	22	34	22	22	22	34	33
fhz_fmf	13	8	5	5	5	5	5	5	5	5
fmh_fmf	55	11	1	6	4	6	1	3	1	4
iu_fmf	84	22	22	22	34	22	22	22	34	33
lhz_fmf	42	22	22	22	34	19	22	22	35	34
psaoca	49	10	5	7	5	4	2	7	5	5
sdr_fmf	1	5	5	9	9	9	9	9	14	9

EMS	25		
UMCP	38	**UMCP+**	1
SHOP	24	**SHOP+**	10

Table 6.11: A strategy-component matrix of the ranks of reliability values in the *UM Translog* domain (**blue** = within 1^{st} quartile, **beige** = within median, **red** = beyond 3^{rd} quartile).

	ModSelection									
PlanSelection	**ems_lcf**	**hz_lcf**	**lcf_da**	**lcf_du**	**lcf_ems**	**lcf_hz**	**lcf_ia**	**lcf_iu**	**lcf_mod**	**lcf_pExp**
cloca	84	57	43	51	71	50	43	40	38	68
du_fmf	84	58	47	52	68	51	48	49	31	73
fhz_fmf	62	28	23	37	59	31	22	23	25	58
fmh_fmf	72	44	34	38	48	36	19	23	10	51
iu_fmf	84	59	48	46	70	46	42	46	29	74
lhz_fmf	80	57	46	53	71	51	46	44	35	78
psaoca	78	51	33	47	60	43	25	41	33	63
sdr_fmf	43	28	31	31	58	28	25	25	23	47

EMS	52		
UMCP	55	**UMCP+**	45
SHOP	59	**SHOP+**	65

Table 6.12: A strategy-component matrix for the ranks of stability values in the *UM Translog* domain (**blue** = within 1^{st} quartile, **beige** = within median, **red** = beyond 3^{rd} quartile).

It is worth noting that the induced hierarchy does not include any contradictory dominance result and that it exhibits two clearly dominating strategies (built from the same plan selection prefix) and one clearly identifiable sink that is to be regarded as the inferior combination.

Our first observation is that the combinations with a HotZone plan selection do not only dominate their rotated competitors but in practically all characteristics every strategy that deploys $f^{planSel}_{FewerHZones}$ as a secondary component. Regarding the question whether it is more suitable to prefer constrained plans or to avoid them, our findings in the *UM Translog* domain are in favour of the analogous result in the *Satellite* setting: preferring uncommitted plan structures for further development seems to be the more promising method.

Let us examine the TankerTruck problem as an exemplary plan generation objective for the `mcp`/`lcp` strategies ($f^{planSel}_{ConstrPlans}$ and its inversion); a solution plan is shown in Fig. 6.18. As far as the "inter-" and "intra-method" issue is concerned, the depicted causal structure is essentially representative for all *UM Translog* problems: The transportation task is composed of a loading operation, followed by a transporter movement, and finally an unloading procedure; some specific goods require an additional treatment. The coloured task cliques as they are described in the figure's caption represent the respective combinatorial choice with respect to the suitable decomposition methods. The black causal links, which have to be synthesized by the planning process, are the second dimension that influences the search effort. What makes these kinds of problems considerable easier than the *Satellite* instances is mainly the fact, that the decomposition methods operate along a deeply structured sort hierarchy such that the inter-method dependencies are typically resolved immediately after an expansion step due to variable constraints. For example, as soon as the system commits to the (green) method of unloading a liquid transport, the only consistent expansion for the (red) loading procedure is exactly that of the matching liquid transport loading – the parameter sorts in alternative

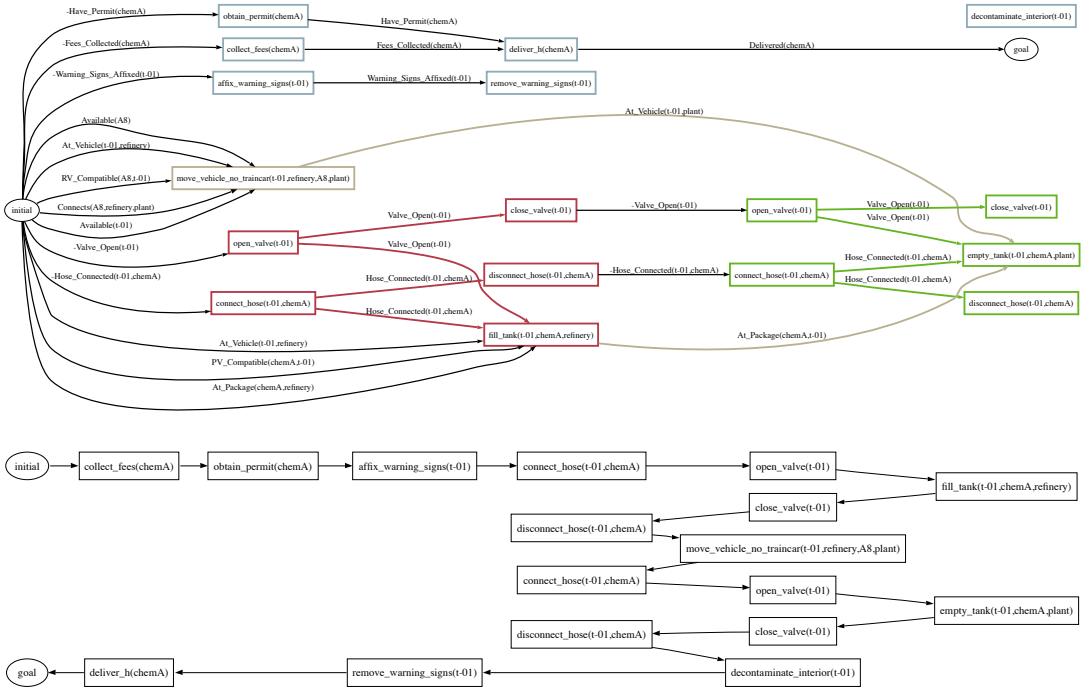


Figure 6.18: The causal structure (top) of a solution to the "TankerTruck" problem in the *UM Translog* domain and the induced temporal structure (bottom). The coloured task nodes and causal links indicate structures that are added to the plan in the context of an abstract task's expansion: specific **handling of chemicals**, **loading** and **unloading** operations, and a **transport implementation**.

methods induce an inconsistency. This constellation of constraints works significantly more effective than solely relying on maintaining a consistent causality.

When we examine the search space that is built by the least constrained plan preference strategy for the TankerTruck problem, displayed in Fig. 6.19, we first look at the key indicators: the `lcp` strategy visited 238 nodes with an average branching factor of 2,4 (branching ranges from 1 to 9 nodes) and a depth of the tree of 32 (in which also the solution found). The ratio of open nodes is 8% and that of discarded plans 50%. The figure also shows three exmplary fringes A, B, and C, which exhibit the breadth-first style of plan development. We can observe the same pattern of symmetry as we did for *Satellite*, which causes $f^{planSel}_{ConstrPlans^{-1}}$ to behave this way: It is evidently a characteristic for the *UM Translog* problem, as we have argued in the above paragraph, that task decomposition is typically not ambiguous. For example, we can see that just above fringe A two complex task flaws are subsequently resolved by 9 decomposition each, but we can also see that in both cases 8 of them are instantly discarded. That means, that the planning system merely explores variations of basically the same implementations, which will naturally have (sooner or later) the same ratio of constraints to plan steps. But this decomposition exclusiveness also explains the high percentage of cut options in the plan space and consequently the moderate branching factor (having in mind that it is only twice as high as in the considerably less complex *Satellite* domain). This confirms the impression we got from the reliability analysis, namely that *UM Translog* problems are much "simpler" for hybrid planning configurations than they appear regarding the overall complexity of the domain model. For comparison, most *Satellite* complex task flaws can be addressed by two refinements that appear consistent, see for example Fig. 6.6.

Fig. 6.20 visualizes the plan space as it is generated by the antagonistic "most constrained plan" strategy $f^{planSel}_{ConstrPlans}$ and, as the above arguments have predicted, it is traversed in a depth-first manner. The search tree is basically the same as for the `lcp` case, because the open nodes from the breadth-first search schema are apparently very close to their final refinement possibilities, in all cases unresolvable flaw situations, though. The numbers are consequently similar, although the explored space includes 385 nodes (which is 60% larger

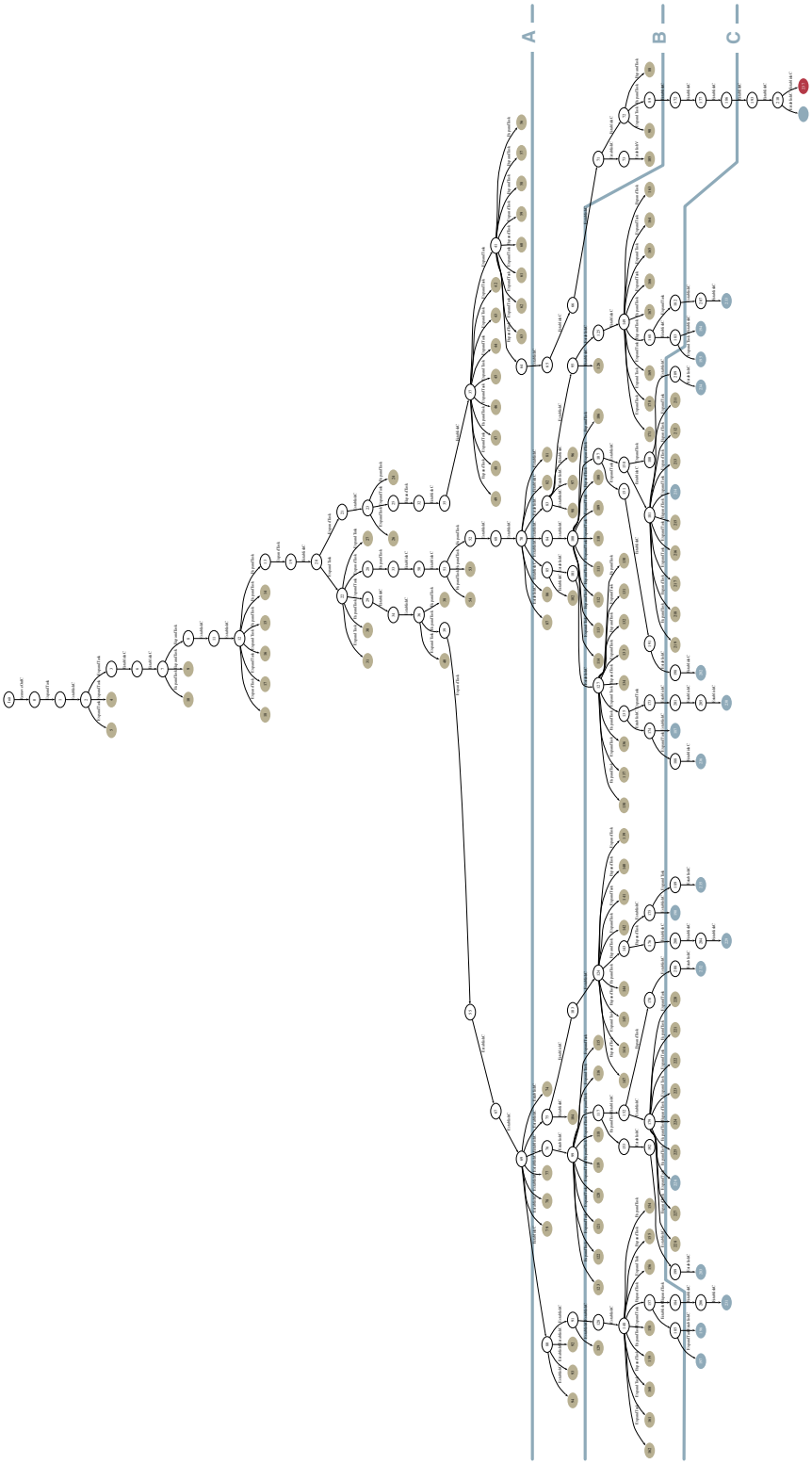


Figure 6.19: The spanned plan space by the modification selection $f^{modSel}_{HZone} \triangleright f^{modSel}_{LCF} \triangleright f^{modSel}_{IndUniHS}$ and plan selection $f^{planSel}_{ConstrPlans^{-1}} \triangleright f^{planSel}_{FewerHZones}$ with a hybrid planning system configuration on the TankerTruck problem in the *UM Translog* domain (**red** = solution, **blue** = open node, **beige** = discarded node).

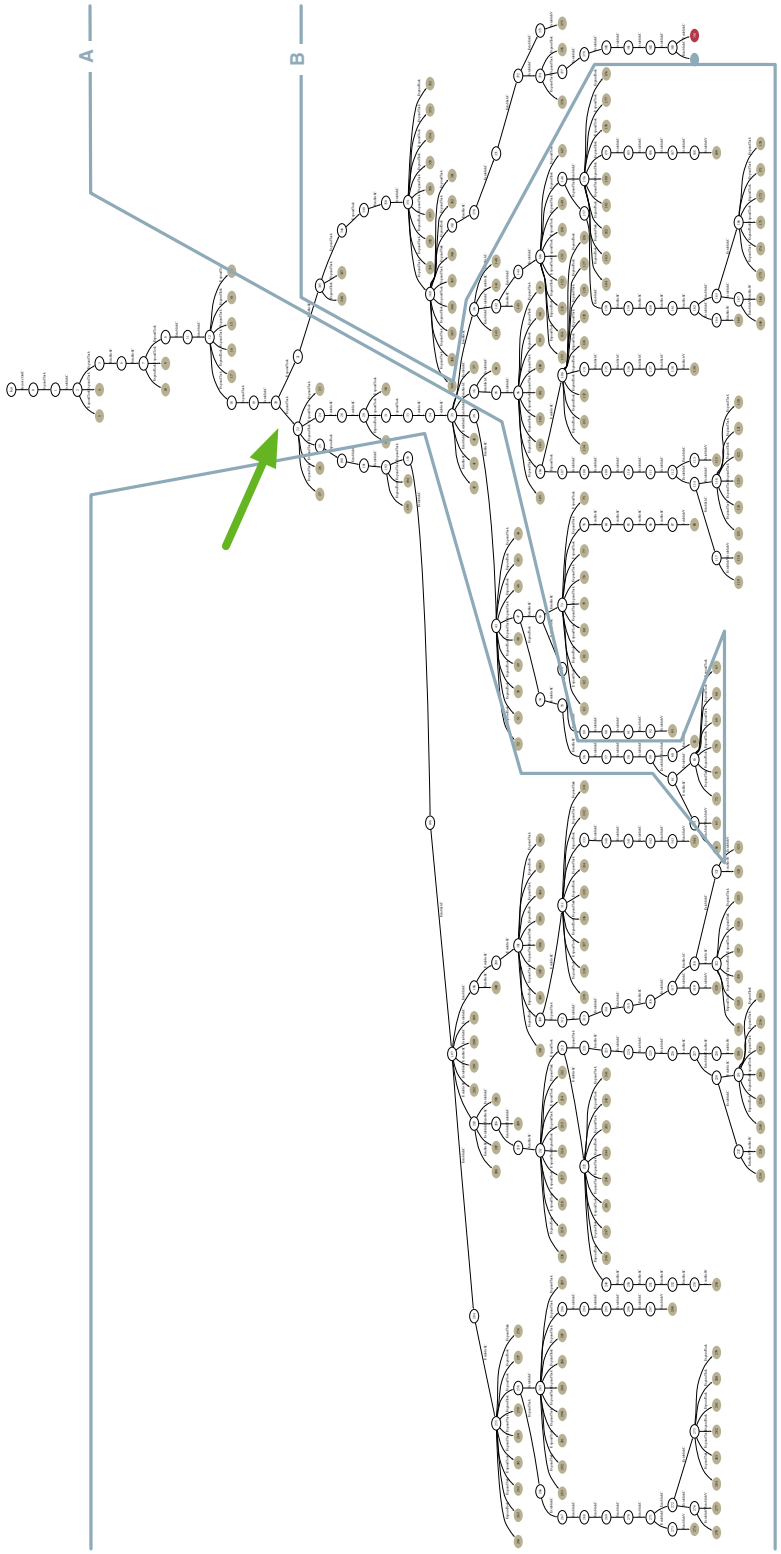


Figure 6.20: The spanned plan space by the modification selection $f^{modSel}_{HZone} \triangleright f^{modSel}_{LCF} \triangleright f^{modSel}_{IndUniHS}$ and plan selection $f^{planSel}_{ConstrPlans} \triangleright f^{planSel}_{FewerHZones}$ with a hybrid planning system configuration on the TankerTruck problem in the *UM Translog* domain (**red** = solution, **blue** = open node, **beige** = discarded node).

than before) and is slightly deeper (35 nodes): the branching factor ranges from 1 to 9 with an average of 2,3; there are naturally only few open nodes (approx. 0,3% if all nodes) but with a 55% fraction of discarded plans the eventual shape of the tree is not substantially different. It is however obvious from the figure, that the TankerTruck problem is already beyond the point where the `mcp` plan development can be competitive to its inversion. The depicted fringes A and B are snapshots from the early and late planning phases, respectively, which show that a very critical decision lies close to the initial node (green arrow at node 20). The two alternatives at this decision point are two decomposition methods for an abstract movement, one that assumes that the transport vehicle is at the customer site and one that includes a prior additional movement to it. The latter does not constitute a solution to the problem, however the "superfluous" additional task is tightly embedded by causal relationships, which makes it systematically attractive for this heuristic. Please note that the same problematic decision is observable for the corresponding *Satellite* example in Fig. 6.7 (node 9, becomes selected not until the D fringe). The $f^{planSel}_{ConstrPlans}$ strategy is therefore biased towards a specific kind of *UM Translog* problems and will become decreasingly useful the more of the un-addressed sub-problems occur. Aware of this detail, we have to specify the above result of the least constrained plan heuristic performing a breadth-first search: it is in fact slightly preferring the solution-path side of the search tree, the one without the superfluous task, however completing causality makes the strategy loose its focus and return to the "wrong" nodes, thereby processing the fringe layer-wise.

Let us finally compare these search space findings with the top performing reference strategy combination of $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ and $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ (see Fig. 6.21). While the least commitment modification selection is responsible for the lean entry path, it is the modification HotSpot-aware plan selection that defers the further development of nodes 19 (see green arrow), 62 (a member of fringe B), and 87 (is processed together with solution node). A particularly remarkable decision is the early "critical node" 19, which contributes most to the strategy's efficiency and reliability; it is the same decision at which the `mcp` strategy fails. The numerous interactions between the flaw resolution proposals for the superfluous tasks and those for the rest of the plan make this branch of plan development unattractive for the HotSpot strategy. Although we suggest further experimentation, in particular with increasingly complex maps, we are confident that this positive effect of the heuristic will continue to pay off for the current hybrid configurations. This is mainly because of the "causally dense" method design in *UM Translog*, which we have discussed above and in which missing properties are typically earlier detected than superfluous ones. For the sake of completeness, we note that this plan space consists of only 88 nodes with a maximum depth of 32, which is probably[12] optimal. The average branching factor is 2,4 (ranging from 1 to 9), 3% of the nodes are regarded as open, and 56% are discarded refinements.

Returning to Fig. 6.17, the analysis identifies further strategy relationships: Even clearer than in the corresponding *Satellite* analysis, the *UM Translog* domain favors strategies that deploy a $f^{planSel}_{FewerHZones}$ as a primary plan selection rather than as a second one. All combinations with $f^{planSel}_{FewerHZones} \triangleright f^{planSel}_{ConstrPlans^{-1}}$ and $f^{planSel}_{FewerHZones} \triangleright f^{planSel}_{ConstrPlans}$ dominate the switched variants in all characteristics practically completely.

Another result from the *Satellite* domain can be confirmed, but this time a negative one: the addition of a third plan selection, namely the preference for plans with fewer modification options $f^{planSel}_{Fewer-\mathbb{M}}$, does not stabilize the respective strategy combinations. Since in all but one cases there is no dominance result between such strategies with a binary plan selection and its extension, we repeat our previous conjecture, hoping that more experiments with larger fringes may give more insight on the influence of a third heuristic in these cases.

The question whether CL+OCA or its specialisation PSA+OCA is the more suitable A^* heuristic for hybrid planning, will be answered in the section about classical strategies.

Concerning the methodologically closely related HotZone strategy components $f^{planSel}_{FewerHZones}$ and $f^{planSel}_{LeastHZone}$, the results in the *UM Translog* domain are more coherent than in the *Satellite* domain. We find the preference of fewer HotZone clusters consistently superior to that of preferring a lower maximum of HotZone

[12] Although breadth-first search constructs the plan space up to the same depth and is theoretically guaranteed to find a solution path of minimal length, the deployed modification selection provides a different set of refinement options for `lcp` and this HotSpot strategy. This influences also the cut of "unreachable" modifications (cf. p. 172) and leads, for instance, to the linear path between nodes 4 and 10. The corresponding segments in the trees of the `lcp` and `mcp` combinations address other flaws, each induces three alternative modifications.

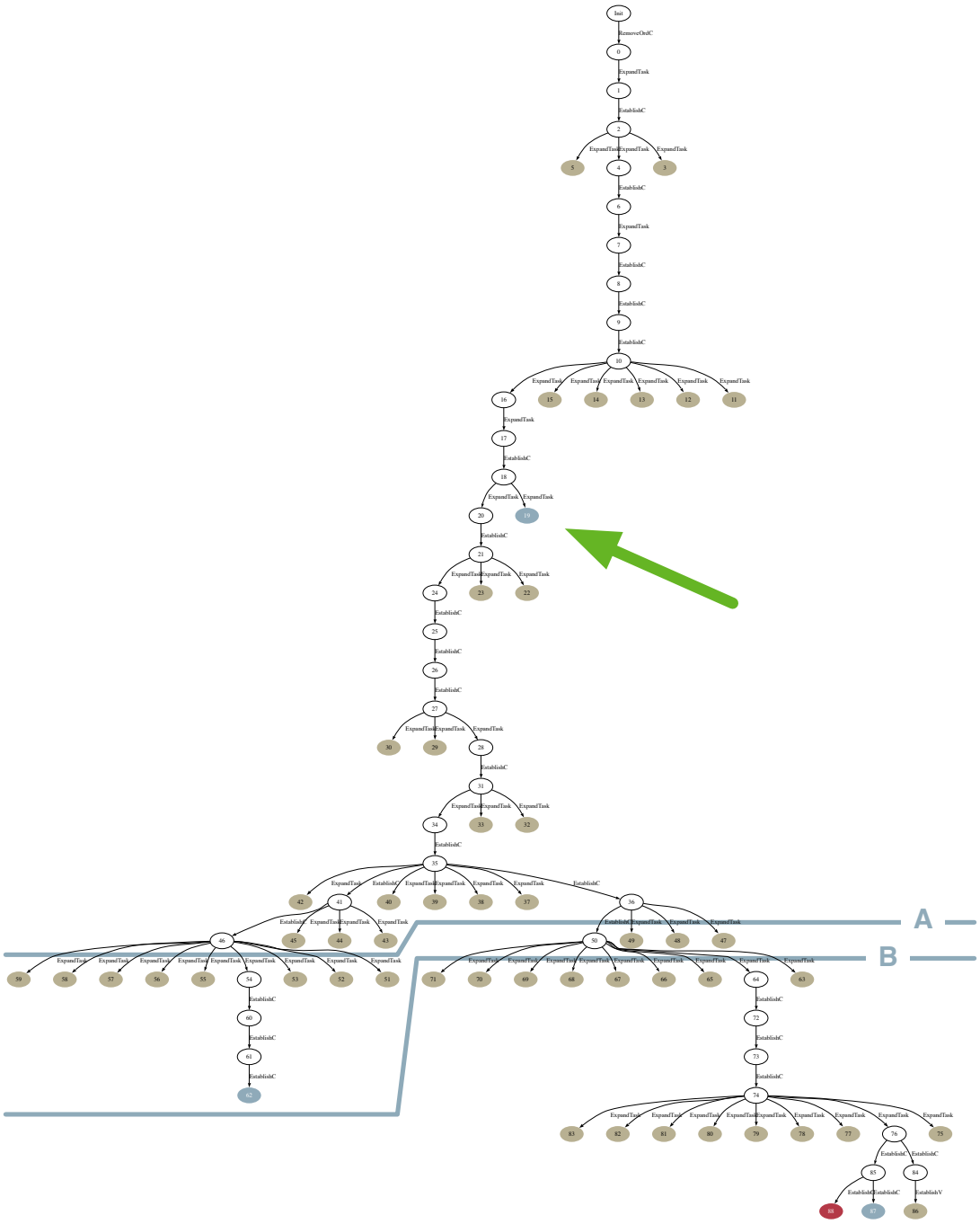


Figure 6.21: The spanned plan space by the modification selection $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ and plan selection $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ with a hybrid planning system configuration on the TankerTruck problem in the *UM Translog* domain (**red** = solution, **blue** = open node, **beige** = discarded node).

values. Strategy combinations with a primary FewerHZones plan selections are members of the most efficient strategies (10% of the candidates in Fig. 6.14) while there are no LeastHZone combinations. But also a direct comparison gives a clear-cut result: FewerHZones individuals dominate their relatives almost completely, namely in 24% of all possible cases with respect to stability and in even 84% and 91% in terms of efficiency and reliability, respectively. Further experimentation will give us information about the reason for the performance discrepancies between those two strategies in both domains. In particular, we have to clarify if this can be attributed to the presence of recurring sub-problem patterns, for example, in analogy to the multiple observation scenarios, by introducing additional (parallel) transportation jobs with additional transportation means that allow an optional sharing of resources.

The evaluation of our modifications of classical strategies, `shop+` and `umcp+`, confirmed some of the results that we obtained in the *Satellite* domain: `shop+` is substantially less efficient and stable than the original *SHOP*, although it is found much more reliable than in the previous domain. Interestingly enough, the *UMCP* strategy benefits from the enhanced plan selection in the *UM Translog* domain, its former testbed scenario (see Sec. 5.2.2). While the unmodified *UMCP* is undominated in terms of efficiency (Fig. 6.14), the efficiency and stability yield of `umcp+` outperforms it on the absolute scale with an average ranking in the second quartile (Tab. 6.8 and 6.12). With respect to reliability, `shop+` is not only significantly superior to its ancestor, but even positioned among the top performing strategies (Fig. 6.16 and Tab. 6.11).

Let us now turn to the analysis of the HotSpot strategies. In the *Satellite* problems, the direct plan selection did produce better results than combinations deploying an indirect one. The corresponding *UM Translog* result is less counter-intuitive: the combinations with $f^{planSel}_{IndUniHS}$, the indirect uniform plan selection, are more efficient than direct combinations in 26 cases, the opposite holds for 23, and more stable in 15, less stable in 11. In terms of reliability, the domination is balanced with 31 for each. That means, that the indirect HotSpot plan preference made up for the direct one and hast in the logistics scenario a slight advantage. We may hypothesize that this is a result of the *UM Translog* plans containing more plan elements than in the *Satellite* problems before. It appears plausible that the more structures are present, the less noisy the finer granular calculations become and the more they are able to differentiate plans. However, we are also aware of the tendency that both types of strategies play a minor role in the *UM Translog* domain and are even not among the most performing individuals in any characteristic. Future experiments will have to investigate whether there are application perspectives for this kind of strategy or they are conceptually subsumed by the HotZone method and may be deployed only as subsequent decision support.

The situation with respect to HotSpot modification selections is a completely different one. The indirect adaptive and indirect uniform HotSpot functions are the most successful modification selection methods in terms of efficiency as well as reliability (see Tab. 6.8 and 6.11). But also the direct versions do perform very well, and therefore we are now examining their relationships more closely. Fig. 6.22 gives an overview over the dominance findings within this subset of evaluation individuals. The principle outcome is that the direct uniform selection is dominated by every other variant over all characteristics. Concerning the general comparison of uniform versus adaptive or direct versus indirect, the numbers are not decisive: while the adaptive strategies dominate their uniform counterparts, a general relation between direct and indirect cannot be deduced from the individual characteristics. It is remarkable that adaptivity positively contributes in terms of reliability (two adaptive modification selections among the most reliable) and stability, even if this contribution is in some cases a small one. Since these benefits have apparently not been gained at the cost of efficiency (some adaptive candidates among the most efficient), we assume that the power of the adaptive selections will emerge more prominently in larger problem constellations that enable the indirect calculations and in more sophisticated weight and weight-adaption assignments. We are furthermore confident that the positive evaluation of adaptive techniques in the *Satellite* and now in the *UM Translog* domain can be confirmed in future experiments as well.

Compatibility between the HotSpot modification and selection functions is evident in the evaluation matrices, in particular regarding efficiency (Tab. 6.8 to 6.12). Since the respective plan selections are far behind most of their competitors, this is however not of too much relevance. More interesting is the combination of similar HotZone strategies: deploying f^{modSel}_{HZone} as the primary modification selection together with $f^{planSel}_{LeastHZone}$ or $f^{planSel}_{FewerHZones}$ constitutes the most successful instances of the respective plan selections; in the latter case the synergy is significantly higher and even results in a most effective strategy. It is also worth noting that apparently the indirect HotSpot modification selections are very suitable combination partners for HotZone

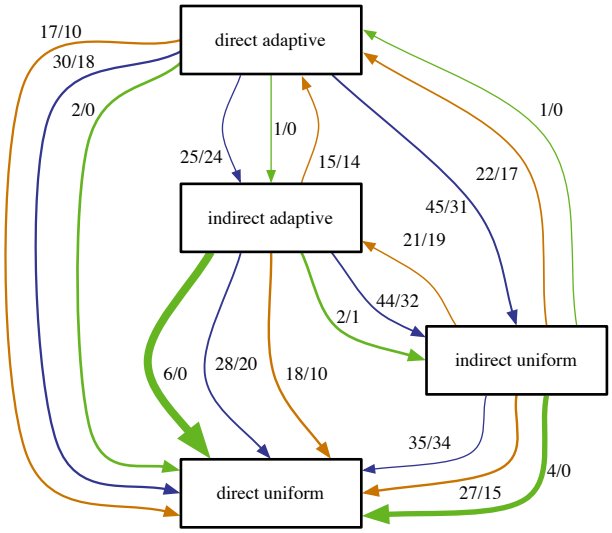


Figure 6.22: Quantitative dominance comparison between direct, indirect, adaptive, and uniform modification selection in the *UM Translog* domain. Edge labels denote number of dominant instances in / contrary to edge direction; edge width corresponds to dominance ratio (**orange** = efficiency, **green** = stability, **blue** = reliability).

	Efficiency		**Stability**		**Reliability**	
Strategy	dom.	dom. by	dom.	dom. by	dom.	dom. by
CL+OCA*	13	30	6	12	32	37
PSA+OCA**	23	16	7	10	52	16
UMCP	6	0	3	0	4	7
SHOP	11	0	3	0	13	9
EMS	11	0	3	0	9	6

* Average dominance values for 10 combinations.

** Average dominance values for 11 combinations.

Table 6.13: The dominance situation for the classical strategy implementations in the *UM Translog* domain.

plan selection functions. This result matches the conceptual design of the functions such that addressed HotSpots shape the HotZones of the subsequent refinement plans.

Analysis of Classical Strategies

The leading questions for this empirical study explicitly asked for the performance of literature's classical strategies. Concerning efficiency, the candidates from hierarchical planning are well positioned in the field of competitors in the *UM Translog* domain: *UMCP*, *SHOP*, and *EMS* are all members of the most efficient strategies (Fig. 6.14). The derivates from Partial-Order Planning, CL+OCA and PSA+OCA, are less prominent and only the latter seems to be efficient and very reliable here. The concrete dominance situation is given in Tab. 6.13: *UMCP*, *SHOP*, and *EMS* are undominated in terms of efficiency and stability, confirming the excellent efficiency of the trio. However, their numbers of dominated strategies in *UM Translog* is significantly smaller over all characteristics than before in *Satellite*, although the general dominance incidence doubled. Their outstanding performance is therefore not based on being consistently more efficient or reliable than other strategies, but on being specialized in particular problem instances and therefore harder to be dominated by others. The experimental data reveals, for instance, that *EMS* holds rank 1 in the efficiency analysis of the RegularTruck-3Locations problem, but its average rank is 43. *UMCP*, average rank is 48, is

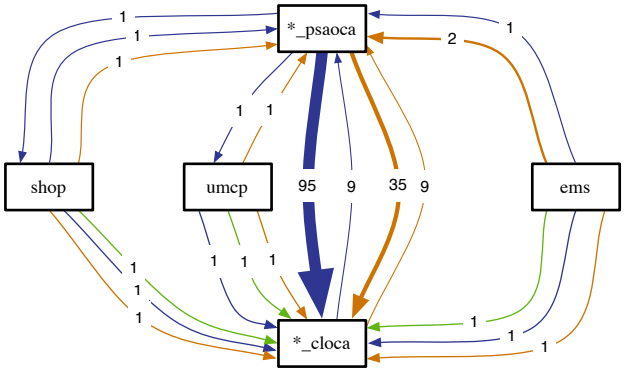


Figure 6.23: Quantitative dominance comparison between classical strategy combinations in the *UM Translog* domain. Edge labels denote number of dominant instances; edge width corresponds to the ratio of dominance declarations to the total number of combinatorial possibilities (**orange** = efficiency, **green** = stability, **blue** = reliability).

exceptionally efficient in the AutoTraincar problem, where it gains rank 5 while most other strategies fail completely. *SHOP* is an expert for the AutoTraincar and the 2-RegularTruck problems, but despite its 1st rank there, its average position is only the 34th (see also Tab. B.4 to B.6).

The classical CL+OCA strategy and its hybrid counterpart PSA+OCA have improved their standing compared to the *Satellite* domain results. Both strategy classes are more efficient, stable, and in particular more reliable than before, while at the same time reduced the number of situations in which they are dominated. In addition, it has to be taken into account that the inter-class dominances have been reduced drastically, and in these problems, only about 10% of the dominated strategies are from the same class.

In order to determine the adequacy of designing a strategy as to address expansion eagerly versus cautiously, Fig. 6.23 focuses on the dominance results within the subset of classical strategies. As we can see, there is no dominance finding between *UMCP*, *SHOP*, and *EMS*; the question for which decomposition tactics to deploy in general cannot hence be answered yet for the *UM Translog* domain. The only evident finding is that PSA+OCA combinations are definitely more reliable and stable as CL+OCA strategies, a tendency that is confirmed by the prominent performance of the PSA+OCA plan selection – four combinations are among the most efficient strategies (Fig. 6.14) – and the respective rankings (tables 6.11 and 6.12).

An idea for an alternative interpretation of the results in Fig.6.23 may also be the following: Since *UMCP*, *SHOP*, and *EMS* are so successful and in particular dominate CL+OCA and PSA+OCA (here only in few cases, but more in the *Satellite* domain) it is probably a value in itself to adopt "a" attitude towards the issue of task decomposition rather than regarding it as par inter pares. But on the whole, we feel that upon the current data a definite judgement for or against early threat detection cannot be reached.

Together with the previous results from the *Satellite* domain we suggest further experimentation on this matter, although we believe that our findings so far already indicate that in spite of the success of the classical strategies our flexible planning strategy with their opportunistic modus operandi will become widely accepted.

Problem Analysis

Following our leading questions (Def. 6.6), we now turn from interpreting the strategies to interpreting the characteristics of the problem instances. Analogously to the strategy performance, we present diagrams that show the mapping of strategy results on *UM Translog* problems in the diagrams Fig. 6.24 (efficiency difficulty), 6.25 (stability difficulty), and 6.26 (reliability difficulty). Recalling the problem descriptions from the experimental setup section (p. 224), there are only few expectations with respect to emerging

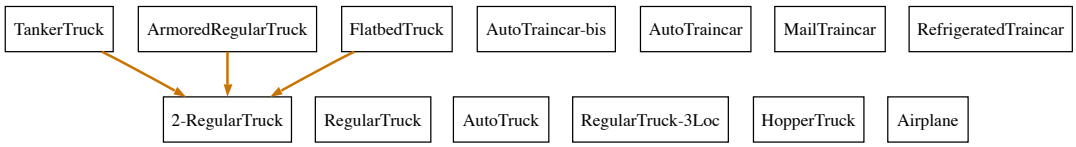


Figure 6.24: Difficulty of the problems in the *UM Translog* domain in terms of efficiency.

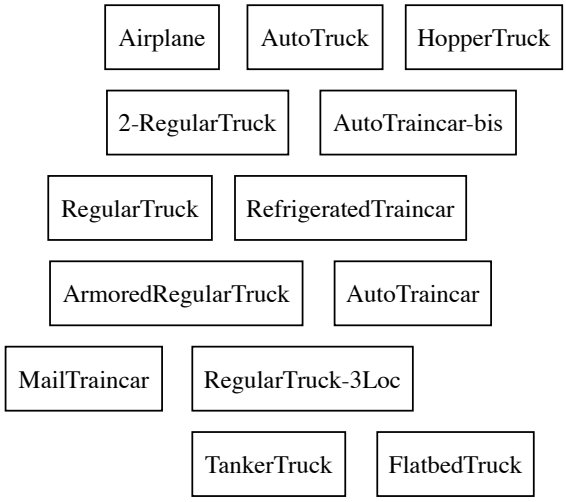


Figure 6.25: Difficulty of the problems in the *UM Translog* domain in terms of stability.

difficulty levels except the variations of a regular truck transport in which an additional package and a slightly more complex traveling route are involved.

Efficiency difficulty is only weakly developed in the *UM Translog* problems. The double transport problem 2-RegularTruck was expected to be more difficult than the singular ones, but obviously, there were only the three shown problem that could be solved more efficiently by the evaluation candidates. To our disappointment, the "trivial" result that the double transport or the more complex one takes consistently longer to be planned than the single regular one cannot be found. Another unintuitive outcome is missing, too: the AutoTraincar-bis problem describes an automobile transport by train (the AutoTraincar problem) in which the transportation means are initially available at the customer's place. The additional effort that is necessary to bring the locomotive to the transport's origin is not reflected by a reduced efficiency, that means by AutoTraincar-bis dominating AutoTraincar in terms of **efficiency**. In summary, if a solution to one of the *UM Translog* problems is found, the search effort is not consistently higher for most of the instances.

Difficulty in terms of **stability** is statistically the same for all posed problems. Since there is no problem inducing a higher sample variance on all strategies, our *UM Translog* problem portfolio cannot contribute to the corresponding discussion of the *Satellite* domain. We encourage our hypothesis that stability may not be a very useful problem characteristic on such an inhomogeneous set of strategy candidates.

A completely different situation is encountered in the **reliability** analysis of the *UM Translog* problems as shown in Fig. 6.26; its dominance relationships induce fine granular difficulty levels. We can see that it is considerably more difficult to reliably produce plans for two transportation tasks than for a single (which is trivial) and also that is more difficult to do so than for a slightly more complex routing sub-problem. In this sense, the above mentioned plausibility issues are addressed. We also get from this analysis the information that the intuitively more complex expansions, that means, the liquid loading procedure for tankers and the transportation means availability, do not introduce a significant overhead for plan generation. The only consistently conceivable overhead that is apparently relevant is the problem of the *indirect* navigation of

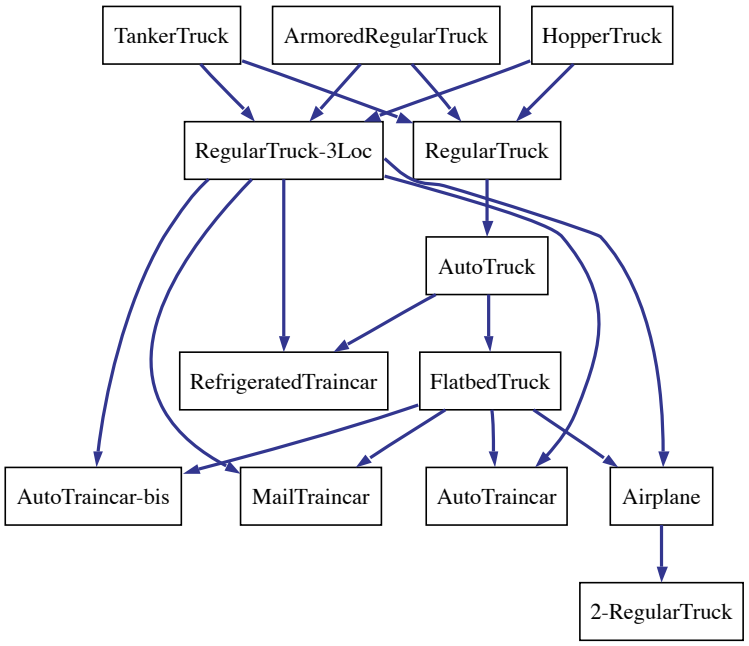


Figure 6.26: Difficulty of the problems in the *UM Translog* domain in terms of reliability.

trains: while it is the railroad car that actually transports the good (and is bound by the parameters of the respective tasks) it is however the locomotive that is to be moved. This obviously induces enough combinatorial sub-problems that many strategies fail on these problems instances (cf. complete presentation of average failure rates in Tab. B.6).

As we stated before, the number of involved plan steps is not an issue in hybrid planning; the *UM Translog* model puts this more precise, namely that the key factors are a tight pre-defined causality in the expansion networks and a strong dependency among the decomposition methods. Since the evaluated strategy portfolio is on the whole capable of handling the posed problem instances – at a moderate failure rate of 23% on average – we are confident that future experiments will show how the above findings can be applied to scenarios with multiple jobs, joint and exclusive transportation requirements, and more realistic traffic environments.

6.5 Evaluation Results in the *CrissCross* Domain

Efficiency-Centered Performance Evaluation

When we introduced the *CrissCross* domain in Sec. 5.2.3 we described its design as a test case for the strategies that perform HotSpot and HotZone calculations. But while *Satellite* and *UM Translog* proved to be appropriate domains for these strategies in terms of efficiency, the *CrissCross* domain is apparently most efficiently addressed by others.

As we can see, there is a large number of finalists in the efficiency-centered performance evaluation (Fig. 6.27). The subsequent stability and reliability analyses however produce substantial additional dominance results, which reduce the candidate set to the following three top performing strategies:

- $f^{modSel}_{LCF} \triangleright f^{modSel}_{EMS}$ with $f^{planSel}_{\mathbb{F}/TE} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$
- $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ with $f^{planSel}_{\mathbb{F}/TE} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$
- $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ with $f^{planSel}_{\text{PSA+OCA}}$

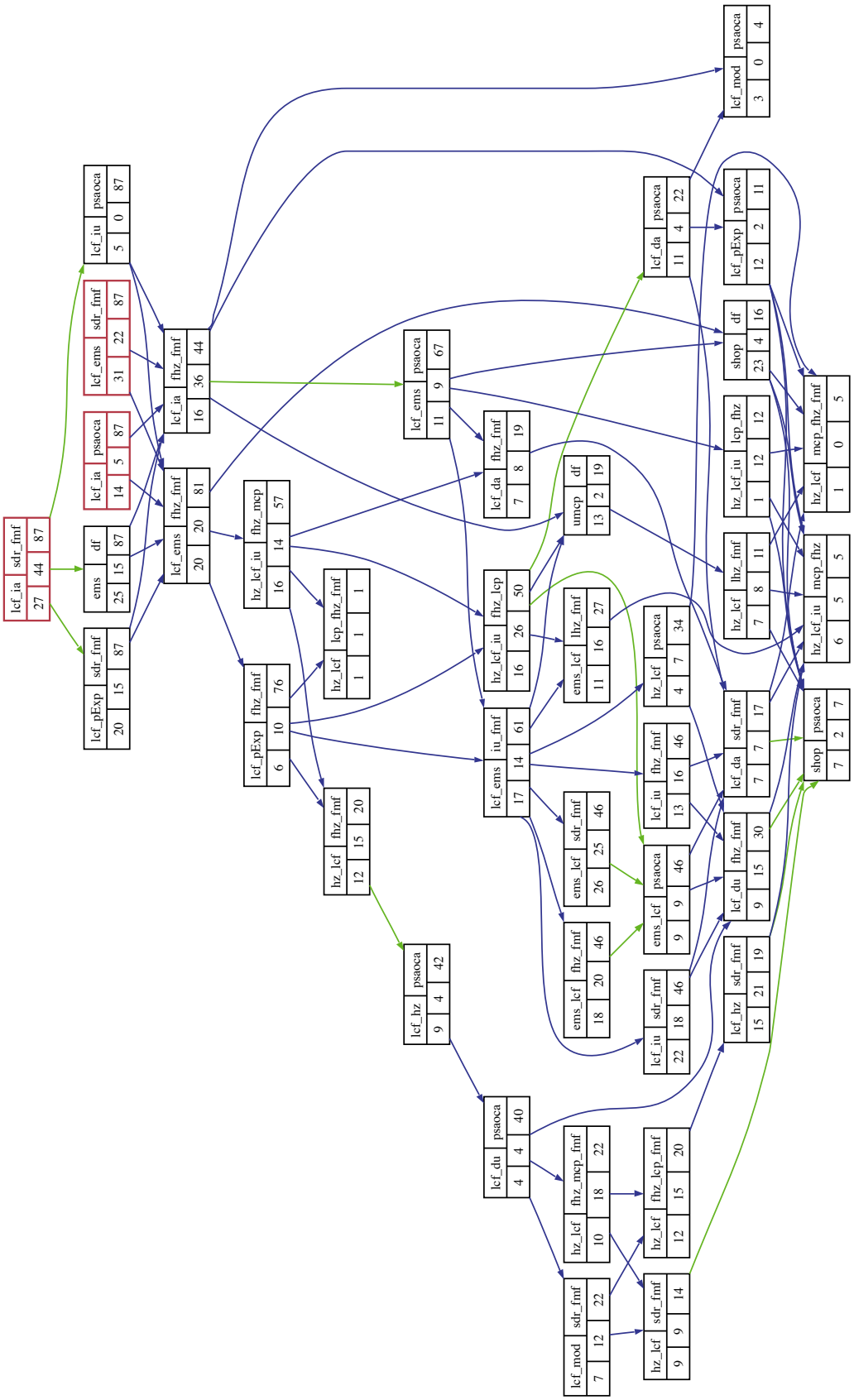


Figure 6.27: The efficiency-centered performance evaluation in the *CrissCross* domain. Nodes are the most efficient strategies, edges represent stability (**green**) and reliability (**blue**) dominance. Node annotation shows number of strategy combinations dominated by the node in terms of efficiency, stability, and reliability (**red** = undominated strategy).

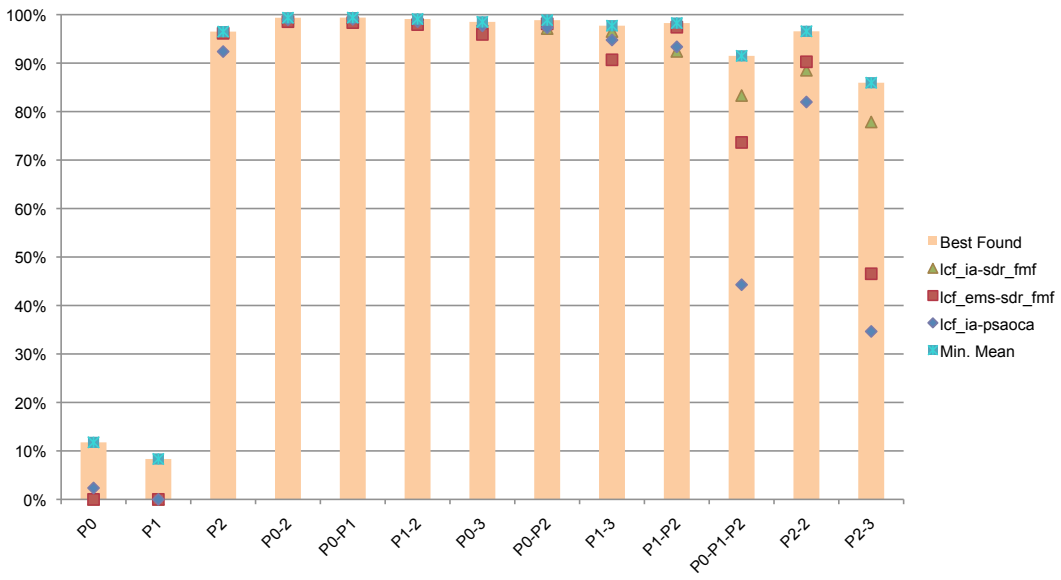


Figure 6.28: The *relative* search efficiency of the undominated strategies in the *CrissCross* domain. 100% on the vertical axis corresponds to a "perfect" strategy that solves the problem immediately, 0% represents the worst efficiency value in the experiments.

The winning combinations do not occur in the previous evaluations of the other planning domains. But we find again strategies that appear to perform rather poorly like $f^{modSel}_{HZone} \triangleright f^{modSel}_{LCF}$ with $f^{planSel}_{ConstrPlans^{-1}} \triangleright f^{planSel}_{FewerHZones} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$, which only dominates one single competitor in every category.

With 42 efficiency-undominated strategies, Fig. 6.27 contains by far the largest candidate set in our empirical study. This raises of course the question whether all these individuals, and in particular the above winning strategies, perform exceptionally good or the remaining 51 so bad or all together so diverse that no dominance can be constituted? The answer lies somewhere in between: On the one hand, the data shows that the strategies' performance ranges considerably from problem to problem (cf. efficiency data table B.7) and this fluctuation is certainly a reason for the size of the candidate set. On the other hand, the winning strategies perform very well as can be seen in Fig. 6.28, which puts the efficiency values into perspective of the complete spectrum of results. P0 and P1 are not representable in a meaningful way by the figure: the difference between the best known result and the worst case is hardly visible and the relative performance gain is therefore minimal. For the other problems, we can see that they appear to be addressed quite well by the evaluation winners, not as well as in the *UM Translog* domain, but better as in *Satellite*. This is also additional evidence for the choice of analysis method, because the dominance "cascade" we applied obviously selects (as a by-product) relatively competitive strategies. We would like to point out, however, that this relative measure will have to be re-evaluated in more advanced candidate sets of future experimentation in order to get a better scale resolution. At the moment, all reasonably good strategies are densely clustered due to the inclusion of extremely bad performing strategies that induce a wide range on the relative scale. Instead of the worst value, we may also use in the future a more performing statistical entity as point of reference, for example, the first quartile.

Let us now look into the contributions of the strategy components, divided into modification and plan selections in the matrix in Tab. 6.14. In contrast to our previous findings, the *CrissCross* problems do only induce a more or less clear performance pattern on the plan selection functions and not on the modification selections. A clearly positive association is given for the components $f^{planSel}_{FewerHZones} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ (`fhz_fmf`) and $f^{planSel}_{\mathbb{F}/TE} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ (`sdr_fmf`); the latter leads as part of two combinations the field of candidates. These two are followed by the PSA+OCA method, which is involved in the third winning strategy. The plan selection components that have a negative influence on the efficiency of the strategy are the simple CL+OCA heuristic, the direct uniform HotSpot, and the preference of fewer modification-based HotSpots. Regarding the modification selection functions, we can only see a relatively weak modification-based HotSpot

PlanSelection \ ModSelection	ems_lcf	hz_lcf	lcf_da	lcf_du	lcf_ems	lcf_hz	lcf_ia	lcf_iu	lcf_mod	lcf_pExp
cloca	62	67	60	67	63	66	64	63	67	66
du_fmf	37	70	51	56	37	65	56	58	48	47
fhz_fmf	28	21	29	34	21	29	15	26	36	29
fmh_fmf	61	42	63	48	61	50	49	50	50	54
iu_fmf	32	68	49	58	28	63	55	59	45	43
lhz_fmf	37	39	52	54	43	53	49	50	44	43
psaoca	42	34	32	36	33	32	22	22	49	40
sdr_fmf	23	22	34	35	19	25	11	23	30	24

EMS	19		
UMCP	37	UMCP+	66
SHOP	25	SHOP+	53

Table 6.14: A strategy-component matrix of the average ranks of efficiency values in the *CrissCross* domain (**blue** = within 1^{st} quartile, **beige** = within median, **red** = beyond 3^{rd} quartile).

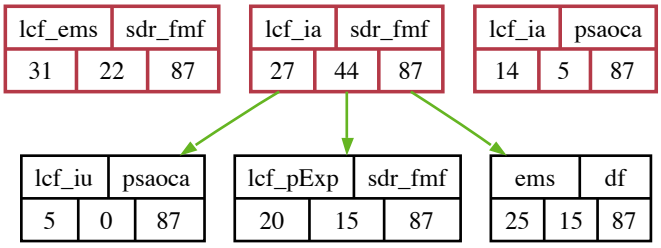


Figure 6.29: The reliability-centered performance evaluation in the *CrissCross* domain. Nodes are the most reliable strategies and (**green**) edges represent stability dominance. Node annotation shows number of strategy combinations dominated by the node in terms of efficiency, stability, and reliability (**red** = undominated strategy).

preference `lcf_mod` and the direct adaptive and uniform HotSpot selection. In general, no consistently performing modification selection can be identified. Consequently, there are no distinct crossing points as well.

It is not clear yet, why the *CrissCross* domain does not induce a recognizable efficiency pattern onto the modification selection functions. The data indicates that efficiency depends exclusively – in contrast to *primarily* – on the plan selection principle and that it is very fragile with respect to the modification selection function.

Reliability-Centered Performance Evaluation

Reliability has been the harder dominance criterion in the *Satellite* and *UM Tranlsog* domains, and so it is in *CrissCross*, too. The reliability-centered performance evaluation is conducted on a candidate set of only 6 strategy combinations and the graph representation of the findings is identical to the top-level layers in the efficiency-centered evaluation graph. As it can be seen in Fig. 6.29, the undominated strategies and winners of this evaluation schema are the same trio as for efficiency above:

- $f^{modSel}_{LCF} \triangleright f^{modSel}_{EMS}$ with $f^{planSel}_{\mathbb{F}/TE} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$
- $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ with $f^{planSel}_{\mathbb{F}/TE} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$
- $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ with $f^{planSel}_{\text{PSA+OCA}}$

In the first evaluation scenario, the *Satellite* domain, we noted that it is a mere coincidence if the candidate and/or winning strategy sets overlap for both types of evaluation. Obtaining this result for the third time however leads us to the assumption that *in general* an efficient strategy is at the same time a reliable one as

Strategy	Average Failure Rate
Best value	0%
$f^{modSel}_{LCF} \triangleright f^{modSel}_{EMS}$ with $f^{planSel}_{\mathbb{F}/TE} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$	0%
$f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ with $f^{planSel}_{\mathbb{F}/TE} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$	0%
$f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ with $f^{planSel}_{\text{PSA+OCA}}$	0%
1. Quartile	15%
Average value	31%
Median	31%
3. Quartile	46%
Worst value	82%

Table 6.15: Positioning of the most reliable strategies within the field of competitors in the *CrissCross* domain.

	ModSelection									
PlanSelection	**ems_lcf**	**hz_lcf**	**lcf_da**	**lcf_du**	**lcf_ems**	**lcf_hz**	**lcf_ia**	**lcf_iu**	**lcf_mod**	**lcf_pExp**
cloca	28	52	36	53	37	52	33	33	59	58
du_fmf	22	67	38	43	19	58	38	31	35	34
fhz_fmf	11	15	15	16	2	15	5	11	11	2
fmh_fmf	38	8	40	25	37	29	24	24	29	27
iu_fmf	13	59	45	41	5	51	28	31	29	28
lhz_fmf	19	36	25	37	28	34	36	36	28	30
psaoca	11	11	8	7	3	6	1	1	36	25
sdr_fmf	11	17	21	21	1	16	1	11	13	1

EMS	1		
UMCP	23	**UMCP+**	53
SHOP	20	**SHOP+**	48

Table 6.16: A strategy-component matrix of the ranks of reliability values in the *CrissCross* domain (**blue** = within 1^{st} quartile, **beige** = within median, **red** = beyond 3^{rd} quartile).

well, although it does not *necessarily* need to be. In particular, we are convinced that the logical negation does hold: An unreliable strategy is not an efficient one. We also conjecture that the reliability-centered performance evaluation is systematically much more selective than the efficiency-centered analysis, even for a larger number of planning episodes per configuration, which will lead to more diverse reliability values. In order to minimize the number of experiments that are necessary to identify suitable strategy combinations for a given domain, we therefore suggest to conduct exclusively the reliability-centered performance evaluation on the full set of candidates. All kinds of efficiency analysis can be performed on the reduced set of reliable strategies thereafter and we can be relatively sure that we do not miss a highly performative candidate.

Tab. 6.15 gives us the statistical characteristics that describe the candidate set in terms of reliability. Like for the *UM Translog* domain, the minimum failure rate lies at 0%, that means, the most reliable strategies never failed a run; the three winning strategies were also perfect ones. The maximum failure rate is much lower than in *UM-Translog*, the quartiles are however much larger values; the evaluation population has therefore more difficulties to solve the *CrissCross* problems than the *UM Translog* instances. On the other hand, the *Satellite* domain findings show that there are apparently even more complications, since the respective quartiles are even above the *CrissCross* values.

We conclude our reliability analysis with a compilation of the participating components in a matrix of ranks (Tab. 6.16). It apparently confirms the findings from the above efficiency analysis with only minor deviations: $f^{planSel}_{\text{PSA+OCA}}$, $f^{planSel}_{\mathbb{F}/TE} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ (`sdr_fmf`), and $f^{planSel}_{FewerHZones} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ (`fhz_fmf`) have a positive connotation, the direct and indirect HotSpot and the CL+OCA selection a negative one. Like it is the case for all our reliability findings, there is no clear trend with respect to the modification selection components.

Primary plan selection	Total	Undom.	e	s	r	%	EC	RC
$f^{planSel}_{ConstrPlans^{-1}}$	2	2	2	–		67	–	–
$f^{planSel}_{First}$	3	3	2	1		67	–	–
$f^{planSel}_{\mathbb{F}/TE}$	10	9	8	3		67	2	2
$f^{planSel}_{ConstrPlans}$	2	2	2	–		67	–	–
$f^{planSel}_{\text{PSA+OCA}}$	11	11	4	2		52	1	1
$f^{planSel}_{FewerHZones}$	14	12	10	–		52	–	–
$f^{planSel}_{LeastHZone}$	10	2	6	–		27	–	–
$f^{planSel}_{IndUniHS}$	10	1	4	–		17	–	–
$f^{planSel}_{FewerModBHS}$	10	–	4	–		13	–	–
$f^{planSel}_{DirUniHS}$	10	–	3	–		10	–	–
$f^{planSel}_{\text{CL+OCA}}$	10	–	–	–		0	–	–
$f^{planSel}_{Addr-\mathbb{F}_{\text{AbstrTask}}{}^{-1}}$	1	–	–	–		0	–	–
Total:	**93**	**42**	**45**	**6**		**33**	**3**	**3**

Table 6.17: Primary plan selections in the *CrissCross* domain: Number of undominated system configuration instances according to (e)fficiency, (s)tability, (r)eliability, efficiency-centered performance (EC), and reliability-centered performance (RC). % stands for the ratio of possibly to factually undominated.

	ModSelection									
PlanSelection	**ems_lcf**	**hz_lcf**	**lcf_da**	**lcf_du**	**lcf_ems**	**lcf_hz**	**lcf_ia**	**lcf_iu**	**lcf_mod**	**lcf_pExp**
cloca	59	71	58	72	67	76	75	75	78	71
du_fmf	38	67	41	44	38	46	49	51	46	48
fhz_fmf	25	19	26	30	20	27	10	20	31	29
fmh_fmf	51	42	57	42	46	47	48	49	47	45
iu_fmf	31	65	42	50	29	56	48	52	46	40
lhz_fmf	32	35	47	42	44	54	48	47	40	32
psaoca	49	38	42	41	37	47	27	38	60	43
sdr_fmf	21	27	33	37	21	19	8	25	26	27

EMS	24		
UMCP	44	**UMCP+**	78
SHOP	43	**SHOP+**	63

Table 6.18: A strategy-component matrix for the ranks of stability values in the *CrissCross* domain (**blue** = within 1^{st} quartile, **beige** = within median, **red** = beyond 3^{rd} quartile).

Analysis of Strategy Components

When we break down the quantitative undominance occurrences in the *CrissCross* domain according to the deployed primary plan selection, we obtain the data displayed in Tab. 6.17. In accordance with the above efficiency results, we find a higher total ratio of undominated strategies (33%) than in the other domains; we note that there is also a wider "gap" between those strategies below and above the threshold borderline. It is furthermore interesting to see that the increase of undominated strategies in relation to the previous results (cf. Tab. 6.4 and 6.10) is restricted to efficiency only. Apart from the extremely successful principle of preferring the smaller detection ratio ($\mathbb{F}/TE$), there are three more plan selection functions that have the maximum ratio of 67% undominated combination instances. These functions are apparently candidates with a high performance potential and further experiments should not forget to include them.

Before we move on to the specific strategy subsets, we would like to complete the general picture with the component-wise stability analysis. Tab. 6.18 does however not give us additional information, it only confirms that those positive and negative associations between performance and specific plan selections that we reported above are reflected in the stability property as well. With this result, we have no characteristic that can be uniquely attributed to one of our modification selections. There are in principle two explanations for this *CrissCross* particularity: First, our evaluation's modification selection repertoire may be not represen-

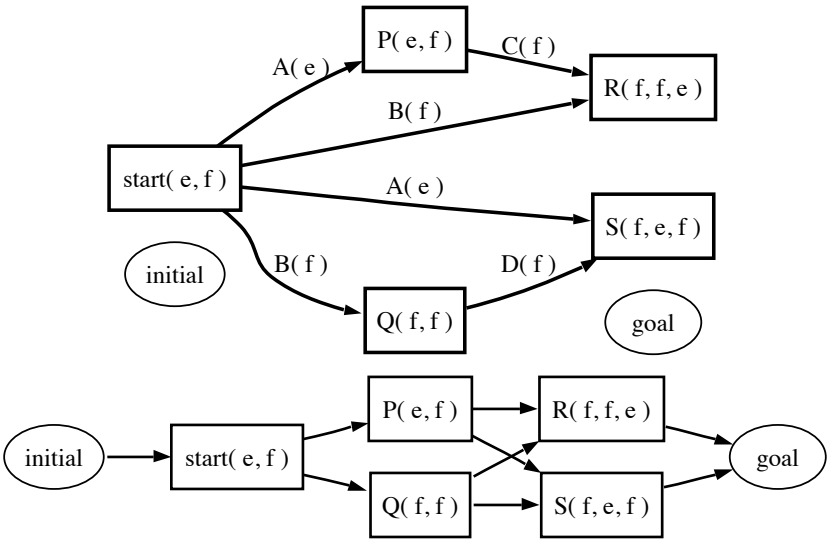


Figure 6.30: The causal structure (left) of a solution to the P2 problem in the *CrissCross* domain and the induced temporal structure (right).

tative enough for the problems in this domain. In this case, further experimentation became necessary and we hoped that more candidates from our global strategy portfolio would turn out to be definitely suitable or unsuitable, respectively the additional participants would shape the statistical ranges such that a better judgement on the present strategies could be made. We believe that this explanation is valid, but also that it is not the only mechanism we observe.

For the second explanation we have to draw from the basic strategy concepts, in particular the "task sharing" of modification and plan selection functions. Until now, we perceived strategy performance as an emerging phenomenon to which the two selection mechanisms contribute in a modular way. This was evidently confirmed for *Satellite* and *UM Translog* problems by the exceptionally good or bad performance of good or bad components, respectively. In essence, finding a top performing strategy can be viewed as finding the right combination of top performing plan and modification selections. While this argument is generally a valid one, it does not take into account the fundamental difference between the selection parts, which is their strategic *scope*, that is to say, the *local* plan perspective of modification selection and the *global* fringe perspective of the plan selection. In this view, a non-existent modification selection trend can be interpreted as a reduced influence of the local decisions on performance *per se*. Of course, there is still an interdependence between the two strategy components, otherwise the results for a plan selection would not significantly differ from one modification selection function to another. But for some reason, in this domain providing suitable plan refinements does not support or delay the process systematically.

We conjecture that the *CrissCross* domain is completely different in nature and therefore look more closely into the corresponding plan generation processes. Fig. 6.30 shows a typical solution structure in the *CrissCross* domain, namely in the P2 problem instance. If we recapitulate the domain's task decomposition structure (Fig. 5.17, p. 211), the causal interactions between the alternative expansions are deliberately hidden in the hierarchy of method application. As an effect of this way of modelling, any domain-independent strategy in principle cannot properly anticipate the refinement situation beyond the currently processed plan. These considerations perfectly agree with our findings and explanation hypothesis above. The final evidence gives us the examination of a typical search space in the *CrissCross* domain. Fig. 6.31 shows two of them, taken from two completely different strategies. The left one is built by the preference of more constrained plan, the typical depth-first method, and the right one is constructed by a modification-based HotSpot avoidance.

We can make two important observations here: First, there are long chains of singular decisions, so-called "unit modifications", which are not subject to modification selection (cf. p. 172). The recurring deferral of "real" points of choice in the generation of refinements shifts search control to a great extent towards plan

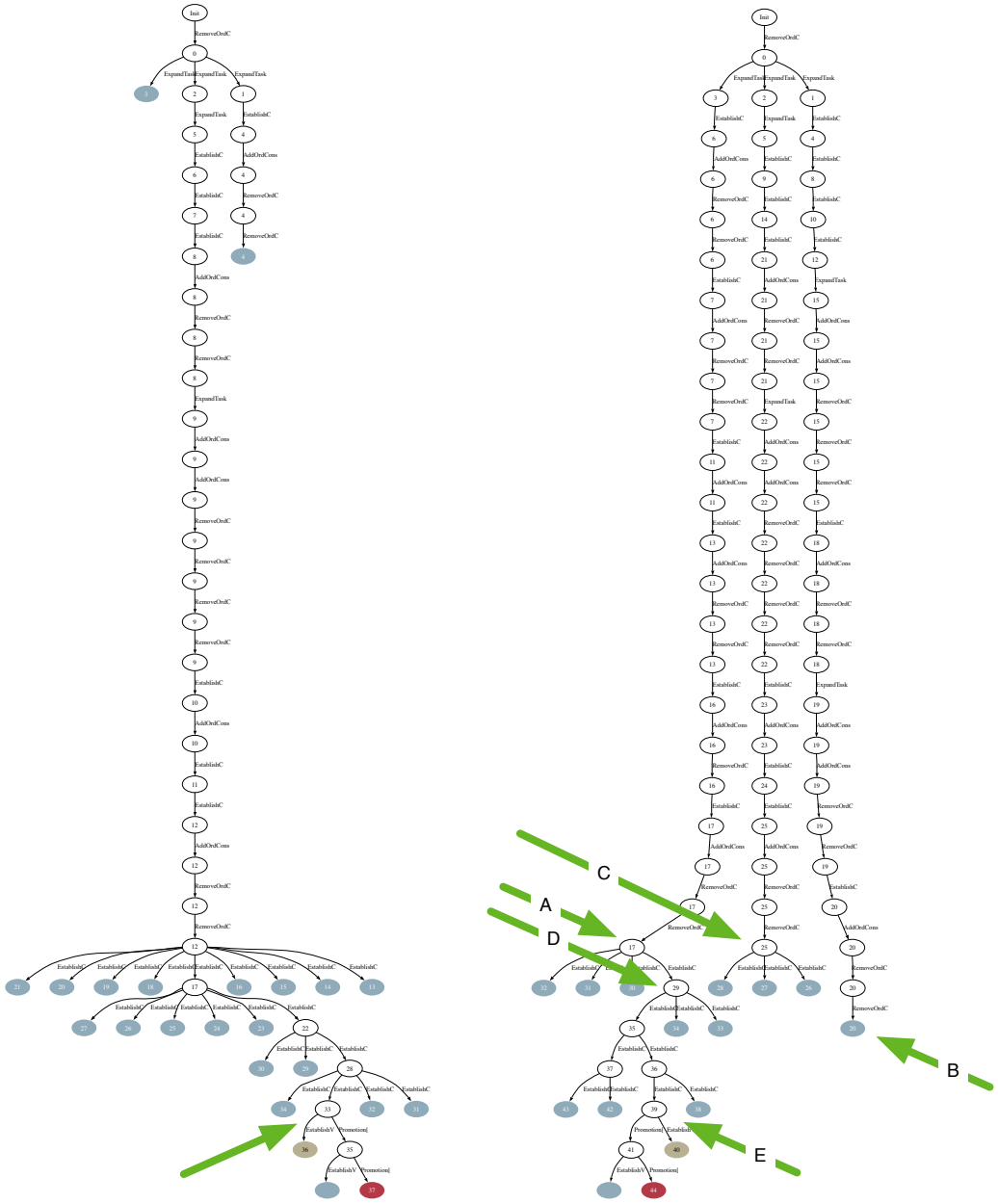


Figure 6.31: The spanned plan spaces by the modification selection $f^{modSel}_{HZone} \triangleright f^{modSel}_{LCF} \triangleright f^{modSel}_{IndUniHS}$ and plan selection $f^{planSel}_{ConstrPlans} \triangleright f^{planSel}_{FewerHZones}$ (left) and the strategy tuple consisting of $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ and plan selection $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ (right) with a hybrid planning system configuration on the P2 problem in the *CrissCross* domain (**red** = solution, **blue** = open node, **beige** = discarded node).

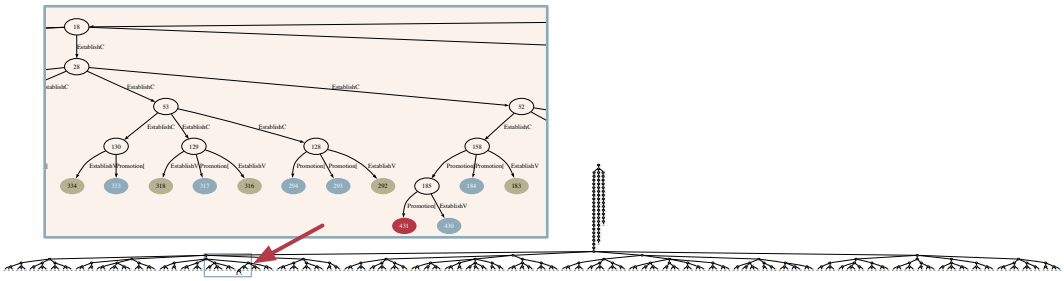


Figure 6.32: The spanned plan space by the modification selection $f^{modSel}_{HZone} \triangleright f^{modSel}_{LCF} \triangleright f^{modSel}_{IndUniHS}$ and plan selection $f^{planSel}_{ConstrPlans^{-1}} \triangleright f^{planSel}_{FewerHZones}$ with a hybrid planning system configuration on the P2 problem in the *CrissCross* domain (**red** = solution, **blue** = open node, **beige** = discarded node).

selection. Second, this effect is emphasized by the method design: the first branching decisions generate unsuspicious alternative refinement options, which are consequently not to be discarded like it is often the case in *UM Translog* or *Satellite*. It is important to stress that the alternative expansions are more or less equivalent from the point of view of most modification metrics, including the HotSpot and HotZone computations, since they do not introduce substantially differing linkage. The treatment of causal threats, which are inevitable but hidden in the *CrissCross* domain, happens at an extremely late stage of the planning process: The green arrow in the left part of Fig. 6.31, respectively the "E" labeled arrow in the right figure point to this decision. It is **only at these points** where modification selection does begin to contribute significantly and is actually able to make a difference. Please note that in both search spaces threat handling only occurs once, namely on the solution path, and that only one inconsistent plan can be discarded before a solution is reached. Given that many strategies draw information from sufficiently different and evidently inconsistent candidates, we can thus imagine how difficult these problems must be. The global failure rates are only in a tolerable range because of the many symmetrical solutions that are available.

The basic numbers for the search spaces are the following: $f^{modSel}_{HZone} \triangleright f^{modSel}_{LCF} \triangleright f^{modSel}_{IndUniHS}$ with plan selection $f^{planSel}_{ConstrPlans} \triangleright f^{planSel}_{FewerHZones}$ explores 40 plans up to a depth of 30, leaving 21 open (53%) and discarding one (3%). Taking into consideration the 16 unit modifications (many of them are inference modifications), the branching factor is 2,4. The other focused strategy $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ with $f^{planSel}_{FewerModBHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ operates in a similar fashion and hence the similar data is obtained: 47 nodes are explored in three major threads, the space is 30 nodes deep and contains 28% open (13) and 2% (1) discarded nodes. The threads are mainly built from unit modifications, including many inference modifications, and with only nine decision points, the branching factor is about 1,4.

For the sake of completeness, let us also look into the less constraint plan preference, which represents the breadth-first type of plan space traversal. Fig. 6.32 shows the search space as it presents itself to $f^{modSel}_{HZone} \triangleright f^{modSel}_{LCF} \triangleright f^{modSel}_{IndUniHS}$ with $f^{planSel}_{ConstrPlans^{-1}} \triangleright f^{planSel}_{FewerHZones}$. The search tree is certainly too large to be displayed properly, but the basic shape is already indicating the key problem: the combinatorial explosion beneath the three linear threads. This particular planning episode defers two open nodes in the main trunk and focuses on the third one that induces a huge amount of symmetrical causality construction and fixing. The figure shows a small detail in a magnification of the area in which the solution is found; it thereby gives an idea why this plan generation took ten times longer than the runs above. We see a construction of causal links that leads to numerous isomorphic sub-trees of height 5 in which more or less every second leaf constitutes a developable candidate. We have examined a similar, complete tree, generated by true breadth-first plan selection with an LCF modification selection, which confirms that the first solutions appear at a depth of 6, just as they do here. We also find that the space fragment visited by the `lcp` component as it is shown in Fig. 6.32 is actually representative in terms of self-similar subspaces. The breadth-first strategy encounters 20.000 twin solutions within the first 70.000 nodes (the fringe contains at this point 40.000 nodes).[13] In other words, if

[13] A comparison: The first ten solutions for the *CrissCross* P2 problem are found by the breadth-first search within 5.400 cycles at a final fringe size of about 3.300 nodes, that means, the fringe holds more than 60% of the nodes. In the *Satellite* problem that deals with 2 observations on one platform in one mode, the first ten solutions take 26.800 cycles and the fringe contains only 2.100 nodes (8%) and for *UM Translog* we reach the ten solutions for the TankerTruck problem in 2.400 cycles with a fringe size of 170 (7%).

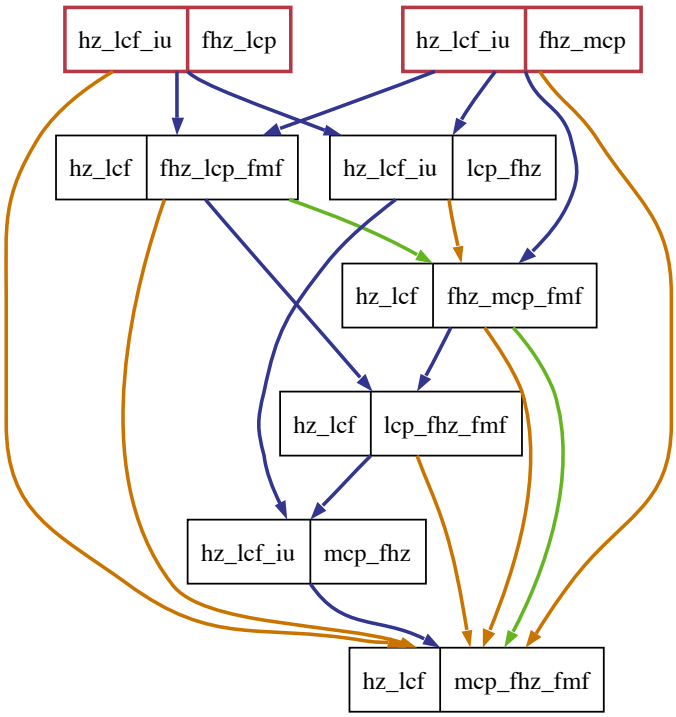


Figure 6.33: A detailed analysis of performance characteristics of related strategy combinations in the *CrissCross* domain. The edges represent a dominance relationship with respect to efficiency (**orange**), stability (**green**), and reliability (**blue**). **Red** boxes denote undominated strategies.

a strategy is clueless when it enters this area, it may need an arbitrary number of plan refinement steps to reach a decisive search node.

Consequently, the difficulty for any strategy in the *CrissCross* domain can be attributed to two aspects: the first is the amount of paths through the search space, in which more or less every alternative development is supported up to a certain depth. The second is the fact that all these alternatives are very similar or even isomorphic plans that cannot be discriminated by many selection principles. It is therefore an essential property of a successful strategy to focus on a single sub-tree.

Let us return to the least constrained plan preference: the search space may be structurally degenerated, but its global proportions are not much different from the successful examples. With a total number of 433 visited nodes and 132 open ones (30%), it is more or less a scaled version of the previous ones. The main trunk with 36 unit modifications is of course similar, only the number of discarded plans is much higher (123, 28%). However, we have to keep in mind that with any additional planning objective, say, a second initial task as in P2-2, the above described overhead with respect to causality maintenance will grow exponentially and the degeneration effect will predominate.

As we have learned, plan selection in the *CrissCross* domain is in a more prominent position of strategic decisions than before. The focus on related strategy combinations therefore becomes more significant as well. The detailed performance analysis is given in Fig. 6.33 and the results are definite. Concerning the relationship between the preference of more constrained plans and that of less constrained ones, we have already discussed aspects of their plan space development. However, although the diagrams suggest the depth-oriented `mcp` paradigm to be superior, our findings imply the opposite: in terms of performance characteristics the results are that least constrained plan preference, that means $f^{planSel}_{ConstrPlans^{-1}}$ as primary plan selection, is more efficient and reliable than the un-inverted strategy. We note that this unintuitive result occurred in the *Satellite* domain as well, The reason for this is not trivial: For didactic purposes we have chosen two extreme candidates for the example search spaces above such that the `mcp` strategy did exceptionally well and the `lcp` exceptionally poor. But as we have argued before, the probability of finding the right plan in a cloud of similar ones also involves chance, and in the larger problems the odds

for encountering a plan are very high in some of the "threat resolution sub-trees" but at the same time very low in others. Once a solution-free sub-tree is entered, a depth-first method will hardly come across a solution efficiently (and sometimes even not within the given time horizon). This is the moment when modification selection comes into play: The modification selection functions that are deployed in this sub-series of experiments prefer modifications that evade HotZone flaws. As it turns out, the preference of a more constrained plan leads to a tighter connectivity in the plan, which in turn leads to sets of uniformly connected flaws – the HotZone modification selection becomes blind. As a consequence, the isomorphic sub-trees typically do not contain unit modifications but multi-modification flaws, even more since the plans are relatively well developed and therefore all forms of threat resolution become available. These plan refinement options are mostly ordered randomly because the modification selection was un-decided. Hence, a probabilistic element enters the process, a probability for hitting a solution path, which we cannot quantify properly yet. In the end, as an effect of this phenomenon, we see the breadth-oriented approaches to be more successful on average. We note that larger numbers of experimental runs per configuration are necessary in order to address this issue and to find proper estimations for the success rates.

A last comment on the way of plan generation as it is performed by the strategy that is aware of modification-based HotSpots (right diagram in Fig. 6.31): The green arrows in the figure point to important decisions and show how they are deferred over the plan generation process. First, the trunk is developed in a straight thread until plan A is reached. This plan stays as an unselected member in the fringe during the development of the B thread and finally the thread that leads to C. These plans are then kept passive in the fringe while A is revisited and developed into a solution. Such a kind of opportunistic plan focus pays off in the *CrissCross* domain.

Let us now turn to other examples of strategy permutations, extension, refinements, and antagonists.

As we can see in the performance analysis of Fig. 6.33, the preference of fewer HotZone clusters is more reliable and efficient than any other combination in the focus set, including the switched versions. The addition of a third plan selection function is surprisingly decreasing performance in all configurations. But recall the above findings concerning the plan space structure: the regular structure of the sub-trees makes the additional fewer modifications preference ($f^{planSel}_{Fewer-\mathbb{M}}$) ineffective. We therefore believe that it is the helpfulness of the third modification selection that an advantage of these strategies, rather than it would be a matter of incompatibility of the third plan selection that is a disadvantage of the others. Another clear finding is that the comparison CL+OCA versus PSA+OCA can be concluded because the *CrissCross* problems are the third evaluation that confirms the usefulness of PSA+OCA; it is more efficient, stable, and particular much more reliable (see data in Tab. B.9).

A more subtle question about strategy relatedness is that of preferring fewer HotZone clusters versus preferring plans in which the maximum overlap in HotSpot elements is minimal ($f^{planSel}_{FewerHZones}$ and $f^{planSel}_{LeastHZone}$ with labels `fhz` and `lhz`). While the *Satellite* domain did not provide clear results, the *UM Translog* problems showed a trend and the *CrissCross* domain confirms it: Strategies that deploy the preference of fewer HotZone clusters are consistently better performing than those with `lhz`. This result manifests itself in the efficiency-centered analysis where 12 `fhz` combinations participate but only 2 `lhz` (Fig. 6.27) and in the significantly better `fhz` ranking results in the individual characteristics (Tab. 6.14, 6.16, and 6.18). But also in direct comparisons the FewerHZones plan selection dominates the LeastHZone instances in 14% of all possible cases in terms of stability and in 60% of all cases with respect to reliability. We however repeat our suggestion from the *UM Translog* domain, namely that these result have to be confirmed in experiments that are dedicated to examining the role of multiple similar sub-problems.

The *CrissCross* domain is definitely no application scenario for the *UMCP* and *SHOP* strategy "enhancements" `umcp+` and `shop+`. Although the latter is at least undominated with respect to efficiency, the rankings as well as the individual comparison is without doubt in favour of the original strategies.

Concerning the family of HotSpot functions, we first examine the respective plan selections $f^{planSel}_{DirUniHS}$ and $f^{planSel}_{IndUniHS}$. While in the *Satellite* domain the direct HotSpot computation was preferable and in *UM Translog* indirect HotSpots were slightly superior, *CrissCross* problems are more adequate for the indirect function. Although the ranking results are disappointing for both and hard to compare, a direct domination analysis reveals that (1) 19 strategies which deploy the indirect primary plan selection IndUniHS are more efficient

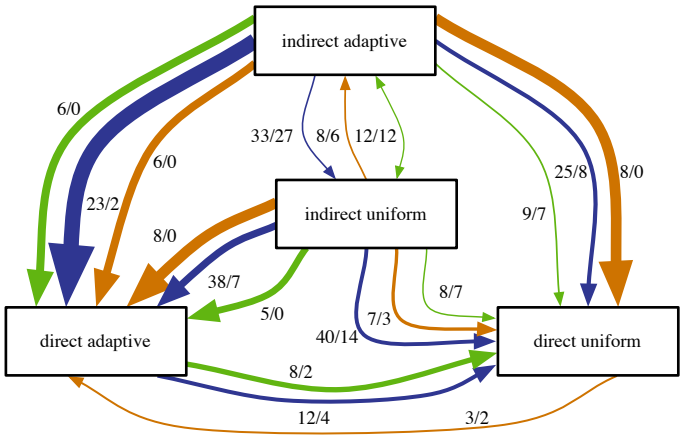


Figure 6.34: Quantitative dominance comparison between direct, indirect, adaptive, and uniform modification selection in the *CrissCross* domain. Edge labels denote number of dominant instances in / contrary to edge direction; edge width corresponds to dominance ratio (**orange** = efficiency, **green** = stability, **blue** = reliability).

than their direct counterparts and 15 less, (2) 10 are more stable with 7 against, and (3) 31 indirect combinations are more reliable with only 17 opposite findings. The superiority is not a completely convincing one, but the tendency is clear. However, with one exception in the efficiency-centered performance evaluation, the individuals that are built from the HotSpot plan selections do not perform satisfactory in this domain as well. This supports our conjecture from the *UM Translog* discussions that the basic HotSpot concept is better suited for modification selection or as a subsequent tie-break decision. It has however taken into consideration that *CrissCross* plans are relatively small in terms of plan steps and state facts and are therefore causally very dense structures. This constellation is naturally not very well suited for the (indirect) HotSpot selection. From this point of view, these functions contribute surprisingly well to the strategy instances and we recommend that they should be included in further experimentation as secondary plan selections.

The modification selection functions that are based on the HotSpot technology do perform well, but as we have seen above, the *CrissCross* domain does not induce a clear positive or negative connotation on any modification selection. It is however worth noting that three of the six most reliable strategies and 15 of the 41 most efficient ones are combinations with indirect adaptive and indirect uniform modification selection; two of them are even rated most performing in both categories (Fig. 6.27 and 6.29). We focused the evaluation results on the clique that deploys HotSpot modification selection functions – the corresponding dominance analysis is shown in Fig. 6.34. The results are undoubtedly supporting the deployment of an indirect HotSpot computation rather than a direct one, the respective strategies dominate the direct combinations over all characteristics clearly. Secondly, the adaptive HotSpot strategies are apparently more stable and reliable than the uniform versions. We believe that the minor inefficiency is due to an adaptation lag and therefore suggest more experiments concerning suitable initial HotSpot weights and adaptation algorithms. But we would like to point out that even our initially blind-guessed strategy parametrization yields a considerably more reliable behaviour than the static, uniform method.

The *CrissCross* part of our evaluation agrees on the suitability of indirect HotSpot computation for both, modification and plan selection functions. We therefore may expect that the respective functions are compatible and their combination to amplify the tendency of the components: The experimental data however shows that neither combination, that means neither $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndUniHS}$ $f^{planSel}_{IndUniHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ nor $f^{modSel}_{LCF} \triangleright f^{modSel}_{DirUniHS}$ $f^{planSel}_{DirUniHS} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$, exhibits a satisfactory efficiency, stability, or reliability above the median (Tab. 6.14, 6.18, and 6.16). The same holds for a cross-wise combination, too. The *CrissCross* domain displays once more the same phenomena as the *Satellite* domain.

The HotZone technique turns out only to work synergetic if we deploy the $f^{modSel}_{HZone} \triangleright f^{modSel}_{LCF}$ together with

	Efficiency		Stability		Reliability	
Strategy	dom.	dom. by	dom.	dom. by	dom.	dom. by
CL+OCA*	7	8	1	48	7	54
PSA+OCA**	8	0	4	3	41	11
UMCP	13	0	2	0	19	14
SHOP	23	0	4	0	16	8
EMS	25	0	15	1	87	0

* Average dominance values for 10 combinations.

** Average dominance values for 11 combinations.

Table 6.19: The dominance situation for the classical strategy implementations in the *CrissCross* domain.

$f^{planSel}_{FewerHZones} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$. Its efficiency quality is very well, given the range of the plan selection, but reliability and stability leave a lot to be desired. All other combinations are not worth mentioning. Mixing a HotSpot modification selection with a HotZone plan selection is apparently a promising concept: $f^{modSel}_{LCF} \triangleright f^{modSel}_{IndAdaptHS}$ with $f^{planSel}_{FewerHZones} \triangleright f^{planSel}_{Fewer-\mathbb{M}}$ is one of six suchlike combinations that are among the most efficient ones (cf. Fig. 6.27). These results confirm our previous compatibility-related findings as well as our suggestions to aim HotSpot and HotZone techniques at modification and plan selection, respectively.

Analysis of Classical Strategies

The classical strategies are in good positions within the performance evaluation of the *CrissCross* problem instances. *EMS*, *UMCP*, and *SHOP* are among the most efficient strategies, together with eleven PSA+OCA combinations. Concerning reliability, and *EMS* and two PSA+OCA strategies constitute half of the most reliable candidates and one of the finalist. Tab. 6.19 shows the dominance situation with respect to the whole strategy population. It appears that the row order coincidentally reflects the global positioning in the field. The numbers are a combination of the *Satellite* results (CL+OCA and PSA+OCA values) and the *UM Translog* findings (*UMCP* and *SHOP*), including for example the mostly class-intern reliability-dominance of CL+OCA strategy. The *EMS* planning strategy is an outlier with an incredible reliability rate; it should however be mentioned that with no missed termination, the two PSA+OCA reliability finalists also dominate practically every other strategy and are concealed in the figure by the statistical measures.

Our focus on the quantitative dominance relationships within the classical subset of candidates summarizes the observation above. Fig. 6.35 shows that the main differences between the classical combinations lie in their reliability, the other characteristics round off the picture in a consistent way, their dominance is however not as prominently developed.

That leaves us with the final question about proper plan development from the classical point of view: it is definitely necessary in the *CrissCross* domain to take decomposition as a *specific* event into consideration and not to treat it indirectly as a mere "cheap" provider of plan steps; otherwise the CL+OCA heuristic would not have turned out to be insignificant. The supremacy of the *expand-then-make-sound* principle is apparent and implies to suggest an extremely cautious policy with respect to task expansion. This finding is consistent with our considerations about constrained plan preference above. *SHOP* is apparently not cautious enough and with a global failure ratio of 11% (cf. Tab. B.9) considerably less reliable than *EMS*. We believe that this is due to the relatively short parallel action sequences in the *CrissCross* solutions, which are causally linked very late in the decomposition process. This structure does not develop regular causal chains, that means, a pattern of step-by-step causality, and therefore does not allow to profit too much from dealing with the issues of early tasks first. It is also interesting that the PSA+OCA way of plan development, which intuitively would be placed "in the middle" between *EMS* and *UMCP*, cannot prevail itself against the eager expansion tactics. This matches our view of losing focus when entering clusters of symmetric plans.

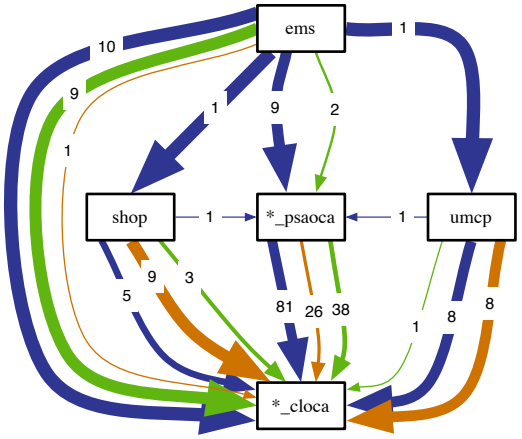


Figure 6.35: Quantitative dominance comparison between classical strategy combinations in the *CrissCross* domain. Edge labels denote number of dominant instances; edge width corresponds to the ratio of dominance declarations to the total number of combinatorial possibilities (**orange** = efficiency, **green** = stability, **blue** = reliability).

Problem Analysis

The *CrissCross* domain has been designed for experimentation and has consequently a built-in concept of difficulty levels along three dimensions: First, we defined three basic instances P0, P1, and P2 as representatives for archetypical instances of causal relations in a small plan fragment. The included decomposition methods stand for possible ways of setting up corresponding decomposition hierarchies, including the problem of "accidently" hiding causal connections. The intended difficulty progression is indicated by the numbering: P0 is the simplest problem, followed by P1 and finally P2. The second and third dimension are ways of combining basic instances in terms of repetition, for example three P0 instances into P0-3, and in terms of diversity, for example handling a P0 and a P2 instance within one course of action for P0-P2. Intuitively, we would expect that our strategies induce a difficulty ranking in terms of the number of basic instances in the initial plan but not between repetitive and diverse problems, because of the variety of strategic methods.

Our experiments basically confirm these expectations, however, partly due to the relatively small sample sizes, not developed in such a fine granularity. Figures 6.36 to 6.38 show the problem difficulty levels as they are induced by the respective strategy characteristic dominances. As we have expected, the single basic instances are categorized as simple and the ternary problems as difficult. The missing categorization of intermediate levels is however not satisfactory.

One remarkable insight into the *CrissCross* domain is probably closely related to the phenomenon of fuzzy difficulty levels, namely the fact that *CrissCross* problems are already surprisingly complex problems per se, as we have seen in the above performance investigations. An additional evidence is the global failure rate of 31% (Tab. B.9), which is not as dramatic as in the *Satellite* domain, but considerably worse than for *UM Translog*. *CrissCross* reminds us that while the hybrid planning paradigm allows for expressive procedural constructs and concise modular formulations at the same time, its domain models are still responsible for the shape of the search space. Although our approach works with completely declarative models, an in some sense "unfortunate" formulation may nevertheless induce intractable combinatorial phenomena, in other words, unnecessarily generate isomorphisms. Please note the subtle difference between isomorphic plans and isomorphic plan generation paths, which both are an issue in refinement-based planning. Recall that systematic algorithms are able to avoid the generation of redundant paths at the cost of flexibility (cf. discussion in Sec. 2.8.5) and that it is thereby also possible to eliminate many of the duplicate nodes from the search tree with justifiable effort (finding all duplicates is computationally very complex). If we

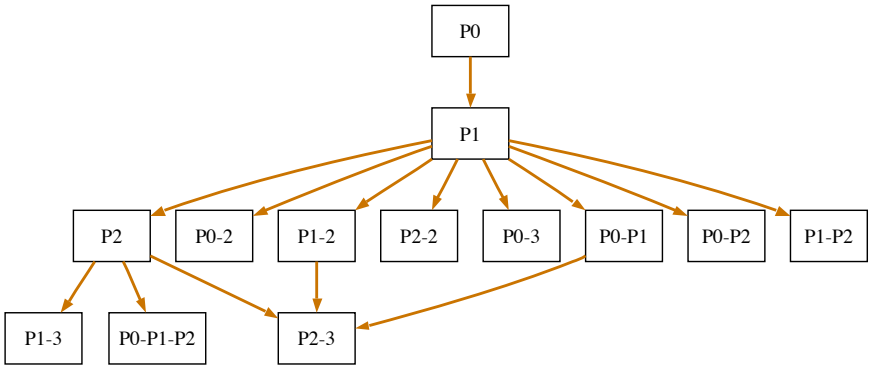


Figure 6.36: Difficulty of the problems in the *CrissCross* domain in terms of efficiency.

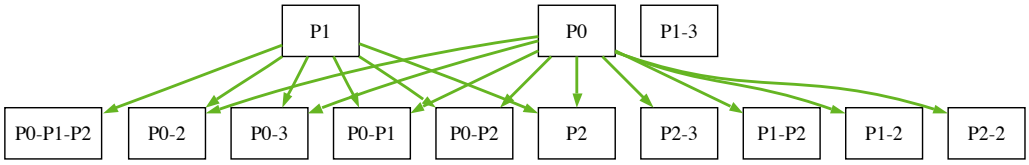


Figure 6.37: Difficulty of the problems in the *CrissCross* domain in terms of stability.

were restricted to Partial-Order Planning, plan and plan generation path isomorphism would coincide and we would "only" have to deal with systematicity and the like. In hybrid planning, the isomorphisms may however be introduced as an integral part of the model itself, as it is the case with *CrissCross*, where we see decomposition methods that basically produce the same refinements within two modification application steps. Another anomaly[14] that we observe in this domain is the scattered introduction of causal dependencies that necessarily interfere in later refinements. In all these aspects, any systematic search gets fooled, and the later the anomalies are encountered the more noticeable are their consequences. These are obviously facets of planning's computational complexity, which is inherent to any planning approach in general. We would however like to point out that it is also a strong argument for empirical studies as integral part of a planning application development: The evaluation data has to be analyzed for symptoms of pathogenic hidden-causality situations like we described in the *CrissCross* analysis. This may initiate a re-design of the domain model and in particular the decomposition methods, if appropriate.

In order to be able to draw further conclusions from *CrissCross* data, we believe it is necessary to classify the participating strategies in basically three categories: (a) strategies that are able to handle repetitive problem formulations, (b) those that are suitable for mixed settings, and (c) those that perform equally well in both.

[14] *CrissCross* has been deliberately designed as a challenge for planning strategies. It is however possible that a domain model accidently exhibits the same characteristics. This can hardly be detected automatically, because it typically does neither affect consistency nor produce undesired solutions; hence we call it an *anomaly*.

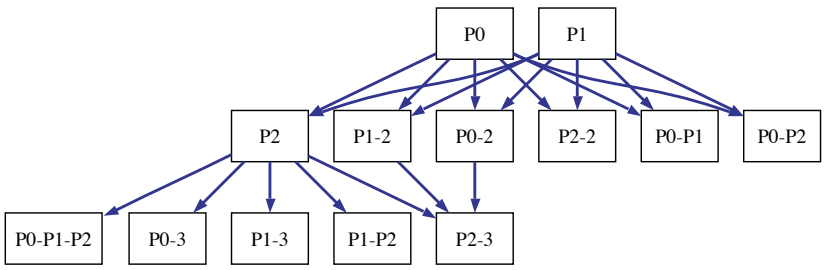


Figure 6.38: Difficulty of the problems in the *CrissCross* domain in terms of reliability.

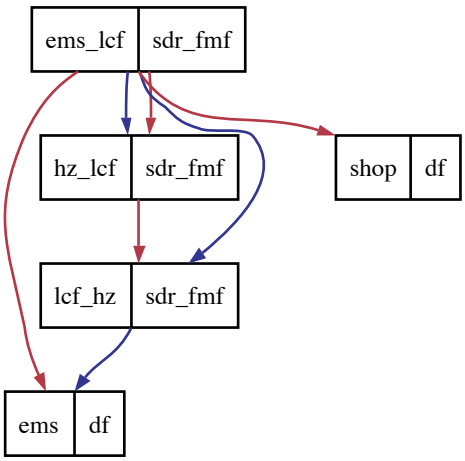


Figure 6.39: The efficiency-centered performance evaluation across all three experiment domains. Nodes are in the intersection of the three respective sets of most efficient strategies, edges represent stability and reliability dominance in the *Satellite* (**blue**) and *UM Translog* domain (**red**).

We suggest to define at least two additional basic instances and corresponding combinations and then to categorize the evaluation participants in a pre-analysis phase. When the above examinations are re-conducted in this new setting, the *CrissCross* domain will not only serve as a testbed for complicated causal interdependencies but also as a classifier for the ability to handle repetitive sub-problems.

6.6 Discussion

6.6.1 Global Results

While the previous sections focused on the interpretation of our empirical findings in each domain separately, we are now briefly revisiting some of the results in order to investigate what kind of domain-*independent* conclusions can be drawn from the data.

A question that immediately comes to mind is certainly whether the strategy portfolio contains a domain-independently well performing candidate or not. Since the dominance analysis method abstracts from the actual characteristic values and hence from domain-specific ranges, we can merge the results from the efficiency- and reliability-centered analyses. Fig. 6.39 shows our findings concerning efficiency-centered performance and we conclude that

$$f_{EMS}^{modSel} \triangleright f_{LCF}^{modSel} \quad \text{with} \quad f_{\mathbb{F}/TE}^{planSel} \triangleright f_{Fewer-\mathbb{M}}^{planSel}$$

is the best performing strategy over all three evaluation domains. It has been among the winners in *UM Translog* as well as in *CrissCross*. All four finalist strategies that are depicted in the figure are most configuration-specific (search) efficient and the edges represent stability and reliability domination results among them in the three domains. Please note that the *CrissCross* domain did not induce any dominance between these candidates. It is interesting to see that only two plan selection functions prevail, namely the depth-first principle and the smaller detection ratio preference $f_{\mathbb{F}/TE}^{planSel}$, a very abstract measure for the degree at which a plan is flawed. Regarding the modification selection functions, the two key techniques are the conservative task expansion as it is realized by the *EMS* and *SHOP* strategies and the HotZone computation. The combinations built from these components do share a "careful" modus operandi in terms of abstract task expansion: the two classical strategies preferably complete the causal plan structures before the next task is decomposed. A similar effect emerges from the $\mathbb{F}/TE$ plan selection: Closing an open precondition or resolving a causal threat lowers the ratio because it eliminates a flaw while maintaining the number of plan steps. Expansion, however, typically raises the detection ratio because the method implementation inherits existing flaws from

Primary plan selection	Total	Undom. e/s/r			%	EC	RC
$f^{planSel}_{First}$	9	8	6	2	60	1	1
$f^{planSel}_{ConstrPlans^{-1}}$	6	4	5	–	50	–	–
$f^{planSel}_{\mathbb{F}/TE}$	30	20	18	4	47	3	3
$f^{planSel}_{FewerModBHS}$	30	12	15	5	36	5	5
$f^{planSel}_{FewerHZones}$	42	23	19	–	33	–	–
$f^{planSel}_{\text{PSA+OCA}}$	33	16	10	3	29	2	2
$f^{planSel}_{ConstrPlans}$	6	2	2	–	22	–	–
$f^{planSel}_{LeastHZone}$	30	5	9	–	16	–	–
$f^{planSel}_{IndUniHS}$	30	2	10	–	13	–	–
$f^{planSel}_{Addr-\mathbb{F}_{\texttt{AbstrTask}}{}^{-1}}$	3	–	–	1	11	–	1
$f^{planSel}_{DirUniHS}$	30	2	7	–	10	–	–
$f^{planSel}_{\text{CL+OCA}}$	30	–	2	–	2	–	–
Total:	**279**	**94**	**103**	**15**	**25**	**11**	**12**

Table 6.20: Primary plan selections over all three domains: Number of undominated system configuration instances according to (e)fficiency, (s)tability, (r)eliability, efficiency-centered performance (EC), and reliability-centered performance (RC). % stands for the ratio of possibly to factually undominated.

the abstract task (except the flaw indicating the task being abstract) and the proportion of flaws to task expressions is relatively high for under-specified task networks in general. The main difference between *EMS* and *SHOP* and the $f^{planSel}_{\mathbb{F}/TE}$ combinations is that the depth-first classical strategies implement the deferral of expansions path-wise and the flaw-oriented ones do so fringe-wise.
We are of course pleased that our advanced technology portfolio is able to perform domain-independently competitive with respect to the well-established classical strategies *EMS* and *SHOP*.

Regarding the most configuration-specific (search) reliable strategies, we have to note that unfortunately there are none. The intersection of the three respective sets of most domain-specific search reliable strategies is empty. Although there are common components or switched primary and secondary selection functions, the requirements of the domains with respect to reliability are apparently too different to be met by a single individual of the given portfolio.

The analyses of primary plan selections (Tab. 6.4, 6.10, and 6.17) summarize the dominance results in the three performance characteristics according to the deployed primary plan selection function. In each domain, the strategy population induces an average of number of dominance finding incidents, which defines a domain-specific threshold above which a strategy may be called "assertive". The intersection of the assertive individuals of all three mentioned tables consists only of two primary plan selections: $f^{planSel}_{ConstrPlans^{-1}}$ and $f^{planSel}_{First}$. For the sake of completeness, we also give the accumulated data as shown in Tab. 6.20, but it has to be noted that these figures are easily misinterpreted without the context of the previous sections. For example, the depth-first plan selection is at the prominent pole-position because it represents in fact multiple, completely different strategies that alternate in performing extremely well on some of the problems.

Apart from these summarized categorical results, some individual strategies have to be discussed from the domain-independent point of view.

With three successful ratings of the $f^{planSel}_{\text{PSA+OCA}}$ plan selection function, we are convinced that we have identified an appropriate incarnation of the A^* principle in hybrid planning configurations. Future experiments may investigate into better informed weights of the heuristic estimators, for example, a more precise approximation of the cost of an abstract task, but the performance is already convincing at this stage. Another important property of PSA+OCA is its ability to collaborate with a great variety of supplementary strategies.

The least committing first principle is obviously a highly compatible component, too, because it is not defined via plan features or flaw dependencies, and the like, but on the branching property of the search space. f_{LCF}^{modSel} may be deployed as a primary modification selection but is also a good choice as a secondary heuristic. It is particularly effective in all situations in which no heuristic information is available, for example, in phases in which too many too similar plans are in the fringe.

It is also a very satisfactory finding that our adaptive HotSpot selections $f_{DirAdaptHS}^{modSel}$ and $f_{IndAdaptHS}^{modSel}$ are performing reasonably well in some domains, even with the given simple adaption approach. Further experimentation will help to obtain an improved parameter adjustment for both, the initial values as well as the adaptation procedure.

The performance of the *SHOP* strategy evidently relates to the presence of rigid facts and to the usage of initial state facts over the course of action. Ordered task decomposition turns out to be adequate for *Satellite* problems, in which many decisions directly depend on rigid state features that are sufficiently constrained. For example, the expansion of the observation task relates the required image mode to a parameter in the leading task for set-up procedure handling. This instrument activation is the head of the plan and selects the appropriate constellation of instrument and satellite that is able to provide the required image mode. These causal links are decisions safe from "backtracking" because the parametrization cannot fail at this point. The contrary situation is given in the *UM Translog* domain: the first decomposition step merely separates the loading procedure from the succeeding vehicle movement. At this point, there is no feedback from the movement task which kind of transport has to be realized. The focus on loading is therefore not sufficiently narrowing down the choices and it may turn out that necessary decisions in the movement tasks invalidate the commitments made in the loading tasks. While the satellite observations can be planned from the power-setup to the image taking, the transportation tasks have to be developed more opportunistically back and forth. *CrissCross* combines the property of being not defined properly in execution order, it is furthermore defined over flexible symbols only, and it is in addition hiding cross-wise causal interactions. The latter however imposes such strong constraints on the subsequent plan that any backtracking is at least initiated quickly.

But there is one more general issue with the depth-first strategies being so successful. Although all of them are classical, well established search procedures, they are sometimes not too reliable because they get lost in some solution-sparse ares. What helps them to gain performance is that they do not suffer from the problem of monocultures: If the strategy does not focus properly on a solution path, then cliques of sufficiently similar plans (depending on the selection schema) may populate the head of the fringe among which the strategy has to choose randomly. This typically becomes a self-feeding process that resembles an uninformed breadth-first search until either the experiment time is up or by chance all members of the clique fail instantly.

Collecting empirical data on particular strategy combinations also provides insights into the effects of the intended "job sharing" between plan and modification selection functions. The plan selection is evidently primarily responsible for efficiency and reliability of the strategy combination, while modification selection plays its major role in contributing stability. This phenomenon becomes in particular prominent in the rank classification matrices of the *UM Translog* performance characteristics: the statistical metrics of efficiency and reliability induce (plan selection) row patterns and stability induces (modification selection) column patterns. For the other two domains, we have to consult the "raw" data tables, because the patterns are not salient enough for the selection-wise aggregation; the plan-selection-oriented stripes are however noticeable for efficiency (Tab. B.1/B.7) and reliability (Tab. B.3/B.9). The sample variance presentation shows the corresponding patterns if re-organized according to modification selection functions.

What we called job sharing can however only be sufficiently developed if the strategy is working in some sense organized as a combination. This can be observed on the one hand in positive examples that combine a very efficient plan selection with a very reliable modification selection (resulting in a very efficient combination) and on the other hand in negative examples where two non-performing components come together to build an inefficient strategy. While in the first case an efficient search focus is sharpened by the local refinement view, in the second case a remarkably inefficient search is worsened (as we have conjectured in the strategy chapter, a sufficiently bad strategy is worth being tried out as an inversion). If however the local information does not allow to infer any reasonable sort of quality for a particular refinement, the modification selection function becomes blind, the selection process consequently random, and the plan

selection function unsupported. We can observe this at wide range of strategies in the *CrissCross* domain (see discussion in previous section). It has however to be noted that the missing modification selection pattern is merely a symptom of such a domain model anomaly like the "loss of refinement quality". It may be compensated by heuristics that indirectly correspond to branching related measures: the *expand-then-make-sound* principle is not only a way of describing conceptually how to develop a plan but it is implicitly also a commitment schema. It obviously depends on the domain model, how many constraints an expansion implies on consecutive plan refinements, how many *make-sound* alternatives are available on average, and so on. In this way, a modification selection may constitute a domain-specific *least commitment* strategy.

Concerning our performance characteristics, it turned out that although they are somehow semantically related, their practical implications are different. While it is intuitive to associate efficiency with the concept of availability of that efficiency, the stability characteristic does not seem to be that much connected to the other two ones. Furthermore, we do find stability not as developed as the other characteristics and it consequently appears in most analyses as to be of secondary character. This may be an effect of the relatively small sample size but also of a (on some problems) too inefficient strategy population such that we get trivial sample variance values of 0 if only one single run was successful or the value of "unsolved" if the strategy dropped the problem completely.

The opposite experience can be made regarding reliability. This performance characteristic is considerably more selective than stability and efficiency. We also observe that reliable strategies are practically never inefficient or unstable, quite the contrary, the most reliable candidates turn out to be the most efficient as well. Future evaluations may take advantage of this finding: We propose to focus on reliability-centered performance evaluations in order to obtain more concise experimental setups and a more economic experimentation. That means, we propose to begin with a reliability dominance analysis in one domain in order to reduce all subsequent experiments and analyses on the most reliable subset of that reference domain. In combination with (or instead of) this candidate reduction, we can also reduce the experimentation time as well. The experimental frame can be successively adapted by taking into account the measured performance of the reduced candidate set, in other words by adopting the reliability winners' average efficiency as a new cut-off for the successor experiments. By doing so, the efficiency of the most reliable strategies becomes the bottom line for the population and unnecessary inefficient runs are aborted early.

While our abstract notion of performance supports an domain-independent interpretation of our results by abstracting from the actual domain model, it is however not advisable to try to generalize the problem difficulty analysis. It is of course valid, for example, to derive from the average failure rates that the *Satellite* problems are regarded as more difficult than the *CrissCross* problems, which in turn are more difficult than those of the *UM Translog* domain. However, when we start to refer to domain A as being more difficult than domain B, the side condition "given the respective problem sets" must not be be left out and hence makes no sense in practice. It is obviously meaningless to compare the effort of planning one package transport with that of planning two satellite observations. Domain difficulty, or *complexity*, is better addressed by classification techniques in terms of computational complexity (analytical reduction on problems in specific classes of the polynomial hierarchy [133]), expressivity (transformational reduction in terms of computational effort for problem translation [83]), and the like.

But we note that the evaluation subjects may as well be disjunct problem subsets, each of which representing a specific way of problem formulation, for example, specifying a package transport via an abstract task versus a classical goal state feature. Or we may want to compare domain model variations, for example, testing different decomposition methods for the *UM Translog* loading tasks. In these examples, the domain models are comparable and therefore a generalized problem difficulty analysis is worthwhile.

6.6.2 Lessons Learned

The previous sections describe many improvements that we have discovered while conducting the experiments and analyzing the data. A key issue that always raises in this context is obviously the number of experiments that are necessary to be able to draw valid conclusions from the studied material. When we decided to conduct 5 runs per configuration (domain model, problem instance, and deployed strategy

combination) we were well aware of the statistical problems that typically arise from such small sample sizes. Practical considerations, however, compelled us to start with this reduced experimental frame size: The above defined setup implies more than 15.000 recorded plan generation episodes, of which a total of about

- 23% of *UM Translog* problems (cf. Tab. B.6),
- 72% of *Satellite* problems (cf. Tab. B.3), and
- 31% of *CrissCross* problems (cf. Tab. B.3)

failed to solve the posed problem instance, that means approximately $1.390 + 2.670 + 1.870 = 5.930$ unsuccessful system runs. As we have mentioned before, non-terminating runs are in most cases caused by strategies that have lost their focus and explore the plan space stratum-wise like a breadth-first strategy. The enormous fringe size slows the particular implementation of the evaluation system such that no known strategy is able to re-gain its focus in time. The consequence is that practically all failures reached the temporal deadline of 150 minutes. That means, the failed runs are worth about $5.930 \times 2,5\text{h} = 14.825\text{h}$ of computation time. Our evaluation software has been running on a cluster of up to 15 machines in parallel, the total real-time consumed by the failed portion of the experiments alone was therefore about 6 weeks. The remaining 600 plan generation episodes per cluster node are realistically solved within one week, including all additional computations that are necessary to extract and process the experimental data. Regarding the large failure rates and the tremendous time consumptions, it has also to be stressed that the participating strategies and modules did not undergo any kind of heuristic fine tuning of parameters and alike, and that all system incarnations performed a complete search (cf. Sec. 2.8.5). What we report are results that are obtained by our reference implementations, which are still open to all kind of performance enhancements, including run-time optimizations and the like.

After some initial data probing and time estimation in a precursor study [231], we decided with the given time horizon and technical infrastructure to design our first empirical study in favor of a survey over some segment of our strategy portfolio. With the hereby gained experience our consecutive experimentation will be conducted statistically more accurately with larger sample sizes on more focused strategy and problem sets.

One last remark on this topic: The credo of empirical research is of course that there is no such thing as too much data, but the most important question that has to be solved is: are the acquired results statistically significant? Although we have not performed dedicated significance analyses we can at least show that the variance of values per problem is always greater than the variance within each strategy, that means that the strategies actually make a difference in performance.

Another finding of our experiments, in particular a result of the `mcp` versus `lcp` analysis, is that evaluating strategies sometimes reveals more about the domain and problem in which the evaluation takes place. We have inspected the anatomy of the search spaces and obtained some interesting results about the nature of strategies but also about those properties of the respective problem or domain that constitute its difficulty.

In particular the inspection of *CrissCross* problems is furthermore an example of how experimentation can not only be used to determine suitable strategies for a given application domain but also for identifying modeling anomalies and giving advice for debugging the respective domain model.

Last, but not least, the most important conclusion that can be drawn from this chapter is that there is a definite need for (1) a highly configurable planning system as ours and (2) for conducting large scale experiments as we did. The first becomes apparent when recalling, for instance, the discussion about preferring or rejecting constrained plans during search. As it turned out, the choice highly depends on the application domain, and the consequence is that a *domain-independent* planning system has to be sufficiently and adequately *adaptable* to any domain. The second argument aims at the following: if we have an domain-independent planning system at hand, how do we have to tailor it in order to meet the application domain's needs? This includes naturally the specification of an adequate domain model, but also the proper planning strategy in order to obtain solutions efficiently, respectively reliable (the issue of finding *efficient* solutions has not yet been addressed by this chapter). We have seen how focused an empirical evaluation can answer these questions with the necessary statistical background.

6.6.3 Related Empirical Evaluations

Planning and scheduling have always been applied technologies and therefore every new approach has been naturally subject to experimentation and competition, partly in order to prove the utility of new functionality and partly in order to prove the progress in terms of algorithmic quality. The latter can be interpreted as improving the quality of solutions as well as improving the quality of service, for example, system response time, tractable size of data, and the like.

An early example for such a performance evaluation is a documented comparison of the INTERPLAN planner with representatives from the early 1970's planning systems scene in terms of run-time [254]. Measuring the real-time consumption of the plan generation process is, of course, a very imprecise method that does not take into account multi-user environments, multi-threaded task processing, start-up and caching phenomena like the Java Class loader, Garbage Collection, and so forth. It is also worth noting that the early performance tests had the additional problem of a considerable diversity of used programming languages, installed operating systems, and deployed computer hardware.

Most of the more recent experiments are conducted within the identical (or at least comparable) hard- and software environment and are mostly targeted at comparing a new approach to its immediate predecessor or at comparing design alternatives that share the same code base. This restriction relieves the researchers of many practical and statistical issues and allows for a simple user CPU time[15] measurement. However, although this kind of analysis is documented for most approaches, it is in most cases intended merely to provide selective evidence for the improvement of the new method.

Our understanding of evaluation also covers the aspects of an empirical study in the sense of gaining insight into a matter by experiment and analysis. Positive examples are pieces of work about the idea of least commitment planning and the proper quantification of the degree of commitment in the context of the UCPOP system [113, 143, 218, 236], about the relationship between and application perspectives of total-order and partial-order planning [16], and about the interactions of refinement strategies and correlated algorithmic design decisions [148]. The central measure for these evaluations is the net running time but in some cases also a more abstract scale like the number of explored search nodes, the average branching factors induced by strategic decisions, and similar numbers that describe the shape of the search space.

The *International Planning Competition* is currently the largest evaluation effort in the area of planning [137, 171]. When first held at the *4th International Conference on Artificial Intelligence Planning Systems* in 1998, the IPC started out with five competitors that had to solve about 250 problems from seven domains, organized in two tracks. Regarding the initially proposed evaluation method, the planning systems accumulated a total score that they received by solving instances of the presented problem set. Such a single score for a planning system i on a problem j was calculated as the product of i's rank on j and the difficulty of j: $\text{score}(i,j) = (N_j - R_{ij})W_j$ where N_j is the number of planning systems competing on j. The difficulty factor W_j was thereby defined as the ratio of the median of system running times T on that particular problem j to that of all problems: $W_j = \frac{\text{median}_i T_{ij}}{\sum_m \text{median}_m T_{mn}}$.

In relation to our experiments, the early IPC evaluation approach has two methodological parallels: First, the IPC also used ranking as a means for balancing diverse problem qualities, and secondly, the problem difficulty weights W were induced by a performance measure of the competition participants. According to the competition's terminology, the ranking dimensions roughly correspond to our reliability-centered performance evaluation. The planning systems were ranked for each problem firstly according to their correctness, that means, whether they had constructed a valid solution to the problem, secondly according to their need for "advice", that means, the mount of problem-specific knowledge that had been given to the planning system, and finally according to the product of consumed CPU time and a factor obtained from the number of plan steps in the solution.

The precise calculation of the actual score has been a major debate for some time, in particular how to include CPU time properly, how to devalue unsolved problems and incorrect solutions, and how to reward solution quality. While the described compromise was aimed at balancing what we call efficiency and reliability

[15] The *user CPU time* is defined as the amount of time that corresponds to the CPU clock cycles that are spent executing user code. This measure excludes implicitly called system kernel code, input delays, etc.

(the main topic of discussion), one conceptual weakness was not addressed, namely the fact that problem difficulty has been accounted for twice: explicitly in the difficulty weight and implicitly in the solution-length factor of the runt-time ranking. We believe that this is a serious comparability issue when assigning absolute scores to candidates. Over the years, the IPC evaluation schema therefore changed into its current form of, in our terms, absolute reliability as it is formulated in its regulations:

> "So, the only thing that matters is the number of solved problems. All problems (either hard or easier) are considered equivalent. The winner of each category (sequential, temporal, net benefit) is simply the planner that solves most of the problems."

This in some sense more qualitative measure, which corresponds to our reliability characteristic, has been used for some approaches in the past, for example [144], and is nowadays more commonly found in order to maintain comparability, for instance [297].

From the technical point of view, our evaluation can be compared to the IPC as follows: The deterministic section of the latest *International Planning Competition*, held in 2008, had 16 participants, which were organized in four tracks, and an experimental frame consisting of more than thousand problems in 10 domains. The time limit for each plan generation episode was 30 minutes and the memory limit 2GB on 3GHz computers.
Our evaluation was conducted between late 2007 and beginning of 2008 and had 93 "participants", single-tracked, that competed on the basis of 5 samples of 34 problems in 3 domains. The quantity of domains and problem instances is obviously an issue and (in particular the latter) needs to be extended. We restricted the system to a time limit of 150 minutes and specified a maximum search space size of 5.000 nodes, which is in practice equivalent to a memory consumption below 256MB. The deployed "compute servers" ranged from single core 1GHz PCs to a 2GHz multiprocessor server. Having in mind that our reference implementation code is not optimized, as we mentioned above, we think that our experimental frame imposes comparable challenges to the evaluation candidates.

While the examination methods appear similar, the research focus of our empirical evaluation is however different from the competition-oriented experiments in the IPC and the testing-intensive literature: we aim at finding suitable candidates for developing "the best" strategy rather than selecting the best one from a collection of alternatives. The subtle difference regarding the analysis techniques is that our evaluation is always in favor of strategies that are not consistently expendable, in other words, that are undominated with respect to a performance criterion, whereas typical competition schemata are interested in strategies that dominate a set of competitors that is as large as possible. The former is a very conservative approach in the context of *strategy development* and the latter is in particular valid if the candidate set represents *all currently available* options. The difference becomes obvious if we think about changing our performance schemata to change from undominated strategies to those which dominate most of the other strategies. This hypothetical schema would systematically favour strategies that are specializations of bad strategies, because chances are higher to consistently dominate a related strategy than a completely different one. For this reason, we believe that our conservative approach, although not completely unproblematic, is better suited for development purposes.

Another very important difference is that our approach is not intended to deliver one single judgement or winner but rather to provide a set of techniques that support constructing, respectively assembling a proper strategy component in an application-suited system configuration. This is accounted for by proposing a set of complementary evaluation methods that cover a variety of perspectives on the examined strategies and problem instances. We believe that our approach allows for a considerably more efficient tailoring of the evaluation to the needs of the target application(s). The "single/absolute winner" scenario of a competition is not only controversial with respect to the (fusion of) performance criteria but also problematic concerning the suggested generality of the results. The latter have been obtained from few example domains, which "have peculiarities that induce a bias towards certain kinds of plans. It can not be ruled out that the picture would be very different over a different problem set, in particular over a set of 'real' problems" [130].

6.6.4 Perspective

As it is quite common to empirical studies, our evaluation did not answer all our questions definitely and at the same time raised new interesting questions for which a dedicated experimentation would be required. A recurring theme in this chapter is consequently the need for "more experimentation", which we will satisfy as part of our future work specifically as follows:

Statistical Accuracy

A recurring open issue in this chapter is that of statistical accuracy given the relative low sample size of five runs per configuration. With our subsequent studies being more focused we will be able to invest more time and computational resources into the individual candidates so that a doubling of runs can be realized. The question is of course: will ten runs be enough?

We cannot answer this question yet, because we do not know enough about the amount of indeterminism that occurs in the individual plan generation episodes. It is however very unlikely that we will obtain precise data by analytical means except for the most trivial (and uninteresting) cases of strategy design. But we have seen for some of our strategies what the induced search trees look like and discussed to some extent the respective decisions of the involved plan and modification selection functions. Consequently, what we need in future experiments is a precise recording and preprocessing of the corresponding selection results, that means in particular data about the number and type of options that end up in the same equivalence classes after the last selection function call of the strategy. From that data we will get at least an idea of how representative the individual runs probably are and are consequently able to infer suitable sample sizes.

Until then, we will begin with the doubled sample size of ten, because we believe that this gives justice to the majority of strategies while at the same time being a tractable effort.

Generality of Results

Given the various single results that we have obtained and assuming that we manage to obtain a sufficient statistical accuracy as described above, we will have to address the problem of how to generalize from our findings. Since generalization of experimental data is mostly concerned with identifying possible explanations for a set of observations (abduction), we will have to extend that set of observations quantitatively and qualitatively as much as possible, that means, we will have to include more problems and domain models in our analyses.

As an initial step, we will translate more of the IPC domains and problems and transform them into hybrid representations, if that appears reasonable. Furthermore, it will become necessary to provide instances of certain patterns in terms of domain and problem regularities, for example, the concepts of repetitive and diverse problems as we find them in the *CrissCross* domain, and the like. This includes also alternative realizations of the same domain, for instance, different ways of task hierarchization, alternative task specifications, or modeling variants with respect to sort versus resource definitions.

Our ultimate long-term research goal is a generalization of our findings from the deployed strategies and from the concrete domain models and problem instances: We want to identify model- and problem-related key properties that allow to infer from a given application situation which kind of strategy combination is most probably suitable, that means efficient, reliable, etc. This effort ideally leads to a mapping of application features to search strategies in analogy to the classification results in the area of scheduling [37]; we expect at least to gain enough insight into our strategy portfolio to be able to formulate more precise deployment guidelines.

Focused Research Questions

But we also have to take into account that generalization is always a result of the research question that has been formulated in the beginning of the experiments (cf. Def. 6.6), for example, in the IPC "which planner solves most of *the* problems?". For the competition's purpose it appears imperative to include an arbitrary wide range of diverse domains and problems *the* best planners should be able to solve. Although we agree to some extent with this greedy point of view, because it aims at systematically excluding any bias, we are also aware of the fact that this attempt is conceptually questionable: Ultimately, if literally every domain model and problem characteristic is included in the test set, a "best performance" evaluation result will become insignificant because there will be arbitrarily many specialized strategies none of which will be comparable to each other (given that the conjecture about the polynomial hierarchy holds and consequently any strategy will only be able to solve a specific portion of the possible problem spectrum).

We think that these considerations have two consequences: first of all, we will definitely build more problems and domain models with various properties, but we do have consider carefully which of them actually contribute to the research question. Secondly, we have to spend some effort on establishing worthwhile research questions that are sufficiently focused, which in turn will help us to define the representative domain and problem coverage. We will begin with the numerous open issues that are contained in the sections above; there will be sets of extended experiments with larger sample sizes, more problems, and more selectively chosen strategies.

The problem of the reduced dominance occurrence is of course a statistical phenomenon that will become even more apparent the larger our planning problem sets will become. The more problem instances are solved, the larger is the probability for an exceptionally good or bad run, which in turn implies a larger probability for finding a single result that contradicts a case of dominance. As a consequence, the set of undominated strategies will grow and with it the set of "winning strategies" as well. We will have to reconsider this issue on the basis of more available data. One possibility is to introduce k-dominance, that means, a strategy a dominates some other strategy b if and only if a is better performing (more efficient, etc.) than b except for at most k occurrences and b is not dominating a. We will have to analyze this dominance schema in future evaluations, but from our experience in the reported experimentation, k-dominance is even for $k = 1$ considerably more restrictive.

New Experimental Setups

In the reported experiments not all components participate that our strategy portfolio provides. It is therefore our primary aim to include some combinations with potential in a follow-up study in order to complete this chapter's findings. This also includes specific experiments with the adaptive HotSpot modification selection functions and other prototypical strategies that employ learning techniques: due to methodological restrictions, the adaptive strategies were required to set back their weights before each run to their initial value. We are planning to design experimental setups that are particularly tailored at addressing questions in the context of measuring the success of learning and similar dimensions. To this end, we have to modify the experimental platform such that a progression of problems can be specified (from which problems to learn and to which problems apply the learned knowledge) as well as consistent access to the strategies' memories is provided (taking into account a multi-server setting).

Another important aspect of future developments is to take into consideration parameters that are beyond the anatomy of the search space, namely quality aspects of the produced solutions. This is apparently more of an issue for resource-aware system configurations but the concept of optimization can also be applied to hybrid planning: the choice of an expansion method may affect the number of plan steps that are necessary to achieve a certain goal, plan steps may be added for establishing a goal which could have been met by a specific task implementation elsewhere, and the like. Furthermore, the current results are restricted to problems that are solvable by HTN-only systems in order to meet literature's standards on common ground. Future experiments will address the option of task insertion as well, which will challenge the strategies to add new plan steps on demand or to try to achieve the same effect by suitable decompositions. In all these experimental settings that allow for a more diverse plan quality, when one such solution quality has been

studied, these findings can be integrated and related to other qualities, for example, investigating into the relationship between cost optimal plans and the efficiency of their construction.

As we have noted before, we will not be realistically able to evaluate a larger portion of the space of strategy combination candidates by following a naive, systematic approach. Taking into account our experiences in planning strategy design, we have confidence in machine learning techniques being an appropriate means for systematically isolating suitable evaluation candidates in a pre-evaluation phase in order to define the experimental setup for a tractable amount of evaluation runs.

Our next step in this direction will therefore be to investigate into post-mortem analyses of complete plan space traversals, similar to the computations we suggest for increasing statistical accuracy. We will use them to evaluate the respective strategy components in a re-play fashion, similar to learning control rules for the PRODIGY system from system traces [35, 274]. We begin with unary primary selection functions, reproduce their selection behaviour in the prepared search space, and use statistical estimators to quantify their characteristics like the number of plans visited or the induced branching factors. These numbers are the basis for a candidate assembly component that systematically adds subsequent strategy components and feeds them back into the evaluation procedure if any improvement has been be gained. The rationale is to provide a computationally cheap way of determining the "average case" performance of the components, which is in turn be used to build effective sets of combinations.

Apart from serving as a pre-experiment filtering technique, the described simulation approach is a possible way of mechanizing (some of) the presented evaluation methods and realizing an automated strategy deployment support in a newly developed planning application for which no experiences concerning system performance exist.

In this context, it also becomes increasingly important to establish a more formal theory of compatibility of strategy components. When we will have developed a better understanding of the analytical processes of assessing quantitative strategy component characteristics, we will be able to define more precisely the concept of synergy in strategy combinations. We also believe that such a quantification of compatibility significantly contributes to our insights into strategy mechanisms, practically enhances the predictive power of the simulation/recombination approach above, and, last but not least, stimulates research into new strategies.

As a last topic, hybrid planning is only one system configuration amongst many. While our results can be expected to be easily transferable to pure HTN or POCL planning, there is no prospect of their applicability to scheduling or hybrid resource planning configurations. In absence of a suitable established methodology, we will in a first step repeat our experiments on resource-intense domains and problems; it will be an important part of our future research to define the appropriate experimental setups and evaluation measures.

6.7 Summary and Conclusion

This chapter has presented an empirical evaluation of strategy performance that we have conducted in our planning framework. We have given a precise definition for the concept of "performance" and accordingly formulated a set of research questions for guiding our experiments. By doing so we did not only motivate the study design but we could also demonstrate the bandwidth of issues that are addressable by means of experimentation in our proposed planning framework. We would like to point out that this includes two aspects, namely that our approach is not only a flexible framework for building a variety of planning systems and strategies but that it is also for conducting empirical evaluations at a larger scale. By adopting this view, the PANDA framework becomes a tool for identifying suitable system configurations systematically. It is also worth mentioning that the experimentation is not restricted to strategy components but open to all imaginable variations and alterations of the system configuration, domain models, and planning problems. For example, we may as well investigate into alternative modification generating modules that incorporate domain-specific knowledge. This flexibility in experimentation is a unique feature of our approach.

We have detailed our evaluation methods and interpreted the results with respect to the individual domain models as well as from a more consolidated domain-independent perspective. Our conclusive findings are, first of all, that empirical methods are key techniques for understanding many strategy mechanisms in general and in particular the interactions between planning systems and the respective problems and domains. Since we aimed at providing advice for the identification of an appropriate strategy for a given planning application, this result is, of course, less surprising, the evidence for the conjecture is however properly documented above. As our experiments have shown, predicting the performance of a strategy is extremely hard and often impossible, for example, the preference of constrained plans and its inversion take turns at being successful from domain to domain; it is therefore not only *possible* to evaluate a configuration, but also *necessary*.

The second major result is that our portfolio provides a rich assortment of search strategies, in particular from our novel flexible strategy classes, which have interesting properties: The HotSpot and HotZone concepts utilize flaw and modification relationships in an opportunistic and apparently fruitful way. Our hybrid-planning A^* plan selection is a successful example for taking into account POCL and HTN concepts in a seamlessly integrated manner. Both types of strategies can furthermore be easily combined with other components which makes them very valuable assets for our framework.

The last more general conclusion we can draw from our experiments is concerning the "by-products" of the evaluation procedure. We found out that basically all our results give evidence to the fact that hybrid planning as such is much more than a partial-order methodology with a "cheating" method to pump plan steps into a plan easily. This became particularly apparent when we examined the influence of the decisions concerning the treatment of decomposition plan-modifications. Furthermore, we became aware of the fact that experimental studies on planning systems also contribute to our understanding of the involved domain models and problems. This is because, on the one hand, we build hypotheses when observing the behaviour of search algorithms during the problem processing, on the other hand, these hypotheses may not be completely confirmed by subsequent experiments, which in turn initiates a very focused examination of the situation. These are naturally and very probably effects of problem characteristics the researcher was not aware of before.
It has to be noted that incidental findings of this kind in general have to be considered carefully, because the study may have an implicit bias concerning facts outside its design scope. In our research questions, however, the simple difficulty classification for planning problems has been included from the beginning.

We have also discussed our findings in terms of related work and future perspectives. It is important to note that a direct comparison between our results and related experiments in the original literature is neither intended nor adequate for several reasons: The referenced evaluations typically had a different research focus, worked on different domains, examined different plan generation paradigms, and operated on their specific system implementations. This is in particular the case for the SHOP system, which has been "realized" in our framework by lifting it to a purely declarative level. The classical strategies, as well as our findings regarding the least commitment principle, have rather to be viewed as representatives of specific search schemata.

This section concludes our evaluation chapter in which we have presented for the first time a systematic empirical study concerning strategies for hybrid planning. We have shown in an illustrative set of experiments how candidates for an adequate strategic component in a hybrid planning system can be identified. The application of these results is finally a methodology for compiling instances of our planning framework in an application-oriented (and possibly automated) way.

7 Summary and Conclusions

The previous chapters have presented the *formal basis* (Chap. 2) and the *conceptual means* (Chap. 3) of a planning framework that is on the one hand capable of integrating various planning methods in a coherent manner and that allows on the other hand to design and deploy *novel planning strategies* (Chap. 4) that effectively control these planning methods. We have also examined some relevant aspects of fielding a planning and scheduling system: We discussed the *architectural issues* that arise in a productive software environment and presented some representative *domain models* and their particularities with respect to hybrid planning (Chap. 5). The technical details have been rounded up by an *empirical evaluation* in which we have tested a collection of our strategies in use (Chap. 6). Each chapter has presented its specific conclusions and scientific perspective.

It is the duty of this concluding chapter to give a *summary of our contributions*, in particular in the light of the research agenda that we have proposed in the introductory chapter as a touchstone for our scientific results. The following sections will therefore briefly discuss each agenda item with regard to our achievements as well as suggest future developments that may build on our work.

7.1 Summary of Contributions

7.1.1 A Formal Framework for Planning and Scheduling

In Chapter 2 we have defined the formal semantics for all constituents of a domain model, in particular for plans and their components. Our approach is thereby the first formal treatment of purely declarative hierarchical domain models such that abstract actions are given a proper meaning in terms of concrete plans. On the one hand, this gives us the means to implement a sound hybrid hierarchical approach (see Section 7.1.2) and on the other hand, it provides a well-founded universal notion of model *consistency*.

On the basis of our model semantics, we understand planning problems as abstract, underspecified sketches of intended courses of action and accordingly solution plans as implementations of these sketches. The process of plan generation has been consequently realized as a formally well-founded method of transforming *abstract* plans into *concrete* solution plans in a *specification-preserving* manner. This technique is called *refinement planning* and we formulated a generic planning algorithm that uses it to systematically explore the space of possible refinements that is induced by an abstract problem in order to locate a concrete solution.

One novelty of our formal approach to refinement-planning lies in an explicit specification of refinement operators, called *plan modifications*. Any such operator that complies with the corresponding soundness conditions is thereby guaranteed to consistently constrain the implementation candidates of a given plan. This has two important implications: First, any type of plan refinement can be realized within the framework and safely integrated into the the algorithm, which contributes to the topics of Section7.1.3. Second, the refinement operators become examinable for strategic purposes or retrospect explanations.

Another unique feature of our work is to represent also plan deficiencies explicitly. We introduced *flaws* that identify the concrete cause for a plan to fail the solution criteria, that means, we categorize the problem and mark the affected plan components. Like for the plan modifications, flaws are provided with a concept of soundness such that any type of solution criterion can be realized at any level of detail. Consequently, flaws can be operationalized in a generic solution test as well as in more sophisticated analyses of plan defects. In this way, they are the basis for novel planning strategies (see Section 7.1.4).

Explicitly represented flaws and plan modifications are suitable means for describing the refinement planning procedure, but they are more than that: they can actually make a generic refinement planning algorithm operational. Based on the correlation between deficiency characteristics and the effects of refinement operators, we have proposed a *triggering function* that explicitly interprets the occurrence of certain flaws as justification for the application of certain plan modifications. Deployed in our generic algorithm, the triggering function induces a control flow that adapts to any constellation of supported flaw and plan modification classes. Chapter 3 and Section 5.1 proved this formal framework to be a resilient foundation that provides effective mechanisms for a component-based configuration and implementation of planning and scheduling functionality (see Section 7.1.5).

7.1.2 Integration of Hierarchical and Non-Hierarchical Methods

Chapter 2 presented the concept of abstraction as an integral part of our formal framework. It did so in two aspects: First, our approach makes use of *decomposition methods*, similar to those invented in hierarchical task-network (HTN) planning but with more elaborate semantics. In this view, a consistent decomposition method has to relate an abstract action to a partial-order plan such that the plan represents a valid, concrete implementation of the state transition that is specified by the abstract action. This mechanism is, secondly, accompanied by *state-abstraction axioms*, which are essentially a well-founded way of refining abstract state descriptions into more concrete state features.

The combination of these two forms of abstraction allows us to capture domain models that combine the expressiveness of procedural knowledge as in HTN planning with the causality-centered state-based view of partial-order planning – and it allows us to do so very elegantly. Section 5.2 detailed some example domains and discussed their particularities. Concerning the operational aspects of our approach, the integration of the two abstraction mechanisms realizes for the first time a *semantically safe*, hence *seamless merge* of hierarchical and non-hierarchical plan refinements. The hybrid system configurations that we described in Chapter 3 provide the corresponding flaw detection and plan modification generating components, thereby turning our generic refinement algorithm into a system that constructs plans in the tradition of partial-order planning and HTN planning likewise. This means in particular that the hybrid system does not need to deploy any fixed procedure with respect to synchronizing the abstraction levels in a plan, it may freely choose from the available development options. The decomposition semantics and state refinements guarantee that causality can be traced across multiple levels of abstraction.

7.1.3 Integration of Planning and Scheduling

On the basis of the formal framework in Chapter 2, and in particular of the therein defined term manipulation semantics, we have developed a collection of plan refinements for the well-founded manipulation of temporal and resource information. With the derived plan deficiencies and refinement operators, Chapter 3 proposed the essential generator components that implement the corresponding reasoning capabilities, thereby setting up system configurations that realize scheduling functionality. Our flaw and plan modification concept provides the semantic basis for a well-defined modularization as well as the technical means for an appropriate control flow; our generic refinement algorithm is therefore able to operate such that "planning" and "scheduling" plan modifications may *interleave arbitrarily* and consequently the correlated flaws can be addressed in an *opportunistic* fashion. Thus, we succeeded in completely integrating both technologies on the operational level.

Our work also contributed substantially to a *unified representation* of domain models for hybrid planning and scheduling. Incorporating temporal phenomena and resources into our semantics of state and action abstraction creates two innovations: For the first time, decomposition-based planning has been provided with clear semantics concerning duration and resource manipulation on the abstract action level. The correlated notion of consistency becomes universally applicable and therefore supports both constructing meaningful domain models and well-founded reasoning on these models. The second innovation lies in providing resource representations with a semantic dimension of abstraction. Our formal framework captures novel concepts like resource aggregation or explicit approximation, and we presented examples for the application potential of this novel *hierarchical scheduling* approach.

7.1.4 Flexible Planning Strategies

The above integration efforts focus on a common basis for constructing the space of possible refinements as it is provided by our framework in Chapter 2. The problem of actually *navigating* through that space in search of a solution is addressed in the generic refinement algorithm by an *explicit search strategy* that works in two steps: a local-scope strategic function that examines the refinement options for given plan and a global-scope search control that manages the plans that are under examination and that chooses the route through the refinement space.

These two components cooperatively direct the system as follows: Any plan that is not yet a solution is assigned flaws in order to name the deficiencies. This induces, via the triggering function, the construction of plan modifications in response to solve those defects, if possible. After that, as the first step of strategic reasoning, the *modification selection function* chooses which of the proposed refinement operators to apply, thereby deciding which of the possible refinements within the huge search space are to be instantiated. This choice can be made directly by prioritizing refinement classes or indirectly according to a preference of flaws that shall be eliminated earliest. The explicit representation of flaws and modifications, however, makes *much more* information available, including the relationships between the deficiencies, their resolving refinements, and the components of the plan to which both refer. As we have shown in Chapter 4, this kind of structural evidence can be effectively used for a heuristic assessment of the plan development status and thereby employed in a novel class of *flexible*, opportunistic strategies.

The newly created refinements are processed by the flaw and modification generators and then enqueued into the line-up of available solution candidates, the fringe of the explored search space. From there, a second strategic element chooses the plan that is to be processed next. This *plan selection function* can be realized as a classical search heuristic but also as a novel *flexible* one that utilizes the additional information about the deficiencies and refinement submissions for the fringe members.

Chapter 4 developed a broad portfolio of modification and plan-related strategy components that ranges from classical selection schemata to flexible heuristics. In particular the latter are *configuration independent* strategies and can therefore in particular be deployed in hybrid planning configurations as well as in system instances that perform scheduling operations. We furthermore designed the components such that modification and plan selection functions can be assembled, respectively, into more complex and powerful arrangements. We demonstrated in an evaluation in Chapter 6 that our novel strategic concepts are both elegant, modular specifications and adequate search controllers for hybrid planning.

7.1.5 An Effective Software Platform

Based on the generic refinement-planning algorithm in Chapter 2 we developed an open software architecture for the implementation of the system configuration concept that has been presented in Chapter 3. The highly modular nature of the configuration components is thereby emphasized by realizing them as a *multi-agent system*, which optimally supports the notion of concurrency during the parallel computations of flaws and modifications as well as the notion of distributed knowledge. While the former addresses the practical issue of manging multiple computational resources the latter matches nicely the idea of different modules representing different planning and scheduling aspects. In order to improve the non-functional performance features, the whole architecture is realized within a so-called *application server*, a standardized environment for deploying large-scale distributed business software systems. Section 5.1 explained the design issues and detailed the main software components.

Apart from these more technical implementation aspects, we were also able to enhance our planning framework conceptually by employing techniques that originate in the “Semantic Web” community. We translated some of our semantic concepts into an established ontology language, including the representations for flaws, modifications, and related system components. On this basis, the software platform is able deduce the concrete triggering relation of a planning system configuration from a high-level specification of the deployed agent modules and to install the effective data and control flow automatically. Furthermore, the declarative ontology language is expressive enough to capture all functional aspects of the platform, is an appropriate interchange format for many existing modeling tools and reasoning systems, and allows system

configurations to be verified for completeness or plausibility. The knowledge-based techniques perfectly convey the flexibility of the conceptual framework to a software environment.

Chapter 6 showed by conducting a large-scale empirical evaluation that such an implementation of our formal framework follows up on our promise to provide a software platform that supports building planning and scheduling systems in a plug-and-play fashion as well as it is able to serve as an effective experimental platform for planning and scheduling research. The proposed framework is thereby primarily targeted at identifying a suitable configuration for a given application domain such that the result can be compiled afterwards into the "real" productive system. The experiments have however shown that it is performing surprisingly well for being a research prototype and the compilation effort should therefore be not too big. In this sense, our work narrows the gap between the academic result and a practical application.

7.2 Future Work

Throughout this thesis, we gave suggestions for improvements, possible extensions, and future research. At the very end of the concluding chapter, we want to give a review of the general topics that may originate from our work and that we consider worthwhile to investigate.

7.2.1 Advanced Plan Generation Concepts

Hybrid Scheduling: As we have pointed out in the conclusions sections above, this work is focused on the representational side of integrating planning and scheduling, that means, a common domain model representation and a common operational framework. Future research will have to address the optimization aspects that come with resource reasoning. This implies in particular the specification of objective functions and the design of strategies that achieve the desired plan quality. The main open questions in this context will be how to guarantee some level of optimality on the basis of the existing strategy portfolio and how our abstraction mechanisms can be applied to objective functions. Based on our results in resource abstraction types, we believe that this may lead to novel concepts of specifying and measuring plan quality.

Planning under Uncertainty: Notwithstanding all advances of representation features, it is a key issue in all practical applications that they include a certain amount of *uncertain knowledge*. Our approach will definitely be extended in order to address the problem of uncertainty either directly or indirectly and it provides the prerequisites for both kinds of developments.

We understand *direct* uncertainty management as those techniques that anticipate unexpected events and take this into consideration in the plan generation process by prediction or preparation. One form of uncertainty prediction is employing a probabilistic model of the execution environment. The precise mathematical model of some state features' variability allows for effectively estimating the probability for a successful plan execution. It is also a nice property of that approach that the integration of probability distributions matches our notion of task abstraction such that the prediction of uncertain facts becomes traceably less precise on the more abstract action levels. Another important feature is that distribution parameters can be verified and iteratively updated during execution. We have shown this kind of uncertainty handling previously at an earlier stage of this work [24, 25]. Addressing uncertainty by preparation, that means, the integration of conditional plan steps and loop constructs is another interesting direction. Contingency planning has been successfully realized for non-hierarchical approaches, it would therefore be a worthwhile effort to investigate into extending the formal framework accordingly.

An *indirect* uncertainty management does not anticipate events beyond the precise environment specification and therefore has to react if the unexpected occurs. A conceptually simple extension of our framework would be to incorporate it into a continuous planning system, that means, a system that continuously updates a current solution and that produces recovery plans if the previous course of action fails. A more sophisticated technique is that of *plan repair*, which analyzes the circumstances that led to an execution problem and modifies the failed plan accordingly. Two points make plan repair an interesting perspective for our work:

First, repairing a plan is inherently connected with the notion of reusing it, that means that search effort is saved and the commitments a user had made to follow that plan are preserved as much as possible. Second, our explicit representation of flaws and plan modifications allows for a formally well-founded as well as efficient plan repair. We have given in [21] a prototypical repair procedure that is based on our framework: In a first phase, the failed portion of the plan induces a retraction of those plan modifications that contributed to the failure. The second phase operates on the assumption that all decisions that led to the previously failed solution but that did *not* contribute to the failure are still worthwhile options to find a new solution efficiently. The referenced paper describes in detail how the adaptive strategy is designed and what mechanisms guarantee that the new solution, the repaired plan, is not affected by the previous failure situation. It is important to note that repair functionality can be realized via additional flaw and plan modification definitions and a flexible strategy; it is therefore perfectly implementable in our framework.

7.2.2 Mixed Initiative Planning

This thesis developed a planning framework for *automated* planning and scheduling, that means, the instantiated system configurations solve a given problem specification automatically without further user intervention. We believe however, that the area of *mixed initiative* planning, which involves the human user in the plan generation process, is also a topic that we should to address in future work. The rationale for this approach is to create a synergy between the data processing capabilities of automated systems and the strategic reasoning competences of human beings. As a side-effect, the acceptance of planning technology by non-expert users increases.

We have developed evaluation prototypes in several smaller projects that act as mediators between the presented planning framework and user interfaces. Thanks to our explicit representations, it was possible to build abstraction layers for displaying, explaining, and interacting with flaws and plan modifications. The human user thereby acts as strategic advice for a semi-automated system that is concerned with propagating the user decisions, arranging the plan details, and giving feedback about the expected solution quality [114].

It is worth pointing out that a mixed initiative approach in addition substantially benefits from two fundamental design decisions: Our refinement planning framework operates on the plan space, that means that any intermediate solution candidate is a partial plan that can be communicated to the user, interpreted, and also directly compared to other plans. Navigating through the space of refinements corresponds to a human-comprehensible manipulation of the plan data structure. Hence, the progress of search can be directly followed and does not require per se further techniques beyond the abstraction of details. The second user-relevant design feature is the adoption of abstraction techniques in the planning formalism. The universal treatment of state and action abstraction can be directly applied to user interfaces in term of data representation as well as in terms of interaction mechanisms. We think that one promising direction of future research lies in identifying suitable abstractions for communicating plans and their refinement options. On this basis, we will be able to develop techniques for a system-supported explanation of causal structures and plan generation alternatives.

Further research into mixed initiative planning will enable our approach to present itself in a less technical, hence more end-user oriented way, thereby making planning technology substantially more accessible to domain-experts.

7.2.3 Construction of Consistent Domain Models and Problems

Consistency is a recurring topic throughout this thesis and an extremely relevant one with respect to practical applications. While it is obvious that an inconsistent domain model makes any planning approach worthless, the effort of creating a consistent one is however widely underestimated. The same holds for problem specifications, which the literature is only able to categorize into solvable and unsolvable ones; for practical purposes, not solving a problem is never an option, because the product just has to be delivered, a procedure has to be carried out, and so on.

Our work addressed these issue from the perspective of the formal semantics of our planning methodology and formulated consistency properties accordingly. A desirable future development is to investigate into mechanizing these consistency criteria such that adequate tool support for domain modelling can be established. We implemented a first prototype of such an editor (shown, for example, in Fig. 5.7) and had the experience that even the simplest background checks of consistency constitute a valuable aid.

Future work in this direction has also to take into account the aspects of the actual construction process. This includes the adaptation of a suitable knowledge engineering method and the identification of concrete support measures for the incremental nature of model construction. The former means establishing consistency by construction, the latter suggesting model adjustments in order to maintain or re-establish consistency.

Last but not least, tracking anomalies is also an interesting area that has not been addressed yet. At the moment, evaluating the adequacy of a domain model is limited to testing in the sense that the obtained solutions are analyzed off-line and compared to the, probably implicit, expectations. We believe that many of the "best practices" for domain models and problem specifications can be formalized as quality metrics and incorporated in a tool that, in analogy to programming tools, constitutes a *model development environment*.

A Weights In The Adaptive HotSpot Strategy

The following weights are examples for the user-defined weights in the adaptive HotSpot strategy as they have been used in our experiments with the hybrid planning configuration (cf. p. 167ff, strategy (4.27)). The provided comment sections have to be seen against the background of the respective flaw and modification combinations. Please note that a weight-value of 1.0 is the neutral factor, smaller values mean to prefer input flaw class A, greater values imply to favour input flaw class B.

Flaw Class A	Flaw Class B	Value	Comment
`Threat`	`Threat`	1.0	Fellow threats treated symmetrically.
	`AbstrTask`	1.5	Expansion offers additional threat resolution options.
	`OpenPrec`	0.5	Threat resolution changes linkability situation.
	`OpenVarBind`	1.0	Flaws treated symmetrically.
	`UnordTask`	0.25	Threat condsidered more crucial; may cause promotion or demotion.
`AbstrTask`	`Threat`	0.5	
	`AbstrTask`	1.0	
	`OpenPrec`	0.5	Expansion changes linkability situation; closing precondition potentially pre-mature.
	`OpenVarBind`	1.0	Flaws treated symmetrically.
	`UnordTask`	1.0	Flaws treated symmetrical.
`OpenPrec`	`Threat`	1.5	
	`AbstrTask`	1.5	
	`OpenPrec`	1.0	
	`OpenVarBind`	0.5	Implicit parameter binding imposes useful constraints.
	`UnordTask`	0.5	Linking eventually implies ordering.
`OpenVarBind`	`Threat`	1.0	
	`AbstrTask`	1.0	
	`OpenPrec`	1.5	
	`OpenVarBind`	2.0	Overlapping constraint manipulations considered harmful.
	`UnordTask`	1.0	Flaws treated symmetrical.
`UnordTask`	`Threat`	2.0	Threat handling usually better informed about suitable ordering.
	`AbstrTask`	1.0	
	`OpenPrec`	1.5	
	`OpenVarBind`	1.0	
	`UnordTask`	1.0	

B Empirical Evaluation – Data

The cell colour in the data tables on the following pages encodes the problem-related performance: Blue values lie in the first quartile, beige coloured values are between first quartile and median, and red values are above the third quartile or 100% failure runs.

ModSel	PlanSel	Problem 1o-1s-1m	2o-1s-1m	2o-2s-1m	2o-2s-2m	3o-1s-1m	3o-2s-3m	3o-3s-1m	3o-3s-3m
ems_lcf	cloca	70							
hz_lcf	cloca	65	1990						
lcf_da	cloca	63	2287						
lcf_du	cloca	78	2767						
lcf_ems	cloca	81	2741						
lcf_hz	cloca	65	2210						
lcf_ia	cloca	62	2039						
lcf_iu	cloca	63	2359						
lcf_mod	cloca	87	2916						
lcf_pExp	cloca	80	2822						
ems	df	34	864	809	800				
shop	df	50	758	237	4194	3715			
umcp	df	50	1535	2403	1500				
ems_lcf	du_fmf	31		582	683				
hz_lcf	du_fmf	37	3581	1300	1073				
lcf_da	du_fmf	39		453					
lcf_du	du_fmf	43		2162					
lcf_ems	du_fmf	45	3652	3895					
lcf_hz	du_fmf	37		2076					
lcf_ia	du_fmf	33		716					
lcf_iu	du_fmf	39			453				
lcf_mod	du_fmf	46		944					
lcf_pExp	du_fmf	43		1924					
umcp	ff(ct)	65	600	1495					
ems_lcf	fhz_fmf	77	841	502	873				
hz_lcf	fhz_fmf	61	1256	3954					
lcf_da	fhz_fmf	62	966	565					
lcf_du	fhz_fmf	78	1354	1989					
lcf_ems	fhz_fmf	70	1475	2512					
lcf_hz	fhz_fmf	60	1197						
lcf_ia	fhz_fmf	59	1151	349	4019				
lcf_iu	fhz_fmf	57	1410						
lcf_mod	fhz_fmf	84	1569						
lcf_pExp	fhz_fmf	79	1512	3267					
hz_lcf_iu	fhz_lcp	60	884	3397					
hz_lcf	fhz_lcp_fmf	59	1052	5026					
hz_lcf_iu	fhz_mcp	58	1251	3265					
hz_lcf	fhz_mcp_fmf	58	1603						
ems_lcf	fmh_fmf	32	679	2019		2322			
hz_lcf	fmh_fmf	38	400						
lcf_da	fmh_fmf	39	596	175				1256	
lcf_du	fmh_fmf	62	1051					824	
lcf_ems	fmh_fmf	63	879						
lcf_hz	fmh_fmf	42	699						
lcf_ia	fmh_fmf	35	639	144	5182			1328	
lcf_iu	fmh_fmf	38	635						
lcf_mod	fmh_fmf	67	1075						
lcf_pExp	fmh_fmf	54	829	1317				1273	
ems_lcf	iu_fmf	31		1856	1112				
hz_lcf	iu_fmf	38	4658	4215	1388				
lcf_da	iu_fmf	44		247					
lcf_du	iu_fmf	52			1618				
lcf_ems	iu_fmf	51			1500				
lcf_hz	iu_fmf	41		2141	1069				
lcf_ia	iu_fmf	42		436	1048				
lcf_iu	iu_fmf	42							
lcf_mod	iu_fmf	58		884	1872				
lcf_pExp	iu_fmf	50		3665					
hz_lcf_iu	lcp_fhz	48	2163	647	207				
hz_lcf	lcp_fhz_fmf	41	1770	890	176				
ems_lcf	lhz_fmf	26		1342	1405				
hz_lcf	lhz_fmf	35	2387	2402	1221			336	
lcf_da	lhz_fmf	37			1830				
lcf_du	lhz_fmf	50			206				
lcf_ems	lhz_fmf	41		744	493				
lcf_hz	lhz_fmf	33			2002				
lcf_ia	lhz_fmf	37			1105				
lcf_iu	lhz_fmf	37			1306				
lcf_mod	lhz_fmf	52		843	923				
lcf_pExp	lhz_fmf	43		1145	665				
hz_lcf_iu	mcp_fhz	64							
hz_lcf	mcp_fhz_fmf	61	1722						
ems_lcf	psaoca	55	467	295	568			818	
hz_lcf	psaoca	53	717	3633					
lcf_da	psaoca	44	928	893	3014				
lcf_du	psaoca	67	783						
lcf_ems	psaoca	62	973	4747				1056	
lcf_hz	psaoca	55	833	4564					
lcf_ia	psaoca	49	896	446					
lcf_iu	psaoca	52	678	2735					
lcf_mod	psaoca	68	1218						
lcf_pExp	psaoca	65	1084	2504					
shop	psaoca	51	818						
ems_lcf	sdr_fmf	60	432	307	615				
hz_lcf	sdr_fmf	46	464	3706					
lcf_da	sdr_fmf	46	944	334					
lcf_du	sdr_fmf	62	967	3284					
lcf_ems	sdr_fmf	54	1036	2219					
lcf_hz	sdr_fmf	46	667	3320	3696				
lcf_ia	sdr_fmf	48	642	273					
lcf_iu	sdr_fmf	43	899	2429	4475				
lcf_mod	sdr_fmf	64	663	3611					
lcf_pExp	sdr_fmf	56	1253	3423					
	Min-Avg	26	400	144	176	2322	--	336	--
	1. Quartile	42	783	614	683	2670	--	821	--
	Median	52	1051	1856	1112	3018	--	1056	--
	3. Quartile	62	1603	3266	1830	3367	--	1264	--
	Max-Avg	87	4658	5026	5182	3715	--	1328	--

Table B.1: The **arithmetic means** of the explored search space sizes by the strategy combinations (rows) in the problem instances of the ***Satellite*** domain (columns).

ModSel	PlanSel	Problem 1o-1s-1m	2o-1s-1m	2o-2s-1m	2o-2s-2m	3o-1s-1m	3o-2s-3m	3o-3s-1m	3o-3s-3m
ems_lcf	cloca	6							
hz_lcf	cloca	13	21048						
lcf_da	cloca	15	390233						
lcf_du	cloca	18	95731						
lcf_ems	cloca	16	272257						
lcf_hz	cloca	14	453773						
lcf_ia	cloca	21	121761						
lcf_iu	cloca	6	266388						
lcf_mod	cloca	25	46745						
lcf_pExp	cloca	34	229462						
ems	df	108	1025001	660173	861821				
shop	df	326	391304	0	1713101	0			
umcp	df	805	2061651	9232526	2937912				
ems_lcf	du_fmf	0		15036	0				
hz_lcf	du_fmf	80	0	924800	4418				
lcf_da	du_fmf	70		0					
lcf_du	du_fmf	33		3251250					
lcf_ems	du_fmf	110	0	0					
lcf_hz	du_fmf	80		0					
lcf_ia	du_fmf	20		77881					
lcf_iu	du_fmf	120			0				
lcf_mod	du_fmf	61		0					
lcf_pExp	du_fmf	37		0					
umcp	ff(ct)	75	83987	1314591					
ems_lcf	fhz_fmf	7	91257	84505	13122				
hz_lcf	fhz_fmf	11	485319	439674					
lcf_da	fhz_fmf	18	223026	5952					
lcf_du	fhz_fmf	58	795761	0					
lcf_ems	fhz_fmf	19	621344	6613					
lcf_hz	fhz_fmf	31	302969						
lcf_ia	fhz_fmf	21	137509	4945	88200				
lcf_iu	fhz_fmf	15	397514						
lcf_mod	fhz_fmf	67	324163						
lcf_pExp	fhz_fmf	68	254673	4703784					
hz_lcf_iu	fhz_lcp	9	32623	186379					
hz_lcf	fhz_lcp_fmf	15	88184	0					
hz_lcf_iu	fhz_mcp	8	486672	0					
hz_lcf	fhz_mcp_fmf	7	174316						
ems_lcf	fmh_fmf	0	6380	34193		376519			
hz_lcf	fmh_fmf	37	7918						
lcf_da	fmh_fmf	28	27603	1553				45193	
lcf_du	fmh_fmf	15	37471					60915	
lcf_ems	fmh_fmf	6	34869						
lcf_hz	fmh_fmf	21	16522						
lcf_ia	fmh_fmf	3	26381	807	0			169422	
lcf_iu	fmh_fmf	21	7862						
lcf_mod	fmh_fmf	117	80269						
lcf_pExp	fmh_fmf	5	19310	0				32258	
ems_lcf	iu_fmf	0		0	79385				
hz_lcf	iu_fmf	129	51200	0	647068				
lcf_da	iu_fmf	84		0					
lcf_du	iu_fmf	210			0				
lcf_ems	iu_fmf	176			0				
lcf_hz	iu_fmf	100		2066420	17298				
lcf_ia	iu_fmf	117		1201	0				
lcf_iu	iu_fmf	117							
lcf_mod	iu_fmf	135		0	0				
lcf_pExp	iu_fmf	142		0					
hz_lcf_iu	lcp_fhz	175	686995	0	0				
hz_lcf	lcp_fhz_fmf	203	51977	1350124	0				
ems_lcf	lhz_fmf	0		643702	740556				
hz_lcf	lhz_fmf	30	0	2618293	522582			0	
lcf_da	lhz_fmf	30			288				
lcf_du	lhz_fmf	64			0				
lcf_ems	lhz_fmf	41		26558	0				
lcf_hz	lhz_fmf	20			0				
lcf_ia	lhz_fmf	30			371952				
lcf_iu	lhz_fmf	30			859361				
lcf_mod	lhz_fmf	40		13945	0				
lcf_pExp	lhz_fmf	22		540800	0				
hz_lcf_iu	mcp_fhz	0							
hz_lcf	mcp_fhz_fmf	33	54450						
ems_lcf	psaoca	212	10631	1885	19532			16570	
hz_lcf	psaoca	140	85662	0					
lcf_da	psaoca	179	178777	80678	0				
lcf_du	psaoca	212	21638						
lcf_ems	psaoca	325	138227	0				0	
lcf_hz	psaoca	253	42635	0					
lcf_ia	psaoca	126	21447	19726					
lcf_iu	psaoca	108	13345	453462					
lcf_mod	psaoca	191	319376						
lcf_pExp	psaoca	149	32813	2597491					
shop	psaoca	227	9322						
ems_lcf	sdr_fmf	0	382	72	299151				
hz_lcf	sdr_fmf	11	83174	1050997					
lcf_da	sdr_fmf	29	133916	4069					
lcf_du	sdr_fmf	24	116147	3301008					
lcf_ems	sdr_fmf	16	336694	3129676					
lcf_hz	sdr_fmf	29	109915	238342	0				
lcf_ia	sdr_fmf	18	198705	348					
lcf_iu	sdr_fmf	7	100125	3760384	152190				
lcf_mod	sdr_fmf	28	157936	280501					
lcf_pExp	sdr_fmf	13	250266	860322					
	Min-Avg	0	0	0	0	0	--	0	--
	1. Quartile	15	27603	0	0	94130	--	8285	--
	Median	30	95731	13945	288	188259	--	32258	--
	3. Quartile	110	266388	651937	299151	282389	--	53054	--
	Max-Avg	805	2061651	9232526	2937912	376519	--	169422	--

Table B.2: The **sample variance** of the explored search space sizes by the strategy combinations (rows) in the problem instances of the ***Satellite*** domain (columns).

ModSel	PlanSel	Problem 1s-1o-1m	2o-1s-1m	2o-2s-1m	2o-2s-2m	3o-1s-1m	3o-2s-3m	3o-3s-1m	3o-3s-3m	Avg
ems_lcf	cloca	0%	100%	100%	100%	100%	100%	100%	100%	88%
hz_lcf	cloca	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_da	cloca	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_du	cloca	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_ems	cloca	0%	40%	100%	100%	100%	100%	100%	100%	80%
lcf_hz	cloca	0%	20%	100%	100%	100%	100%	100%	100%	78%
lcf_ia	cloca	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_iu	cloca	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_mod	cloca	0%	40%	100%	100%	100%	100%	100%	100%	80%
lcf_pExp	cloca	0%	0%	100%	100%	100%	100%	100%	100%	75%
ems	df	0%	40%	0%	0%	100%	100%	100%	100%	55%
shop	df	0%	40%	80%	60%	80%	100%	100%	100%	70%
umcp	df	0%	20%	40%	40%	100%	100%	100%	100%	63%
ems_lcf	du_fmf	0%	100%	40%	80%	100%	100%	100%	100%	78%
hz_lcf	du_fmf	0%	80%	60%	60%	100%	100%	100%	100%	75%
lcf_da	du_fmf	0%	100%	80%	100%	100%	100%	100%	100%	85%
lcf_du	du_fmf	0%	100%	60%	100%	100%	100%	100%	100%	83%
lcf_ems	du_fmf	0%	80%	80%	100%	100%	100%	100%	100%	83%
lcf_hz	du_fmf	0%	100%	80%	100%	100%	100%	100%	100%	85%
lcf_ia	du_fmf	0%	100%	40%	100%	100%	100%	100%	100%	80%
lcf_iu	du_fmf	0%	100%	100%	80%	100%	100%	100%	100%	85%
lcf_mod	du_fmf	0%	100%	80%	100%	100%	100%	100%	100%	85%
lcf_pExp	du_fmf	0%	100%	80%	100%	100%	100%	100%	100%	85%
umcp	ff(ct)	0%	0%	0%	100%	100%	100%	100%	100%	63%
ems_lcf	fhz_fmf	0%	0%	0%	60%	100%	100%	100%	100%	58%
hz_lcf	fhz_fmf	0%	0%	40%	100%	100%	100%	100%	100%	68%
lcf_da	fhz_fmf	0%	0%	0%	100%	100%	100%	100%	100%	63%
lcf_du	fhz_fmf	0%	0%	80%	100%	100%	100%	100%	100%	73%
lcf_ems	fhz_fmf	0%	0%	60%	100%	100%	100%	100%	100%	70%
lcf_hz	fhz_fmf	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_ia	fhz_fmf	0%	0%	0%	60%	100%	100%	100%	100%	58%
lcf_iu	fhz_fmf	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_mod	fhz_fmf	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_pExp	fhz_fmf	0%	0%	20%	100%	100%	100%	100%	100%	65%
hz_lcf_iu	fhz_lcp	0%	0%	20%	100%	100%	100%	100%	100%	65%
hz_lcf	fhz_lcp_fmf	0%	0%	80%	100%	100%	100%	100%	100%	73%
hz_lcf_iu	fhz_mcp	0%	0%	80%	100%	100%	100%	100%	100%	73%
hz_lcf	fhz_mcp_fmf	0%	0%	100%	100%	100%	100%	100%	100%	75%
ems_lcf	fmh_fmf	0%	0%	0%	100%	0%	100%	100%	100%	50%
hz_lcf	fmh_fmf	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_da	fmh_fmf	0%	0%	0%	100%	100%	100%	0%	100%	50%
lcf_du	fmh_fmf	0%	0%	100%	100%	100%	100%	20%	100%	65%
lcf_ems	fmh_fmf	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_hz	fmh_fmf	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_ia	fmh_fmf	0%	0%	0%	80%	100%	100%	40%	100%	53%
lcf_iu	fmh_fmf	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_mod	fmh_fmf	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_pExp	fmh_fmf	0%	0%	80%	100%	100%	100%	60%	100%	68%
ems_lcf	iu_fmf	0%	100%	80%	0%	100%	100%	100%	100%	73%
hz_lcf	iu_fmf	0%	60%	80%	20%	100%	100%	100%	100%	70%
lcf_da	iu_fmf	0%	100%	80%	100%	100%	100%	100%	100%	85%
lcf_du	iu_fmf	0%	100%	100%	80%	100%	100%	100%	100%	85%
lcf_ems	iu_fmf	0%	100%	100%	80%	100%	100%	100%	100%	85%
lcf_hz	iu_fmf	0%	100%	40%	60%	100%	100%	100%	100%	75%
lcf_ia	iu_fmf	0%	100%	60%	80%	100%	100%	100%	100%	80%
lcf_iu	iu_fmf	0%	100%	100%	100%	100%	100%	100%	100%	88%
lcf_mod	iu_fmf	0%	100%	80%	80%	100%	100%	100%	100%	83%
lcf_pExp	iu_fmf	0%	100%	80%	100%	100%	100%	100%	100%	85%
hz_lcf_iu	lcp_fhz	0%	0%	80%	80%	100%	100%	100%	100%	70%
hz_lcf	lcp_fhz_fmf	0%	0%	40%	80%	100%	100%	100%	100%	65%
ems_lcf	lhz_fmf	0%	100%	0%	0%	100%	100%	100%	100%	63%
hz_lcf	lhz_fmf	0%	80%	20%	40%	100%	100%	80%	100%	65%
lcf_da	lhz_fmf	0%	100%	100%	60%	100%	100%	100%	100%	83%
lcf_du	lhz_fmf	0%	100%	100%	80%	100%	100%	100%	100%	85%
lcf_ems	lhz_fmf	0%	100%	40%	80%	100%	100%	100%	100%	78%
lcf_hz	lhz_fmf	0%	100%	100%	80%	100%	100%	100%	100%	85%
lcf_ia	lhz_fmf	0%	100%	100%	0%	100%	100%	100%	100%	75%
lcf_iu	lhz_fmf	0%	100%	100%	60%	100%	100%	100%	100%	83%
lcf_mod	lhz_fmf	0%	100%	60%	80%	100%	100%	100%	100%	80%
lcf_pExp	lhz_fmf	0%	100%	60%	80%	100%	100%	100%	100%	80%
hz_lcf_iu	mcp_fhz	0%	100%	100%	100%	100%	100%	100%	100%	88%
hz_lcf	mcp_fhz_fmf	0%	60%	100%	100%	100%	100%	100%	100%	83%
ems_lcf	psaoca	0%	0%	0%	0%	100%	100%	40%	100%	43%
hz_lcf	psaoca	0%	0%	80%	100%	100%	100%	100%	100%	73%
lcf_da	psaoca	0%	0%	20%	80%	100%	100%	100%	100%	63%
lcf_du	psaoca	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_ems	psaoca	0%	0%	80%	100%	100%	100%	80%	100%	70%
lcf_hz	psaoca	0%	0%	80%	100%	100%	100%	100%	100%	73%
lcf_ia	psaoca	0%	0%	0%	100%	100%	100%	100%	100%	63%
lcf_iu	psaoca	0%	0%	0%	100%	100%	100%	100%	100%	63%
lcf_mod	psaoca	0%	0%	100%	100%	100%	100%	100%	100%	75%
lcf_pExp	psaoca	0%	0%	20%	100%	100%	100%	100%	100%	65%
shop	psaoca	0%	0%	100%	100%	100%	100%	100%	100%	75%
ems_lcf	sdr_fmf	0%	0%	0%	0%	100%	100%	100%	100%	50%
hz_lcf	sdr_fmf	0%	0%	20%	100%	100%	100%	100%	100%	65%
lcf_da	sdr_fmf	0%	0%	0%	100%	100%	100%	100%	100%	63%
lcf_du	sdr_fmf	0%	0%	20%	100%	100%	100%	100%	100%	65%
lcf_ems	sdr_fmf	0%	0%	20%	100%	100%	100%	100%	100%	65%
lcf_hz	sdr_fmf	0%	0%	40%	80%	100%	100%	100%	100%	65%
lcf_ia	sdr_fmf	0%	0%	0%	100%	100%	100%	100%	100%	63%
lcf_iu	sdr_fmf	0%	0%	20%	40%	100%	100%	100%	100%	58%
lcf_mod	sdr_fmf	0%	0%	60%	100%	100%	100%	100%	100%	70%
lcf_pExp	sdr_fmf	0%	0%	0%	100%	100%	100%	100%	100%	63%
	Min-F-Ratio	0%	0%	0%	0%	0%	100%	0%	100%	**43%**
	1. Quartile	0%	0%	20%	80%	100%	100%	100%	100%	**65%**
	Median	0%	0%	80%	100%	100%	100%	100%	100%	**75%**
	3. Quartile	0%	100%	100%	100%	100%	100%	100%	100%	**80%**
	Max-F-Ratio	0%	100%	100%	100%	100%	100%	100%	100%	**88%**
	Average	0%	36%	63%	84%	99%	100%	96%	100%	**72%**

Table B.3: The **failure ratio** of the strategy combinations (rows) in the problem instances of the ***Satellite*** domain (columns).

ModSel	PlanSel	Problem 2-Regular Truck	Airplane	Armored Regular Truck	Auto Traincar	Auto Traincar bis	Auto Truck	Flatbed Truck	Hopper Truck	Mail Traincar	Refrigerated Traincar	Regular Truck	Regular Truck - 3Loc	Tanker Truck
ems_lcf	cloca													
hz_lcf	cloca		858	124		2316	541	279	221	396	484	124	323	174
lcf_da	cloca		1793	102		1174	274	255	184	911	845	85	546	250
lcf_du	cloca		2625	94		1081	352	293	222	957	1006	109	591	250
lcf_ems	cloca		2696	162		1826	510	681	361	1806	1947	167	1159	506
lcf_hz	cloca		2562	107		1077	324	283	219	1020	917	128	619	258
lcf_ia	cloca		1801	81		782	305	244	182	625	599	80	538	237
lcf_iu	cloca		2299	88		832	321	271	193	703	684	86	596	239
lcf_mod	cloca		2442	84		1072	302	274	178	905	915	81	592	228
lcf_pExp	cloca		2720	196		1742	565	779	413	1560	1548	166	1413	429
ems	df	852	99	274	180	1266	623	229	480	1309	1700	315	110	149
shop	df	321	88	188	170	821	1084	249	402	337	233	403	123	219
umcp	df	949	152	394	353	280	679	856	698	2626	1315	723	182	130
ems_lcf	du_fmf													
hz_lcf	du_fmf			290			1024	341	263	1981	1992	278	320	270
lcf_da	du_fmf			260			788	280	262	1563	1648	261	344	291
lcf_du	du_fmf			288			811	352	255	1967	1970	287	389	317
lcf_ems	du_fmf			630			1393	612	463	2331	2242	625	708	624
lcf_hz	du_fmf			289			754	339	255	1654	1678	291	397	302
lcf_ia	du_fmf			257			775	278	257	1383	1453	255	349	277
lcf_iu	du_fmf			280			768	324	251	1457	1378	282	395	293
lcf_mod	du_fmf			243			819	332	253	1554	1515	282	396	278
lcf_pExp	du_fmf			627			1043	747	541	2357	2535	425	1001	554
umcp	ff(ct)	1367	314	156	395	451	327	193	194	273	313	182	202	151
ems_lcf	fhz_fmf			467	2626	1762	601	2783	621	348	448	587	1210	376
hz_lcf	fhz_fmf	2685	240	77		377	221	143	75	90	91	71	194	68
lcf_da	fhz_fmf	951	98	78		515	109	91	79	503	436	79	212	88
lcf_du	fhz_fmf	855	154	82		537	110	97	78	691	856	82	245	93
lcf_ems	fhz_fmf	1425	185	129		735	155	150	131	829	999	116	249	117
lcf_hz	fhz_fmf	609	176	82		610	109	92	80	708	701	82	248	90
lcf_ia	fhz_fmf	1067	90	78		425	107	87	76	525	451	76	213	85
lcf_iu	fhz_fmf	846	168	82		421	108	93	76	552	545	80	244	85
lcf_mod	fhz_fmf	377	111	73		601	102	85	73	731	900	74	243	80
lcf_pExp	fhz_fmf	1639	176	140		967	159	123	131	1305	1235	120	304	110
hz_lcf_iu	fhz_lcp	1969	413	76		526	222	132	74	89	89	74	154	65
hz_lcf	fhz_lcp_fmf	2989	287	74		410	202	127	73	92	90	73	174	68
hz_lcf_iu	fhz_mcp		255	76		370	229	135	74	89	89	74	184	68
hz_lcf	fhz_mcp_fmf		326	75		392	225	151	73	91	91	75	194	73
ems_lcf	fmh_fmf			879			4677		4524		2808	4389	1050	3259
hz_lcf	fmh_fmf	1070	132	74		346	196	101	79	87	135	90	158	107
lcf_da	fmh_fmf	1765	158	75	470	184	104	82	74	129	195	72	207	82
lcf_du	fmh_fmf		153	74	436	331	97	83	65	206	278	68	124	100
lcf_ems	fmh_fmf	1953	171	117	453	433	129	140	134	419	379	114	256	139
lcf_hz	fmh_fmf		155	71	393	293	101	77	66	233	239	73	144	103
lcf_ia	fmh_fmf	1471	146	69	501	284	100	77	69	254	252	70	208	75
lcf_iu	fmh_fmf	1263	146	70	315	291	95	76	64	249	311	66	122	71
lcf_mod	fmh_fmf	385	679	73	607	476	152	129	121	468	455	122	114	127
lcf_pExp	fmh_fmf	2173	208	119	757	414	134	169	121	434	217	131	341	160
ems_lcf	iu_fmf													
hz_lcf	iu_fmf			257			1038	344	280	2015	2036	278	303	250
lcf_da	iu_fmf			258			767	287	260	1699	1753	255	344	284
lcf_du	iu_fmf			290			755	351	251	1936	1915	283	393	318
lcf_ems	iu_fmf			558			1216	668	600	2277	2293	423	798	595
lcf_hz	iu_fmf			290			750	331	255	1596	1604	290	397	304
lcf_ia	iu_fmf			252			769	279	256	1310	1308	256	345	280
lcf_iu	iu_fmf			280			747	328	249	1306	1474	284	393	293
lcf_mod	iu_fmf			244			750	336	253	1504	1450	284	386	278
lcf_pExp	iu_fmf			590			1526	678	522	2342	2402	567	718	556
hz_lcf_iu	lcp_fhz		1442	237			846	273	240	1483	1724	229	163	244
hz_lcf	lcp_fhz_fmf			237			827	276	223	1772	1783	238	136	227
ems_lcf	lhz_fmf			2021			3447	2467	3068			2965	3147	2023
hz_lcf	lhz_fmf			256			986	308	255	1399	1850	267	313	245
lcf_da	lhz_fmf			257			767	286	256	1677	1612	260	328	294
lcf_du	lhz_fmf			281			751	344	258	2278	1959	285	386	320
lcf_ems	lhz_fmf			483			1392	722	548	2561	2118	498	796	538
lcf_hz	lhz_fmf			292	3290		757	328	260	1880	1698	294	395	303
lcf_ia	lhz_fmf			254			765	281	256	1377	1465	252	338	278
lcf_iu	lhz_fmf			281			746	324	254	1474	1433	282	381	289
lcf_mod	lhz_fmf			244			750	322	254		1622	282	395	276
lcf_pExp	lhz_fmf			521			1430	645	411	2780		569	710	628
hz_lcf_iu	mcp_fhz		1669	224			735	265	206	1129	844	212	291	159
hz_lcf	mcp_fhz_fmf		1683	229			798	294	225	1034	1110	229	332	183
ems_lcf	psaoca			1007			1619		1552			1559	2509	1242
hz_lcf	psaoca		339	82		1018	613	199	112	211	229	118	272	86
lcf_da	psaoca	723	208	79		741	160	134	82	708	676	81	327	90
lcf_du	psaoca	1109	315	213		853	428	191	156	765	776	161	356	101
lcf_ems	psaoca	1812	337	137		1540	215	252	136	1496	880	116	494	141
lcf_hz	psaoca	767	319	132	2215	768	159	173	186	777	718	128	354	102
lcf_ia	psaoca	717	200	79	2937	500	147	147	78	424	421	78	329	85
lcf_iu	psaoca		307	208	1936	497	801	183	187	459	461	210	359	96
lcf_mod	psaoca	342	183	75		847	151	120	75	821	780	77	352	87
lcf_pExp	psaoca	1602	358	175		1418	186	320	142	1248	1596	143	520	150
shop	psaoca		489	708		3228	767	682	724	2792	3115	693	1227	517
ems_lcf	sdr_fmf	919	481	125	253	208	317	768	308	183	185	314	381	114
hz_lcf	sdr_fmf	729	135	77		295	90	126	71	94	89	75	153	68
lcf_da	sdr_fmf	472	93	218		123	110	89	79	720	1138	80	209	245
lcf_du	sdr_fmf	585	102	79		122	110	99	80	942	989	81	244	93
lcf_ems	sdr_fmf	805	134	194		173	155	116	129	1001	1316	122	297	228
lcf_hz	sdr_fmf	486	101	76		128	108	94	79	915	506	80	248	89
lcf_ia	sdr_fmf	487	91	255		122	107	88	78	301	1184	77	211	277
lcf_iu	sdr_fmf	591	99	75		118	110	91	75	504	909	79	244	88
lcf_mod	sdr_fmf	347	87	73		113	102	84	73	955	785	73	244	272
lcf_pExp	sdr_fmf	952	112	86		137	145	129	125	841	1393	124	282	105
	Min-Avg	321	87	69	170	113	90	76	64	87	89	66	110	65
	1. Quartile	600	138	79	363	294	151	127	79	440	453	80	220	97
	Median	919	204	181	462	499	390	260	200	929	989	152	331	227
	3. Quartile	1448	464	270	1641	881	768	329	257	1558	1617	283	397	290
	Max-Avg	2989	2720	2021	3290	3228	4677	2783	4524	2792	3115	4389	3147	3259

Table B.4: The **arithmetic means** of the explored search space sizes by the strategy combinations (rows) in the problem instances of the ***UM Translog*** domain (columns).

ModSel	PlanSel	Problem 2 - Regular Truck	Airplane	Armored Regular Truck	Auto Traincar	Auto Traincar - bis	Auto Truck	Flatbed Truck	Hopper Truck	Mail Traincar	Refrigerated Traincar	Regular Truck	Regular Truck - 3Loc	Tanker Truck
ems_lcf	cloca													
hz_lcf	cloca		19489	1146		235541	15273	1069	455	9202	31030	605	33	49
lcf_da	cloca		247	828		33777	922	67	8	3728	18590	25	6	22
lcf_du	cloca		582	96		12914	2215	67	166	45869	23669	1808	28	89
lcf_ems	cloca		1814	3580		448616	22908	61953	19178	390212	338944	3508	196394	35494
lcf_hz	cloca		3110	284		2933	879	38	375	6303	27498	1875	49	37
lcf_ia	cloca		279	32		1390	1005	32	15	14540	11079	23	86	24
lcf_iu	cloca		2777	1		32813	174	16	502	16904	16789	13	16	15
lcf_mod	cloca		30	232		10513	255	6	239	9400	13286	26	63	2
lcf_pExp	cloca		0	3245		469917	58936	33744	12091	409423	437182	5369	220872	31460
ems	df	102152	48	42304	369	2322013	1126643	64838	47566	0	0	84946	122	12377
shop	df	0	2	23719	648	1451857	1664186	23269	197877	38916	41103	88635	2091	34342
umcp	df	221	120	162723	50	8845	915248	1997758	203219	0	2631218	801408	328	92
ems_lcf	du_fmf													
hz_lcf	du_fmf			755			155	128	586	59457	52872	777	7	191
lcf_da	du_fmf			31			900	12	9	679	20168	15	22	37
lcf_du	du_fmf			7			6996	61	14	46033	32561	27	14	37
lcf_ems	du_fmf			22994			216560	54269	22256	0	2521	21055	63740	40124
lcf_hz	du_fmf			28			28	201	5	33206	35164	82	18	124
lcf_ia	du_fmf			17			279	22	25	22876	35553	26	10	9
lcf_iu	du_fmf			45			632	6	17	41785	29975	10	9	51
lcf_mod	du_fmf			1			8870	0	0	0	0	4	0	16
lcf_pExp	du_fmf			22184			125923	57945	26981	4676	88163	20372	1247	42564
umcp	ff(ct)	71171	1949	1180	4030	23783	6366	132	118	3984	5224	477	72	426
ems_lcf	fhz_fmf			3301	329672	94331	3593	4825	559	6757	59839	4532	980	28853
hz_lcf	fhz_fmf	0	1189	1		21700	34	359	2	7	13	32	514	6
lcf_da	fhz_fmf	37918	7	2		19916	5	8	3	13572	3261	6	10	3
lcf_du	fhz_fmf	97256	107	5		50626	7	23	9	35773	77089	12	7	59
lcf_ems	fhz_fmf	585610	3542	747		44757	650	4142	255	72585	247294	527	2292	571
lcf_hz	fhz_fmf	19631	2670	3		31486	5	18	13	36915	76439	2	5	6
lcf_ia	fhz_fmf	67178	8	3		576	4	6	7	12289	12154	8	4	1
lcf_iu	fhz_fmf	145383	1707	4		285	6	2	6	2052	6337	5	5	5
lcf_mod	fhz_fmf	63	2001	1		26625	1	0	1	91538	77304	0	4	9
lcf_pExp	fhz_fmf	460356	2749	585		145376	471	607	393	420625	533727	432	372	789
hz_lcf_iu	fhz_lcp	252828	19476	0		0	0	7	0	0	0	0	2734	0
hz_lcf	fhz_lcp_fmf	0	3722	17		12376	568	269	27	14	17	20	745	8
hz_lcf_iu	fhz_mcp		4958	0		48805	3	9	0	0	0	0	333	4
hz_lcf	fhz_mcp_fmf		3950	15		40234	76	498	22	16	15	0	67	3
ems_lcf	fmh_fmf			866			82722		74703		0	85694	5296	41867
hz_lcf	fmh_fmf	0	543	20		491	216	4222	253	67	1041	2000	3806	7708
lcf_da	fmh_fmf	542236	129	39	150	7942	18	20	29	5492	8578	15	66	58
lcf_du	fmh_fmf		160	14	32391	23394	48	115	4	4159	20765	10	58	1009
lcf_ems	fmh_fmf	0	2305	605	23686	16250	81	802	997	18075	21100	1057	3219	3226
lcf_hz	fmh_fmf		191	29	6008	1570	14	40	24	31	16	61	67	1609
lcf_ia	fmh_fmf	146536	67	4	4554	24	6	8	3	11	87	6	57	7
lcf_iu	fmh_fmf	42307	139	0	156	4828	48	30	5	2523	2562	57	10	6
lcf_mod	fmh_fmf	14011	1	3	4157	2954	0	1	0	594	398	0	0	2
lcf_pExp	fmh_fmf	0	4429	769	181818	35748	1682	154	782	14356	9982	1391	9917	3535
ems_lcf	iu_fmf													
hz_lcf	iu_fmf			209			303	1787	1156	52508	46291	1035	4	136
lcf_da	iu_fmf			2			7	34	7	105679	144586	21	43	31
lcf_du	iu_fmf			19			51	75	24	10858	2183	16	8	25
lcf_ems	iu_fmf			33119			199189	37540	23132	4069	5941	18490	59955	37810
lcf_hz	iu_fmf			12			55	14	33	4133	2894	39	55	31
lcf_ia	iu_fmf			21			10	8	26	50	43	33	26	23
lcf_iu	iu_fmf			14			29	34	18	17	139622	27	7	42
lcf_mod	iu_fmf			0			1	117	1	0	0	0	7	8
lcf_pExp	iu_fmf			41602			121550	44295	31619	33550	16749	31941	63195	36977
hz_lcf_iu	lcp_fhz		0	10			137	6	6	201030	2768	448	8311	22
hz_lcf	lcp_fhz_fmf			13			501	7	669	279	391	240	0	989
ems_lcf	lhz_fmf			11151			353218	92839	52736			107358	77374	21930
hz_lcf	lhz_fmf			304			353	316	11	115767	31189	364	54	228
lcf_da	lhz_fmf			11			31	35	6	28495	15366	29	41	21
lcf_du	lhz_fmf			13			27	116	22	85120	24947	29	153	47
lcf_ems	lhz_fmf			34304			217485	47024	27000	43557	0	33233	59533	36141
lcf_hz	lhz_fmf			100	0		32	36	43	59504	21321	51	85	141
lcf_ia	lhz_fmf			15			41	20	7	4204	4007	20	89	59
lcf_iu	lhz_fmf			25			8	212	6	3099	8164	9	192	10
lcf_mod	lhz_fmf			1			1	11	0		0	1	5	16
lcf_pExp	lhz_fmf			36610			251286	58135	15697	54057		33437	68155	36419
hz_lcf_iu	mcp_fhz		4719	327			13	1264	461	266360	4	202	7283	30
hz_lcf	mcp_fhz_fmf		787	1054			1769	1681	684	30841	19241	496	5	2302
ems_lcf	psaoca			6845			11506		4881			25883	29408	41689
hz_lcf	psaoca		36210	16		39840	36007	2678	1819	14	865	3271	334	9
lcf_da	psaoca	116596	33	9		7758	384	2681	10	1750	601	6	109	7
lcf_du	psaoca	64222	13	12533		4193	76409	62	9334	1975	12986	10503	52	29
lcf_ems	psaoca	155639	7635	946		251971	880	13725	1205	241581	8297	832	1525	538
lcf_hz	psaoca	23615	363	8823	0	8262	868	2272	9539	2752	6219	9194	64	9
lcf_ia	psaoca	20885	19	3	2	1508	489	3156	4	24	2	10	34	24
lcf_iu	psaoca		30	13113	0	1137	456	16	9345	17	6	12897	169	45
lcf_mod	psaoca	3065	3	3		2044	710	2323	6	2851	17176	3	209	17
lcf_pExp	psaoca	77894	11387	13229		166013	2305	9067	1119	227491	304582	591	395	448
shop	psaoca		2022	413		25853	7027	5879	863	103819	159249	6080	42945	6213
ems_lcf	sdr_fmf	1409	35763	343	3903	5	134	80205	1242	158	59	610	3485	53
hz_lcf	sdr_fmf	61648	4030	0		31396	1	592	34	18	2	1	7	11
lcf_da	sdr_fmf	8116	11	6285		7	1	5	15	324799	24	5	1	8269
lcf_du	sdr_fmf	11270	34	19		25	6	4	3	598424	691473	11	2	23
lcf_ems	sdr_fmf	40174	463	22407		1729	838	400	248	645506	1571533	792	2233	27331
lcf_hz	sdr_fmf	1751	16	2		133	3	7	5	209417	314583	11	5	17
lcf_ia	sdr_fmf	6207	9	11		0	10	9	1	216529	11912	4	3	13
lcf_iu	sdr_fmf	9935	11	0		8	1	6	4	312835	206457	7	2	11
lcf_mod	sdr_fmf	720	0	1		1	0	1	1	645213	667004	1	1	11
lcf_pExp	sdr_fmf	18149	91	9		0	358	809	494	991313	507068	599	2575	17
	Min-Var	0	0	0	0	0	0	0	0	0	0	0	0	0
	1. Quartile	1580	30	6	75	1327	13	14	6	616	499	10	8	11
	Median	20885	321	28	2275	11444	291	115	28	11573	12986	45	65	40
	3. Quartile	87575	2770	1027	5645	36771	2282	2412	963	59492	43697	1703	1455	1004
	Max-Var	585610	36210	162723	329672	2322013	1664186	1997758	203219	991313	2631218	801408	220872	42564

Table B.5: The **sample variance** of the explored search space sizes by the strategy combinations (rows) in the problem instances of the ***UM Translog*** domain (columns).

ModSel	PlanSel	Problem 2 - Regular Truck	Airplane	Armored Regular Truck	Auto Traincar	Auto Traincar - bis	Auto Truck	Flatbed Truck	Hopper Truck	Mail Traincar	Refrigerated Traincar	Regular Truck	Regular Truck - 3 Loc	Tanker Truck	Average
ems_lcf	cloca	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
hz_lcf	cloca	100%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	15%
lcf_da	cloca	100%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	15%
lcf_du	cloca	100%	20%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	17%
lcf_ems	cloca	100%	20%	0%	100%	20%	0%	0%	0%	0%	0%	0%	0%	0%	18%
lcf_hz	cloca	100%	20%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	17%
lcf_ia	cloca	100%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	15%
lcf_iu	cloca	100%	20%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	17%
lcf_mod	cloca	100%	0%	0%	100%	20%	0%	0%	0%	0%	20%	0%	0%	0%	18%
lcf_pExp	cloca	100%	80%	0%	100%	20%	0%	0%	0%	0%	0%	0%	0%	0%	23%
ems	df	60%	40%	0%	40%	60%	0%	0%	0%	80%	80%	0%	0%	0%	28%
shop	df	80%	60%	0%	60%	40%	0%	0%	0%	40%	40%	0%	0%	0%	25%
umcp	df	60%	40%	0%	60%	60%	20%	20%	0%	80%	60%	0%	0%	0%	31%
ems_lcf	du_fmf	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
hz_lcf	du_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_da	du_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_du	du_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_ems	du_fmf	100%	100%	0%	100%	100%	0%	0%	0%	80%	60%	0%	0%	0%	42%
lcf_hz	du_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_ia	du_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_iu	du_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_mod	du_fmf	100%	100%	0%	100%	100%	0%	0%	0%	80%	80%	0%	0%	0%	43%
lcf_pExp	du_fmf	100%	100%	0%	100%	100%	0%	0%	0%	20%	20%	0%	0%	0%	34%
umcp	ff(ct)	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
ems_lcf	fhz_fmf	100%	100%	0%	60%	0%	0%	0%	0%	0%	0%	0%	0%	0%	20%
hz_lcf	fhz_fmf	80%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	14%
lcf_da	fhz_fmf	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_du	fhz_fmf	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_ems	fhz_fmf	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_hz	fhz_fmf	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_ia	fhz_fmf	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_iu	fhz_fmf	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_mod	fhz_fmf	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_pExp	fhz_fmf	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
hz_lcf_iu	fhz_lcp	20%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	9%
hz_lcf	fhz_lcp_fmf	80%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	14%
hz_lcf_iu	fhz_mcp	100%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	15%
hz_lcf	fhz_mcp_fmf	100%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	15%
ems_lcf	fmh_fmf	100%	100%	0%	100%	100%	20%	100%	0%	100%	80%	20%	0%	0%	55%
hz_lcf	fmh_fmf	80%	20%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	15%
lcf_da	fmh_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
lcf_du	fmh_fmf	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_ems	fmh_fmf	80%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	6%
lcf_hz	fmh_fmf	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_ia	fmh_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
lcf_iu	fmh_fmf	40%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%
lcf_mod	fmh_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
lcf_pExp	fmh_fmf	80%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	6%
ems_lcf	iu_fmf	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
hz_lcf	iu_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_da	iu_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_du	iu_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_ems	iu_fmf	100%	100%	0%	100%	100%	0%	0%	0%	40%	60%	0%	0%	0%	38%
lcf_hz	iu_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_ia	iu_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_iu	iu_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_mod	iu_fmf	100%	100%	0%	100%	100%	0%	0%	0%	80%	80%	0%	0%	0%	43%
lcf_pExp	iu_fmf	100%	100%	0%	100%	100%	0%	0%	0%	20%	40%	0%	0%	0%	35%
hz_lcf_iu	lcp_fhz	100%	80%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	29%
hz_lcf	lcp_fhz_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
ems_lcf	lhz_fmf	100%	100%	0%	100%	100%	0%	0%	0%	100%	100%	0%	40%	0%	49%
hz_lcf	lhz_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_da	lhz_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_du	lhz_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_ems	lhz_fmf	100%	100%	0%	100%	100%	0%	0%	0%	40%	80%	0%	0%	0%	40%
lcf_hz	lhz_fmf	100%	100%	0%	80%	100%	0%	0%	0%	0%	0%	0%	0%	0%	29%
lcf_ia	lhz_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_iu	lhz_fmf	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	31%
lcf_mod	lhz_fmf	100%	100%	0%	100%	100%	0%	0%	0%	100%	80%	0%	0%	0%	45%
lcf_pExp	lhz_fmf	100%	100%	0%	100%	100%	0%	0%	0%	40%	100%	0%	0%	0%	42%
hz_lcf_iu	mcp_fhz	100%	0%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	23%
hz_lcf	mcp_fhz_fmf	100%	20%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	25%
ems_lcf	psaoca	100%	100%	0%	100%	100%	0%	100%	0%	100%	100%	0%	40%	0%	57%
hz_lcf	psaoca	100%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	15%
lcf_da	psaoca	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_du	psaoca	20%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	9%
lcf_ems	psaoca	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_hz	psaoca	20%	0%	0%	80%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_ia	psaoca	0%	0%	0%	60%	0%	0%	0%	0%	0%	0%	0%	0%	0%	5%
lcf_iu	psaoca	100%	0%	0%	80%	0%	0%	0%	0%	0%	0%	0%	0%	0%	14%
lcf_mod	psaoca	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_pExp	psaoca	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
shop	psaoca	100%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	15%
ems_lcf	sdr_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
hz_lcf	sdr_fmf	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_da	sdr_fmf	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	8%
lcf_du	sdr_fmf	0%	0%	0%	100%	20%	0%	0%	0%	0%	0%	0%	0%	0%	9%
lcf_ems	sdr_fmf	0%	0%	0%	100%	40%	0%	0%	0%	0%	0%	0%	0%	0%	11%
lcf_hz	sdr_fmf	0%	0%	0%	100%	40%	0%	0%	0%	0%	0%	0%	0%	0%	11%
lcf_ia	sdr_fmf	0%	0%	0%	100%	40%	0%	0%	0%	0%	0%	0%	0%	0%	11%
lcf_iu	sdr_fmf	0%	0%	0%	100%	60%	0%	0%	0%	0%	0%	0%	0%	0%	12%
lcf_mod	sdr_fmf	0%	0%	0%	100%	40%	0%	0%	0%	0%	20%	0%	0%	0%	12%
lcf_pExp	sdr_fmf	0%	0%	0%	100%	80%	0%	0%	0%	0%	0%	0%	0%	0%	14%
	Min-F-Ratio	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	**0%**
	1. Quartile	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	**8%**
	Median	100%	0%	0%	100%	20%	0%	0%	0%	0%	0%	0%	0%	0%	**17%**
	3. Quartile	100%	100%	0%	100%	100%	0%	0%	0%	0%	0%	0%	0%	0%	**31%**
	Max-F-Ratio	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	**100%**
	Average	66%	42%	3%	86%	46%	4%	6%	3%	14%	15%	3%	4%	3%	**23%**

Table B.6: The **failure ratio** of the strategy combinations (rows) in the problem instances of the ***UM Translog*** domain (columns).

		Problem												
ModSel	**PlanSel**	**P0**	**P1**	**P2**	**P0-2**	**P0-P1**	**P1-2**	**P0-3**	**P0-P2**	**P1-3**	**P1-P2**	**P0-P1-P2**	**P2-2**	**P2-3**
ems_lcf	cloca	17	24	92	684	1714	4251		2437		6480			
hz_lcf	cloca	17	23	209	3111	7171								
lcf_da	cloca	17	23	112	403	2601	4812	6348	2069		7988			
lcf_du	cloca	17	23	174	2514									
lcf_ems	cloca	17	23	96	802	2534	6287		3545					
lcf_hz	cloca	17	23	125	3683	5639								
lcf_ia	cloca	17	23	181	608	1291	6151		5852					
lcf_iu	cloca	17	23	161	493	1516	4813		5468					
lcf_mod	cloca	16	24	142	4176									
lcf_pExp	cloca	15	23	177	3967				8233					
ems	df	17	24	49	95	112	149	248	233	310	250	424	370	1036
shop	df	15	23	52	120	250	253	488	195	815	272	638	371	985
umcp	df	16	22	57	150	138	855	523	274	917	1716	874	634	
ems_lcf	du_fmf	17	24	46	88	129	143		213		301		558	
hz_lcf	du_fmf	17	24	1028										
lcf_da	du_fmf	17	24	43	144	472			437		1073		2396	
lcf_du	du_fmf	17	24	58	244	608	531		2151				2546	
lcf_ems	du_fmf	17	24	45	155	137	187		199		262	1126	863	
lcf_hz	du_fmf	17	24	1142	1096				1316				1153	
lcf_ia	du_fmf	17	24	846	339	222	654		204				2334	
lcf_iu	du_fmf	17	24	982	347	613	891		200				2565	
lcf_mod	du_fmf	17	24	58	140	276			261		503	1183	1065	
lcf_pExp	du_fmf	17	24	63	166	233	598		295		330	1402	1223	
umcp	ff(ct)	16	23	120	5602									
ems_lcf	fhz_fmf	17	24	64	78	105	129	243	121	394	144		587	
hz_lcf	fhz_fmf	17	24	64	49	63	70	167	860	201		372	234	705
lcf_da	fhz_fmf	17	24	63	74	725	164	192	181		329	893	503	948
lcf_du	fhz_fmf	17	24	68	74	90	108	187	795	261			425	
lcf_ems	fhz_fmf	17	24	64	88	116	117	253	160	393	207	551	459	1334
lcf_hz	fhz_fmf	17	24	65	51	63	72	141	883	194		421	1169	
lcf_ia	fhz_fmf	17	24	62	42	52	70	105	235	190	765	327	559	654
lcf_iu	fhz_fmf	17	24	61	46	53	187	112	233	165	781		501	
lcf_mod	fhz_fmf	17	24	68	103	125	798	241	982	412		485	562	1030
lcf_pExp	fhz_fmf	17	24	69	89	112	1075	253	183	393	356	498	1068	1190
hz_lcf_iu	fhz_lcp	17	24	64	58	96	82	114	388	166	756	190	316	642
hz_lcf	fhz_lcp_fmf	17	24	60	48	60	108	146	797	253		356	218	712
hz_lcf_iu	fhz_mcp	17	24	65	68	70	181	121	239	539	595	172	740	591
hz_lcf	fhz_mcp_fmf	17	24	67	47	66	125	133	571	542		319	254	670
ems_lcf	fmh_fmf	17	24	98	254	528	706	2465						
hz_lcf	fmh_fmf	17	24	87	121	187	335	381	661	1078	665	1002	1203	
lcf_da	fmh_fmf	17	24	99	386	728	517							
lcf_du	fmh_fmf	17	24	91	107	171	285	578	6111	1472	296			
lcf_ems	fmh_fmf	17	24	100	552	1076	1454	3427	1102					
lcf_hz	fmh_fmf	17	24	90	92	166	197	423		1449				
lcf_ia	fmh_fmf	17	24	91	78	192	554	379	2642	2587	855			
lcf_iu	fmh_fmf	17	24	96	117	155	233	518	3211	2205	758			
lcf_mod	fmh_fmf	17	24	93	1307	169	511	619	676	4603	276			
lcf_pExp	fmh_fmf	17	24	93	223	184	240	609	5444	2886				
ems_lcf	iu_fmf	17	24	46	86	114	149	853	219	1284	290		513	
hz_lcf	iu_fmf	17	24	357	372									
lcf_da	iu_fmf	17	24	45	92			2181	241		1092		2107	
lcf_du	iu_fmf	17	24	66	199	335	732		1816				3306	
lcf_ems	iu_fmf	17	24	44	94	133	211	869	197	940	296	880	611	1740
lcf_hz	iu_fmf	17	24	406	1282				522				1696	
lcf_ia	iu_fmf	17	24	177	204	275	404		1625				1912	1302
lcf_iu	iu_fmf	17	24	163	208	502	450		1656				2874	
lcf_mod	iu_fmf	17	24	70	182	150	468		269		400	1211	735	
lcf_pExp	iu_fmf	17	24	64	107	179	496		268		262	1022	1541	
hz_lcf_iu	lcp_fhz	17	24	52	1575	65	77	148	110	233	111	216	157	422
hz_lcf	lcp_fhz_fmf	17	24	49		66	89	134	113	203	151	295	223	
ems_lcf	lhz_fmf	17	24	55	107	117	155		224		324	1298	501	
hz_lcf	lhz_fmf	17	24	54	68	384	471	182	1976		471	605	1719	
lcf_da	lhz_fmf	17	24	57	160	2279	368		463		805		3002	
lcf_du	lhz_fmf	17	24	56	106	420	1121	449					2781	
lcf_ems	lhz_fmf	17	24	57	97	139	159		215		286			
lcf_hz	lhz_fmf	17	24	57	57	481	1651						1531	
lcf_ia	lhz_fmf	17	24	54	75	115	1128	437					2385	
lcf_iu	lhz_fmf	17	24	51	59	201	1527	501					2771	
lcf_mod	lhz_fmf	17	24	60	112	203	346	445	284		516	1728		
lcf_pExp	lhz_fmf	17	24	62	100	144	453	566	296	514	358			
hz_lcf_iu	mcp_fhz	17	24	311	65	58	93		103				175	
hz_lcf	mcp_fhz_fmf	17	24	861	50			132			153			
ems_lcf	psaoca	17	24	75	119	198	210	599	349	1270	296		999	
hz_lcf	psaoca	17	24	140	82	101	126	514	526	526	2276	1082	365	
lcf_da	psaoca	17	23	74	100	653	212	261	226	607	423	621	1013	1433
lcf_du	psaoca	17	24	131	77	115	174	262	814	673		1384	559	1873
lcf_ems	psaoca	17	24	65	106	177	212	354	289	1337	493	607	567	1549
lcf_hz	psaoca	17	23	111	96	98	125	274	792	382		701	518	2549
lcf_ia	psaoca	17	24	87	59	70	92	146	221	241	530	963	796	1666
lcf_iu	psaoca	16	24	132	60	73	94	151	227	236	542	1006	425	1309
lcf_mod	psaoca	16	23	155	1238	1849	591	474	923	1049		954	4415	2116
lcf_pExp	psaoca	15	23	113	262	781	565	539	499	1152	556	1135	541	1947
shop	psaoca	16	22	54	112	1509	325	1868						
ems_lcf	sdr_fmf	17	24	45	73	98	125	250	108	469	134		310	
hz_lcf	sdr_fmf	17	24	43	51	71	135	138		294		357	203	678
lcf_da	sdr_fmf	17	24	43	79	1671	161	171	147		320		624	
lcf_du	sdr_fmf	17	24	49	87	86	192	202	692	342			559	
lcf_ems	sdr_fmf	17	24	44	82	118	129	259	157	429	207	456	430	1362
lcf_hz	sdr_fmf	17	24	47	50	61	139	127	803	192		577	380	
lcf_ia	sdr_fmf	17	24	42	41	50	64	105	237	160	606	288	507	564
lcf_iu	sdr_fmf	17	24	43	42	59	94	117	236	200	604		1055	
lcf_mod	sdr_fmf	17	24	46	91	109	540	282	604	397		466	491	1020
lcf_pExp	sdr_fmf	17	24	47	84	115	175	272	381	372	260	572	804	1134
	Min-Avg	15	22	42	41	50	64	105	103	160	111	172	157	422
	1. Quartile	17	24	54	76	100	132	150	223	247	279	421	467	698
	Median	17	24	65	106	155	212	262	365	412	379	607	617	1085
	3. Quartile	17	24	112	249	477	560	515	893	994	650	1006	1454	1462
	Max-Avg	17	24	1142	5602	7171	6287	6348	8233	4603	7988	1728	4415	2549

Table B.7: The **arithmetic means** of the explored search space sizes by the strategy combinations (rows) in the problem instances of the ***CrissCross*** domain (columns).

		Problem												
ModSel	**PlanSel**	**P0**	**P1**	**P2**	**P0-3**	**P0-2**	**P0-P2**	**P0-P1**	**P1-P2**	**P1-2**	**P0-P1-P2**	**P1-3**	**P2-2**	**P2-3**
ems_lcf	cloca	0	0	356		31078	3225597	250435	9508896	3569364				
hz_lcf	cloca	0	1	5035		2034191		1038364						
lcf_da	cloca	0	1	475	3634485	16034	861394	3108195	2544932	0				
lcf_du	cloca	0	1	4798		2195450								
lcf_ems	cloca	0	1	126		121914	789830	568258		5033523				
lcf_hz	cloca	1	1	1696		5100236		223781						
lcf_ia	cloca	1	1	6474		152787	3067687	345091		6944206				
lcf_iu	cloca	1	1	9228		57433	3527354	308714		2770790				
lcf_mod	cloca	1	1	1743		6480971								
lcf_pExp	cloca	0	1	553		5731070	10825205							
ems	df	0	0	23	1835	155	21590	149	3909	1060	3610	1867	14301	32426
shop	df	1	1	27	22980	319	2324	21082	5949	23275	5263	28879	248	19773
umcp	df	1	0	47	17387	3565	4650	1255	4321420	510668	0	3961	18984	
ems_lcf	du_fmf	0	0	5		493	586	972	5189	1138			15396	
hz_lcf	du_fmf	0	0	50881										
lcf_da	du_fmf	0	0	1		664	10009	0	70706				13559	
lcf_du	du_fmf	0	0	42		14056	0	28537		0			39760	
lcf_ems	du_fmf	0	0	3		6495	1937	902	1109	6544	56716		123288	
lcf_hz	du_fmf	0	0	0		0	146851						965	
lcf_ia	du_fmf	0	0	65784		4596	113	8483		13198			3188	
lcf_iu	du_fmf	0	0	122566		1769	113	6696		15722			339728	
lcf_mod	du_fmf	0	0	76		2137	1454	21985	53309		0		157675	
lcf_pExp	du_fmf	0	0	131		9637	10899	20164	5513	139093	10952		524797	
umcp	ff(ct)	1	1	2354		4389221								
ems_lcf	fhz_fmf	0	0	9	390	218	121	125	127	227		3621	15165	
hz_lcf	fhz_fmf	0	0	6	502	4	26526	69		25	8023	63	143	52
lcf_da	fhz_fmf	0	0	4	762	40	1036	1556832	377	2927	0		19423	108511
lcf_du	fhz_fmf	0	0	4	369	121	12497	447		138		337	5696	
lcf_ems	fhz_fmf	0	0	15	198	114	209	94	689	68	79599	4817	25359	962830
lcf_hz	fhz_fmf	0	0	3	654	94	7905	49		42	4321	395	1189438	
lcf_ia	fhz_fmf	0	0	2	56	2	39	34	120	525	208	7139	2331	153
lcf_iu	fhz_fmf	0	0	1	20	104	29	6	89	29169		174	2975	
lcf_mod	fhz_fmf	0	0	5	1769	671	645	397		1968611	1766	8294	32423	22512
lcf_pExp	fhz_fmf	0	0	8	2925	635	453	41	71948	4150423	2257	2721	607915	15326
hz_lcf_iu	fhz_lcp	0	0	1	120	86	84	1040	724	1269	1169	603	47163	66352
hz_lcf	fhz_lcp_fmf	0	0	79	103	5	0	9		2249	8752	2203	1187	0
hz_lcf_iu	fhz_mcp	0	0	22	301	140	832	562	936	22919	120	28820	261233	60147
hz_lcf	fhz_mcp_fmf	0	0	3	457	16	0	15		14507	18202	63349	2679	564
ems_lcf	fmh_fmf	0	0	5	0	32417		56527		123413				
hz_lcf	fmh_fmf	0	0	4	46904	3270	20563	17148	45562	74012	504098	627947	1470048	
lcf_da	fmh_fmf	0	0	11		25056		107859		27371				
lcf_du	fmh_fmf	0	0	22	50116	458	385442	2893	0	6110		173360		
lcf_ems	fmh_fmf	0	0	5	0	131576	0	405055		1007222				
lcf_hz	fmh_fmf	0	0	2	4261	2239		9933		6667		181409		
lcf_ia	fmh_fmf	0	0	13	27635	500	2599200	4997	261590	112220		4494990		
lcf_iu	fmh_fmf	0	0	28	50918	2393	1051525	3330	353631	12604		2463667		
lcf_mod	fmh_fmf	0	0	19	36949	6424142	70940	2863	0	131885		1883936		
lcf_pExp	fmh_fmf	0	0	6	68627	41926	0	3802		8118		929943		
ems_lcf	iu_fmf	0	0	7	105451	454	1174	294	3200	178		11517	4812	
hz_lcf	iu_fmf	0	0	24134		24638								
lcf_da	iu_fmf	0	0	19	0	66	1835		77618				35442	
lcf_du	iu_fmf	0	0	43		8275	66358	7952		0			7439147	
lcf_ems	iu_fmf	0	0	3	115587	385	427	1059	13004	12926	116340	125052	77269	0
lcf_hz	iu_fmf	0	0	16327		0	42863						3008383	
lcf_ia	iu_fmf	0	0	1734		1855	718	21538		17373			101219	44402
lcf_iu	iu_fmf	0	0	2000		2053	135	18799		18814			4781617	
lcf_mod	iu_fmf	0	0	402		6084	2993	1770	6503	37457	124266		160176	
lcf_pExp	iu_fmf	0	0	27		83	1640	3650	2499	51441	63725		260898	
hz_lcf_iu	lcp_fhz	0	0	18	520	10235	187	194	43	524	215	10085	79	6074
hz_lcf	lcp_fhz_fmf	0	0	19	640		39	82	745	197	485	1316	111	
ems_lcf	lhz_fmf	0	0	4		1406	588	525	4162	398	0		9185	
hz_lcf	lhz_fmf	0	0	2	183	176	40045	396475	2377	103740	0		123505	
lcf_da	lhz_fmf	0	0	18		432	4822	8770013	2855	5454			1715653	
lcf_du	lhz_fmf	0	0	6	0	1465		196797		0			26968	
lcf_ems	lhz_fmf	0	0	21		447	2882	431	8696	608				
lcf_hz	lhz_fmf	0	0	24		143		398285		810046			161221	
lcf_ia	lhz_fmf	0	0	6	23950	398		4165		627406			44610	
lcf_iu	lhz_fmf	0	0	17	0	108		35439		873135			216429	
lcf_mod	lhz_fmf	0	0	65	0	1061	8231	7941	53990	59590	0			
lcf_pExp	lhz_fmf	0	0	12	0	2149	3004	982	4130	0		0		
hz_lcf_iu	mcp_fhz	0	0	6596		227	0	0		0			0	
hz_lcf	mcp_fhz_fmf	0	0	126678	0	18			0					
ems_lcf	psaoca	0	1	98	76977	515	29906	1509	65929	5255		140758	165212	
hz_lcf	psaoca	0	1	1642	40240	598	88937	76	0	957	143737	16913	4194	
lcf_da	psaoca	0	1	385	3814	349	2633	527703	58872	2306	18690	20907	321143	262088
lcf_du	psaoca	0	1	2531	2299	71	24186	1489		234	860588	25832	47831	306647
lcf_ems	psaoca	0	1	66	9759	295	11298	995	24167	545	26399	555723	32409	77370
lcf_hz	psaoca	1	1	1559	3254	597	31522	264		380	99006	9877	26080	4847296
lcf_ia	psaoca	1	0	415	16	39	129	187	345	236	153199	1397	49385	207889
lcf_iu	psaoca	1	1	5686	656	277	184	601	352	121	161384	2147	9935	301155
lcf_mod	psaoca	1	1	7818	1910	5151414	76204	6005773		308127	69064	0	6417955	31501
lcf_pExp	psaoca	0	1	980	23242	111371	54170	1207763	314836	126437	47911	0	0	0
shop	psaoca	1	0	44	3928405	237		7684598		55423				
ems_lcf	sdr_fmf	0	0	3	599	69	25	106	36	352		23322	2663	
hz_lcf	sdr_fmf	0	0	12	218	10		767		7130	16589	3425	249	14836
lcf_da	sdr_fmf	0	0	4	99	148	33	5036967	3300	2953			7417	
lcf_du	sdr_fmf	0	0	5	111	934	13554	286		17730		38501	20367	
lcf_ems	sdr_fmf	0	0	7	320	79	332	452	586	1055	1832	5245	46178	1026020
lcf_hz	sdr_fmf	0	0	1	12	54	0	15		3416	40373	191	2811	
lcf_ia	sdr_fmf	0	0	2	17	1	108	8	150	11	3269	81	2725	2054
lcf_iu	sdr_fmf	0	0	2	171	13	22	228	45	5017		12604	1300783	
lcf_mod	sdr_fmf	0	0	1	2570	295	0	740		515978	6934	3663	22978	16263
lcf_pExp	sdr_fmf	0	0	4	1238	245	235001	250	1938	5925	10977	1996	246967	90862
	Min-Var	0	0	0	0	0	0	0	0	0	0	0	0	0
	1. Quartile	0	0	5	109	117	127	257	358	461	1169	1632	4349	5069
	Median	0	0	21	647	500	2130	1509	3250	6544	8752	7139	29688	31963
	3. Quartile	0	0	475	18785	7385	30310	31988	51372	66801	63725	33690	164214	133356
	Max-Var	1	1	126678	3928405	6480971	10825205	8770013	9508896	6944206	860588	4494990	7439147	4847296

Table B.8: The **sample variance** of the explored search space sizes by the strategy combinations (rows) in the problem instances of the ***CrissCross*** domain (columns).

		Problem													
ModSel	**PlanSel**	**P0**	**P1**	**P2**	**P0-2**	**P0-P1**	**P1-2**	**P0-P2**	**P2-2**	**P0-3**	**P1-P2**	**P1-3**	**P0-P1-P2**	**P2-3**	**Avg**
ems_lcf	cloca	0%	0%	0%	0%	0%	0%	0%	100%	100%	0%	100%	100%	100%	38%
hz_lcf	cloca	0%	0%	0%	0%	40%	100%	100%	100%	100%	100%	100%	100%	100%	65%
lcf_da	cloca	0%	0%	0%	0%	20%	80%	0%	100%	0%	40%	100%	100%	100%	42%
lcf_du	cloca	0%	0%	0%	0%	100%	100%	100%	100%	100%	100%	100%	100%	100%	69%
lcf_ems	cloca	0%	0%	0%	0%	0%	0%	20%	100%	100%	100%	100%	100%	100%	48%
lcf_hz	cloca	0%	0%	0%	0%	60%	100%	100%	100%	100%	100%	100%	100%	100%	66%
lcf_ia	cloca	0%	0%	0%	0%	0%	0%	0%	100%	100%	100%	100%	100%	100%	46%
lcf_iu	cloca	0%	0%	0%	0%	0%	0%	0%	100%	100%	100%	100%	100%	100%	46%
lcf_mod	cloca	0%	0%	0%	20%	100%	100%	100%	100%	100%	100%	100%	100%	100%	71%
lcf_pExp	cloca	0%	0%	0%	20%	100%	100%	60%	100%	100%	100%	100%	100%	100%	68%
ems	df	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
shop	df	0%	0%	0%	0%	20%	20%	0%	0%	20%	20%	0%	40%	20%	11%
umcp	df	0%	0%	0%	0%	0%	0%	0%	40%	20%	20%	60%	80%	100%	25%
ems_lcf	du_fmf	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	100%	100%	100%	31%
hz_lcf	du_fmf	0%	0%	60%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	82%
lcf_da	du_fmf	0%	0%	0%	0%	80%	100%	0%	0%	100%	20%	100%	100%	100%	46%
lcf_du	du_fmf	0%	0%	0%	0%	20%	80%	80%	0%	100%	100%	100%	100%	100%	52%
lcf_ems	du_fmf	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	100%	40%	100%	26%
lcf_hz	du_fmf	0%	0%	80%	80%	100%	100%	20%	0%	100%	100%	100%	100%	100%	68%
lcf_ia	du_fmf	0%	0%	20%	0%	0%	0%	0%	20%	100%	100%	100%	100%	100%	42%
lcf_iu	du_fmf	0%	0%	0%	0%	0%	0%	0%	20%	100%	100%	100%	100%	100%	40%
lcf_mod	du_fmf	0%	0%	0%	0%	20%	100%	0%	20%	100%	0%	100%	80%	100%	40%
lcf_pExp	du_fmf	0%	0%	0%	0%	20%	20%	0%	40%	100%	0%	100%	60%	100%	34%
umcp	ff(ct)	0%	0%	0%	0%	100%	100%	100%	100%	100%	100%	100%	100%	100%	69%
ems_lcf	fhz_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	100%	15%
hz_lcf	fhz_fmf	0%	0%	0%	0%	0%	0%	40%	0%	0%	100%	20%	0%	40%	15%
lcf_da	fhz_fmf	0%	0%	0%	0%	20%	0%	0%	0%	0%	0%	100%	80%	20%	17%
lcf_du	fhz_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	100%	100%	23%
lcf_ems	fhz_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	20%	2%
lcf_hz	fhz_fmf	0%	0%	0%	0%	0%	0%	20%	0%	0%	100%	0%	0%	100%	17%
lcf_ia	fhz_fmf	0%	0%	0%	0%	0%	0%	0%	20%	0%	0%	0%	0%	0%	2%
lcf_iu	fhz_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	100%	15%
lcf_mod	fhz_fmf	0%	0%	0%	0%	0%	0%	40%	0%	0%	100%	0%	0%	0%	11%
lcf_pExp	fhz_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	40%	3%
hz_lcf_iu	fhz_lcp	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	40%	0%	40%	6%
hz_lcf	fhz_lcp_fmf	0%	0%	0%	0%	0%	0%	80%	0%	0%	100%	0%	0%	80%	20%
hz_lcf_iu	fhz_mcp	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	20%	0%	20%	3%
hz_lcf	fhz_mcp_fmf	0%	0%	0%	0%	0%	0%	80%	0%	0%	100%	0%	0%	40%	17%
ems_lcf	fmh_fmf	0%	0%	0%	0%	0%	0%	100%	100%	80%	100%	100%	100%	100%	52%
hz_lcf	fmh_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	20%	100%	9%
lcf_da	fmh_fmf	0%	0%	0%	0%	0%	0%	100%	100%	100%	100%	100%	100%	100%	54%
lcf_du	fmh_fmf	0%	0%	0%	0%	0%	0%	60%	100%	0%	80%	0%	100%	100%	34%
lcf_ems	fmh_fmf	0%	0%	0%	0%	0%	0%	80%	100%	80%	100%	100%	100%	100%	51%
lcf_hz	fmh_fmf	0%	0%	0%	0%	0%	0%	100%	100%	0%	100%	0%	100%	100%	38%
lcf_ia	fmh_fmf	0%	0%	0%	0%	0%	0%	60%	100%	0%	0%	20%	100%	100%	29%
lcf_iu	fmh_fmf	0%	0%	0%	0%	0%	0%	40%	100%	0%	0%	40%	100%	100%	29%
lcf_mod	fmh_fmf	0%	0%	0%	0%	0%	20%	20%	100%	0%	80%	0%	100%	100%	32%
lcf_pExp	fmh_fmf	0%	0%	0%	0%	0%	0%	80%	100%	0%	100%	0%	100%	100%	37%
ems_lcf	iu_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	20%	100%	100%	17%
hz_lcf	iu_fmf	0%	0%	0%	20%	100%	100%	100%	100%	100%	100%	100%	100%	100%	71%
lcf_da	iu_fmf	0%	0%	0%	0%	100%	100%	20%	40%	80%	60%	100%	100%	100%	54%
lcf_du	iu_fmf	0%	0%	0%	0%	0%	80%	40%	20%	100%	100%	100%	100%	100%	49%
lcf_ems	iu_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	40%	80%	9%
lcf_hz	iu_fmf	0%	0%	0%	80%	100%	100%	20%	0%	100%	100%	100%	100%	100%	62%
lcf_ia	iu_fmf	0%	0%	0%	0%	0%	0%	0%	20%	100%	100%	100%	100%	60%	37%
lcf_iu	iu_fmf	0%	0%	0%	0%	0%	0%	0%	20%	100%	100%	100%	100%	100%	40%
lcf_mod	iu_fmf	0%	0%	0%	0%	0%	40%	0%	40%	100%	0%	100%	40%	100%	32%
lcf_pExp	iu_fmf	0%	0%	0%	0%	0%	20%	0%	20%	100%	0%	100%	60%	100%	31%
hz_lcf_iu	lcp_fhz	0%	0%	0%	0%	20%	0%	20%	20%	20%	0%	0%	20%	0%	8%
hz_lcf	lcp_fhz_fmf	0%	0%	0%	100%	0%	20%	0%	0%	0%	0%	20%	0%	100%	18%
ems_lcf	lhz_fmf	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	100%	80%	100%	29%
hz_lcf	lhz_fmf	0%	0%	0%	0%	0%	40%	60%	60%	40%	40%	100%	80%	100%	40%
lcf_da	lhz_fmf	0%	0%	0%	0%	0%	0%	0%	20%	100%	0%	100%	100%	100%	32%
lcf_du	lhz_fmf	0%	0%	0%	0%	0%	80%	100%	0%	80%	100%	100%	100%	100%	51%
lcf_ems	lhz_fmf	0%	0%	0%	0%	0%	0%	0%	100%	100%	0%	100%	100%	100%	38%
lcf_hz	lhz_fmf	0%	0%	0%	0%	0%	0%	100%	0%	100%	100%	100%	100%	100%	46%
lcf_ia	lhz_fmf	0%	0%	0%	0%	0%	20%	100%	0%	40%	100%	100%	100%	100%	43%
lcf_iu	lhz_fmf	0%	0%	0%	0%	0%	20%	100%	0%	80%	100%	100%	100%	100%	46%
lcf_mod	lhz_fmf	0%	0%	0%	0%	0%	20%	0%	100%	80%	0%	100%	80%	100%	37%
lcf_pExp	lhz_fmf	0%	0%	0%	0%	0%	80%	0%	100%	80%	0%	80%	100%	100%	42%
hz_lcf_iu	mcp_fhz	0%	0%	0%	0%	80%	80%	80%	80%	100%	100%	100%	100%	100%	63%
hz_lcf	mcp_fhz_fmf	0%	0%	0%	0%	100%	100%	100%	100%	80%	80%	100%	100%	100%	66%
ems_lcf	psaoca	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	100%	15%
hz_lcf	psaoca	0%	0%	0%	0%	0%	0%	0%	0%	0%	80%	0%	40%	100%	17%
lcf_da	psaoca	0%	0%	0%	0%	20%	0%	0%	0%	0%	0%	0%	0%	60%	6%
lcf_du	psaoca	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	20%	9%
lcf_ems	psaoca	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	20%	0%	2%
lcf_hz	psaoca	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	8%
lcf_ia	psaoca	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
lcf_iu	psaoca	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
lcf_mod	psaoca	0%	0%	0%	20%	20%	40%	0%	20%	20%	100%	80%	0%	60%	28%
lcf_pExp	psaoca	0%	0%	0%	0%	20%	40%	0%	80%	20%	0%	80%	0%	80%	25%
shop	psaoca	0%	0%	0%	0%	20%	40%	100%	100%	60%	100%	100%	100%	100%	55%
ems_lcf	sdr_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	100%	15%
hz_lcf	sdr_fmf	0%	0%	0%	0%	0%	0%	100%	0%	0%	100%	20%	0%	40%	20%
lcf_da	sdr_fmf	0%	0%	0%	0%	20%	0%	0%	0%	0%	0%	100%	100%	100%	25%
lcf_du	sdr_fmf	0%	0%	0%	0%	0%	0%	40%	0%	0%	100%	0%	100%	100%	26%
lcf_ems	sdr_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
lcf_hz	sdr_fmf	0%	0%	0%	0%	0%	0%	80%	0%	0%	100%	0%	0%	100%	22%
lcf_ia	sdr_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
lcf_iu	sdr_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	100%	15%
lcf_mod	sdr_fmf	0%	0%	0%	0%	0%	0%	80%	0%	0%	100%	0%	0%	40%	17%
lcf_pExp	sdr_fmf	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	Min-F-Ratio	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	**0%**
	1. Quartile	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	60%	**15%**
	Median	0%	0%	0%	0%	0%	0%	0%	0%	20%	80%	80%	100%	100%	**31%**
	3. Quartile	0%	0%	0%	0%	20%	40%	80%	100%	100%	100%	100%	100%	100%	**46%**
	Max-F-Ratio	0%	0%	80%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	**82%**
	Average	0%	0%	2%	5%	16%	24%	31%	35%	45%	52%	55%	65%	78%	**31%**

Table B.9: The **failure ratio** of the strategy combinations (rows) in the problem instances of the ***CrissCross*** domain (columns).

Bibliography

[1] M. Aarup, M. M. Arentoft, Y. Parrod, J. Stader, and I. Stokes. OPTIMUM-AIV: A knowledge-based planning and scheduling system for spacecraft AIV. In Zweben and Fox [299], chapter 16, pages 451–469. ISBN 1-55860-260-7.

[2] John M. Agosta and David E. Wilkins. Using SIPE-2 to plan emergency response to marine oil spills. *IEEE Expert*, 11(6):6–8, 1996.

[3] Rachid Alami, Sara Fleury, Matthieu Herrb, Felix Ingrand, and Frédéric Robert. Multi robot cooperation in the Martha project. *IEEE Robotics and Automation Magazine*, 5(1):36–47, March 1998. Special Issue on "Robotics & Automation in the European Union".

[4] James Allen, James Hendler, and Austin Tate, editors. *Readings in Planning*. Representation and Reasoning. Morgan Kaufmann, San Francisco, CA, USA, 1990. ISBN 1-55860-130-9.

[5] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[6] James F. Allen, Henry A. Kautz, Richard N. Pelavin, and Josh D. Tenenberg. *Reasoning about plans*. Representation And Reasoning. Morgan Kaufmann, San Francisco, CA, USA, 1991. ISBN 1-55860-137-6.

[7] José Luis Ambite, Vinay K. Chaudhri, Richard Fikes, Jessica Jenkins, Sunil Mishra, Maria Muslea, Tomas Uribe, and Guizhen Yang. Design and implementation of the CALO query manager. In Bruce Porter and William Cheetham, editors, *Proceedings of the 18th Conference on Innovative Applications of Artificial Intelligence (IAAI-2006)*, Boston, Massachusetts, July 2006. AAAI Press, Menlo Park, CA, USA.

[8] Scott Andrews, Brian Kettler, Kutluhan Erol, and James A. Hendler. UM Translog: A planning domain for the development and benchmarking of planning systems. Technical Report CS-TR-3487, University of Maryland, 1995.

[9] Paolo Avesani, Anna Perini, and Francesco Ricci. Interactive case-based planning for forest fire management. *Applied Intelligence*, 13(1):41–57, July 2000.

[10] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003. ISBN 0-521-78176-0.

[11] Fahiem Bacchus and Michael Ady. Planning with resources and concurrency: A forward chaining approach. In Nebel [204], pages 417–424.

[12] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.

[13] Fahiem Bacchus and Qiang Yang. The downward refinement property. In Mylopoulos and Reiter [196], pages 286–292.

[14] Ruzena Bajcsy, editor. *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, Chambery, France, August 1993. Morgan Kaufmann, San Francisco, CA, USA.

[15] Chitta Baral and Michael Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming*, 31(1-3):85–117, 1997.

[16] Anthony Barrett and Daniel S. Weld. Partial-order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, 67(1):71–112, 1994.

[17] Howard Beck. The management of job-shop scheduling constraints in TOSCA. In *NSF Workshop on Intelligent and Dynamic Scheduling for Manufacturing Systems*, pages 2–14, Florida, January

1993. Also available as Technical Report AIAI-TR-121, Artificial Intelligence Applications Institute, University of Edinburgh, UK.

[18] Howard Beck and Austin Tate. Open planning, scheduling and constraint management architectures. *British Telecommunication's Technical Journal*, 1995. Special Issue on Resource Management.

[19] Michael Beetz, Joachim Hertzberg, Malik Ghallab, and Martha Pollack. *Advances in Plan-based Control of Robotic Agents*, volume 2554 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, Heidelberg, New York, etc., 2002.

[20] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa. JADE programmer's guide, 2005. `http://jade.tilab.com/doc/programmersguide.pdf`.

[21] Julien Bidot, Susanne Biundo, and Bernd Schattenberg. Plan repair in hybrid planning. In Andreas Dengel, Karsten Berns, Thomas Breuel, Frank Bomarius, and Thomas R. Roth-Berghofer, editors, *KI 2008: Advances in Artificial Intelligence, Proceedings of the 31st German Conference on Artificial Intelligence*, volume 5243 of *Lecture Notes in Artificial Intelligence*, pages 169–176, Kaiserslautern, Germany, September 2008. Springer Verlag, Berlin, Heidelberg, New York, etc.

[22] Susanne Biundo. Present-day deductive planning. In Christer Bäckström and Erik Sandewell, editors, *Current Trends in AI Planning, Proceedings of the 2nd European Workshop on AI Planning (EWSP-93)*, pages 1–5, Vadstena, Sweeden, December 1994. IOS Press, Amsterdam, Oxford, Washington DC, Tokyo.

[23] Susanne Biundo and Maria Fox, editors. *Recent Advances in AI Planning, Proceedings of the 5th European Conference on Planning (ECP-99)*, volume 1809 of *Lecture Notes in Computer Science*, Durham, United Kingdom, September 8–10, 1999 2000. Springer Verlag, Berlin, Heidelberg, New York, etc.

[24] Susanne Biundo, Roland Holzer, and Bernd Schattenberg. Dealing with continuous resources in AI planning. In *Proceedings of the 4th International Workshop on Planning and Scheduling for Space (IWPSS'04)*, number 228 in WPP, pages 213–218, ESA-ESOC, Darmstadt, Germany, June 2004. European Space Agency Publications Division.

[25] Susanne Biundo, Roland Holzer, and Bernd Schattenberg. Project planning under temporal uncertainty. In Castillo et al. [41], pages 189–198. ISBN 1-58603-484-7.

[26] Susanne Biundo and Bernd Schattenberg. From abstract crisis to concrete relief – A preliminary report on combining state abstraction and HTN planning. In Cesta and Borrajo [48], pages 157–168. Preprint.

[27] Susanne Biundo and Werner Stephan. Modeling planning domains systematically. In Wahlster [277], pages 599–603.

[28] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):179–298, 1997.

[29] Jim Blythe and Manuela Veloso. An analysis of search techniques for a totally-ordered nonlinear planner. In Hendler [134], pages 13–19.

[30] Mark Boddy, Maria Fox, and Sylvie Thiébaux, editors. *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, Providence, Rhode Island, USA, September 2007. AAAI Press, Menlo Park, CA, USA.

[31] Mark S. Boddy. Temporal reasoning for planning and scheduling in complex domains: Lessons learned. In Austin Tate, editor, *Advanced Planning Technology Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*, pages 77–83. AAAI Press, Menlo Park, CA, USA, 1996.

[32] Mark S. Boddy. Imperfect match: PDDL 2.1 and real applications. *Journal of Artificial Intelligence Research*, 20:133–137, December 2003. Special Issue on the 3rd International Planning Competition.

[33] R. Peter Bonasso, R. James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Marc G. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1), 1997.

[34] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001. Special Issue on Heuristic Search.

[35] Daniel Borrajo and Manuela Veloso. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *Artificial Intelligence Review*, 10:1–34, 1996. Special Issue on Lazy Learning.

[36] Craig Boutilier and Ronen I. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.

[37] Peter Brucker. *Scheduling Algorithms*. Springer Verlag, Berlin, Heidelberg, New York, etc., third edition, 2001. ISBN 3-540-41510-6.

[38] Ricky W. Butler, Radu I. Siminiceanu, and César A. Muñoz. The ANMLite language and logic for specifying planning problems. Technical Report NASA/TM-2007-215088, National Aeronautics and Space Administration, Langley Research Center, Hampton, Virginia, November 2007.

[39] Giovanni Caire and Federico Pieri. LEAP users guide, 2006. `http://jade.tilab.com/doc/LEAPUserGuide.pdf`.

[40] Diego Calvanese, Giuseppe de Giacomo, and Maurizio Lenzerini. Structured objects: Modeling and reasoning. In *Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases (DOOD-95)*, volume 1013 of *Lecture Notes in Computer Science*, pages 229–246. Springer Verlag, Berlin, Heidelberg, New York, etc., 1995.

[41] Luis Castillo, Daniel Borrajo, Miguel A. Salido, and Angelo Oddi, editors. *Planning, Scheduling, and Constraint Satisfaction: From Theory to Practice*, volume 117 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, Oxford, Washington DC, Tokyo, 2005. ISBN 1-58603-484-7.

[42] Luis Castillo, Juan Fernández-Olivares, Óscar García-Pérez, and Francisco Palao. Efficiently handling temporal knowledge in an htn planner. In Long et al. [172], pages 63–72.

[43] Luis Castillo, Juan Fernández-Olivares, and Antonio González. A hybrid hierarchical/operator-based planning approach for the design of control programs. In Jörg Sauer and Jana Köhler, editors, *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000), Workshop on New Results in Planning, Scheduling and Design*, Berlin, Germany, 2000.

[44] Luis Castillo, Juan Fernández-Olivares, and Antonio González. On the adequacy of hierarchical planning characteristics for real-world problem solving. In Cesta and Borrajo [48], pages 169–180. Preprint.

[45] Luis Castillo, Juan Fernández-Olivares, and Antonio González. A flexible temporal planner. In *Iberamia 2002. I Workshop on Planning, Scheduling and Temporal Reasoning*, pages 91–102, 2002.

[46] Luis Castillo and Antonio González. MACHINE: A model of action for multi-agent domains. In Steel and Alami [246]. Poster Abstract.

[47] Roberto Cervoni, Amedeo Cesta, and Angelo Oddi. Managing dynamic temporal constraint networks. In Hammond [126], pages 13–18.

[48] Amedeo Cesta and Daniel Borrajo, editors. *Recent Advances in AI Planning, Proceedings of the 6th European Conference on Planning (ECP-01)*, Lecture Notes in Computer Science, Toledo, Spain, September 12–14 2002. Springer Verlag, Berlin, Heidelberg, New York, etc. Preprint.

[49] Amedeo Cesta, Gabriella Cortellessa, Simone Fratini, Angelo Oddi, and Nicola Policella. An innovative product for space mission planning – an a posteriori evaluation. In Boddy et al. [30], pages 57–64.

[50] Amedeo Cesta, Gabriella Cortellessa, Angelo Oddi, Nicola Policella, and Angelo Susi. A constraint-based architecture for flexible support to activity scheduling. In Floriana Esposito, editor, *AI*IA 2001: Advances in Artificial Intelligence, 7th Congress of the Italian Association for Artificial Intelligence*, volume 2175 of *Lecture Notes in Computer Science*, pages 369–381, Bari, Italy, September 2001. Springer Verlag, Berlin, Heidelberg, New York, etc.

[51] Amedeo Cesta and Angelo Oddi. Gaining efficiency and flexibility in the simple temporal problem. In Luca Chittaro, Scott D. Goodwin, Howard J. Hamilton, and Angelo Montanari, editors, *TIME-96: Proceedings of the Third International Workshop on Temporal Representation and Reasoning*, Key West, Florida, USA, May 1996. IEEE Press, Piscataway, NJ, USA.

[52] Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, 2002.

[53] Amedeo Cesta and Cristiano Stella. A time and resource problem for planning architectures. In Steel and Alami [246], pages 117–129.

[54] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[55] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran. ASPEN - Automated planning and scheduling for space mission operations. In *6th International Symposium on Space missions Operations and Ground Data Systems (SpaceOps 2000)*, Toulouse, June 19-23 2000.

[56] Steve Chien, Subbarao Kambhampati, and Craig Knoblock, editors. *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, Breckenridge, CO, USA, April 14–17 2000. AAAI Press, Menlo Park, CA, USA.

[57] Steve A. Chien. Static and completion analysis for planning knowledge base development and verification. In Drabble [75], pages 53–61.

[58] Steve A. Chien, Anita Govindjee, Tara A. Estlin, Xuemei Wang, and Randall W. Hill Jr. Automated generation of tracking plans for a network of communications antennas. In Ben Kuipers and Bonnie Webber, editors, *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 97)*, pages 963–970, Providence, Rhode Island, USA, July 1997. AAAI Press, Menlo Park, CA, USA.

[59] Bill Clancey and Daniel S. Weld, editors. *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon, USA, August 1996. AAAI Press, Menlo Park, CA, USA.

[60] Jens Claßen, Yuxiao Hu, and Gerhard Lakemeyer. A situation-calculus semantics for an expressive fragment of PDDL. In Robert C. Holte and Adele Howe, editors, *Proceedings of the 22nd National Conference on Artificial Intelligence and 19th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 07)*, pages 956–961, Vancouver, British Columbia, Canada, July 2007. AAAI Press, Menlo Park, CA, USA.

[61] Bradley J. Clement, Anthony C. Barrett, Gregg R. Rabideau, and Edmund H. Durfee. Using abstraction in planning and scheduling. In Cesta and Borrajo [48], pages 145–156. Preprint.

[62] Bradley J. Clement and Edmund H. Durfee. Theory for coordinating concurrent hierarchical planning agents using summary information. In Jim Hendler and Devika Subramanian, editors, *Proceedings of the 16th National Conference on Artificial Intelligence and 11th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 99)*, pages 495–502, Orlando, Florida, USA, July 1999. AAAI Press, Menlo Park, CA, USA.

[63] Dick Cowan and Martin Griss. Making software agent technology available to enterprise applications. Technical Report HPL-2002-211, Software Technology Laboratory, HP Laboratories, Palo Alto, July 2002. `http://www.hpl.hp.com/techreports/2002/HPL-2002-211.pdf`.

[64] Stephen E. Cross and Edward Walker. DART: Applying knowledge-based planning and scheduling to crisis action planning. In Zweben and Fox [299], chapter 25, pages 711–729. ISBN 1-55860-260-7.

[65] Ken Currie and Austin Tate. O-Plan: The open planning architecture. *Artificial Intelligence*, 52(1):46–86, 1991.

[66] William Cushing and Subbarao Kambhampati. Replanning: A new perspective. In Susanne Biundo, Karen Myers, and Kanna Rajan, editors, *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, pages 13–16, Monterey, California, USA, June 2005. AAAI Press, Menlo Park, CA, USA. Poster Session.

[67] William Cushing, Subbarao Kambhampati, Mausam, and Daniel S. Weld. When is temporal planning really temporal? In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 1852–1859, Hyderabad, India, January 2007.

[68] William Cushing and David E. Smith. The perils of discrete resource models. In Minh Do, Alan Fern, Malte Helmert, and Ioannis Refanidis, editors, *ICAPS 2007 Workshop "International Planning Competition: Past, Present & Future"*, 2007.

[69] William Cushing, Daniel S. Weld, Subbarao Kambhampati, Mausam, and Kartik Talamadupula. Evaluating temporal planning domains. In Boddy et al. [30], pages 105–112.

[70] Marc de la Asunción, Luis Castillo, Juan Fdez.-Olivares, Óscar García-Pérez, Antonio González, and Francisco Palao. Knowledge and plan execution management in planning fire fighting operations. In Castillo et al. [41], pages 149–158. ISBN 1-58603-484-7.

[71] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–91, 1991.

[72] Minh B. Do and Subbarao Kambhampati. Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20:155–194, December 2003. Special Issue on the 3rd International Planning Competition.

[73] Anders Dohn, Esben Kolind, and Jens Clausen. The manpower allocation problem with time windows and job-teaming constraints. In Boddy et al. [30], pages 120–127.

[74] Jürgen Dorn and Wolfgang Slany. A flow shop with compatibility constraints in a steelmaking plant. In Zweben and Fox [299], chapter 22, pages 629–654. ISBN 1-55860-260-7.

[75] Brian Drabble, editor. *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, Edinburgh, Scotland, May 29–31 1996. AAAI Press, Menlo Park, CA, USA.

[76] Brian Drabble and Richard Kirby. Associating A.I. planner entities with an underlying time point network. In Joachim Hertzberg, editor, *Proceedings of the 1st European Workshop on AI Planning (EWSP-91)*, volume 522 of *Lecture Notes in Computer Science*, pages 27–38, St. Augustin, Germany, March 1991. Springer Verlag, Berlin, Heidelberg, New York, etc. Also available as Technical Report AIAI-TR-94, Artificial Intelligence Applications Institute, University of Edinburgh, UK.

[77] Brian Drabble and Austin Tate. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In Hammond [126], pages 243–248.

[78] Brian Drabble and Najam ul Haq. Dynamic schedule management: Lessons from the air campaign planning domain. In Cesta and Borrajo [48]. Preprint.

[79] Stefan Edelkamp, Jeremy Frank, and Mark Kellershoff. Knowledge engineering through simulation. In Stefan Edelkamp and Jeremy Frank, editors, *Proceedings of the International Knowledge Engineering Competition*. AAAI Press, Menlo Park, CA, USA, 2007. Held in conjunction with Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07).

[80] Stefan Edelkamp and Jörg Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Computer Science Department, University of Freiburg, Germany, January 2004.

[81] Amin El-Kholy and Barry Richards. Temporal and resource reasoning in planning: The parcPLAN approach. In Wahlster [277], pages 614–618.

[82] Wolfgang Emmerich. *Engineering Distributed Objects*. John Wiley and Sons, Chichester, UK, 2000.

[83] Kutluhan Erol. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation*. PhD thesis, Department of Computer Science, The University of Maryland, 1995.

[84] Kutluhan Erol, James A. Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In Hayes-Roth and Korf [131], pages 1123–1128.

[85] Kutluhan Erol, James A. Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task network planning. In Hammond [126], pages 88–96.

[86] Kutluhan Erol, James A. Hendler, Dana S. Nau, and Reiko Tsuneto. A critical look at critics in HTN planning. In Mellish [190], pages 1592–1598.

[87] Tara A. Estlin, Steve A. Chien, and Xuemei Wang. An argument for a hybrid HTN/operator-based approach to planning. In Steel and Alami [246], pages 182–194.

[88] Tara A. Estlin, Steve A. Chien, and Xuemei Wang. Hierarchical task network and operator-based planning: Two complementary approaches to real-world planning. *Journal of Experimental and Theoretical Artificial Intelligence*, 13(4):379–395, October 2001.

[89] George Ferguson and James F. Allen. TRIPS: An integrated intelligent problem-solving assistant. In Rich and Mostow [221], pages 567–572.

[90] George Ferguson, James F. Allen, and Bradford W. Miller. TRAINS-95: Towards a mixed-initiative planning assistant. In Drabble [75], pages 70–77.

[91] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288, 1972.

[92] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[93] Eugene Fink and Manuela Veloso. Formalizing the PRODIGY planning algorithm. In Ghallab and Milani [117], pages 261–272.

[94] FIPA - Foundation for Intelligent Physical Agents. *FIPA-ACL Communicative Act Library Specification*, 2002. `http://www.fipa.org/specs/fipa00037/SC00037J.pdf`.

[95] FIPA - Foundation for Intelligent Physical Agents. *FIPA-ACL Message Structure Specification*, 2002. `http://www.fipa.org/specs/fipa00061/SC00061G.pdf`.

[96] Umberto Fonda, Antonio Natali, and Andrea Omicini. An object-oriented approach to planning. In Ute C. Sigmund and Michael Thielscher, editors, *FAPR'96 Workshop "Reasoning about Actions and Planning in Complex Environments"*, pages III–1/8, Darmstadt, Germany, 4 June 1996.

[97] Kenneth Fordyce and Gerald Sullivan. Logistics management system (LMS): Integrating decision technologies for dispatch scheduling in semiconductor manufacturing. In Zweben and Fox [299], chapter 17, pages 473–516. ISBN 1-55860-260-7.

[98] Maria Fox and Alexandra M. Coddington, editors. *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS-02) Workshop on Planning for Temporal Domains*, 2002.

[99] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, December 2004. Special Issue on the 3rd International Planning Competition.

[100] Maria Fox, Derek Long, and Keith Halsey. An investigation into the expressive power of PDDL2.1. In Ramon López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*, volume 110 of *Frontiers in Artificial Intelligence and Applications*, pages 328–342, Valencia, Spain, August 22–27 2004. IOS Press, Amsterdam, Oxford, Washington DC, Tokyo.

[101] Marc Friedman and Daniel S. Weld. Least commitment action selection. In Drabble [75], pages 86–93.

[102] Alex Fukunaga, Gregg Rabideau, Steve A. Chien, and David Yan. ASPEN: A framework for automated planning and scheduling of spacecraft control and operations. In *Proceedings of the 1997 International Symposium on Artificial Intelligence, Robotics and Automation for Space*, pages 181–187, Tokyo, Japan, July 1997.

[103] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1994. ISBN 0-201-63361-2.

[104] Frederick Garcia and Philippe Laborie. Hierarchisation of the seach space in temporal planning. In Ghallab and Milani [117], pages 217–232.

[105] Antonio Garrido, Maria Fox, and Derek Long. A temporal planning system for durative actions of PDDL2.1. In Frank van Harmelen, editor, *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-2002)*, volume 77 of *Frontiers in Artificial Intelligence and Applications*, pages 586–590, Lyon, France, July 2002. IOS Press, Amsterdam, Oxford, Washington DC, Tokyo.

[106] Antonio Garrido, Eva Onaindia, and Federico Barber. A temporal planning system for time-optimal planning. In Pavel Brazdil and Alípio Jorge, editors, *Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving, 10th Portuguese Conference on Artificial Intelligence, EPIA 2001*, volume 2258 of *Lecture Notes in Computer Science*, pages 379–392, Porto, Portugal, December 2001. Springer Verlag, Berlin, Heidelberg, New York, etc.

[107] Antonio Garrido, Miguel A. Salido, and Federico Barber. Scheduling in a planning environment. In Jörg Sauer and Jana Köhler, editors, *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000), Workshop on New Results in Planning, Scheduling and Design*, pages 36–43, Berlin, Germany, 2000.

[108] Michael P. Georgeff and Amy L. Lansky, editors. *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, Timberline, Oregon, June-July 1986. Morgan Kaufmann, San Francisco, CA, USA.

[109] Alfonso Gerevini and Lenhart K. Schubert. Efficient algorithms for qualitative reasoning about time. *Artificial Intelligence*, 74(2):207–248, 1995.

[110] Alfonso E. Gerevini, Ugur Kuter, Dana S. Nau, Alessandro Saetti, and Nathaniel Waisbrot. Combining domain-independent planning and HTN planning: The duet planner. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikos Avouris, editors, *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-2008)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, Patras, Greece, July 21–25 2008. IOS Press, Amsterdam, Oxford, Washington DC, Tokyo.

[111] Alfonso E. Gerevini and Derek Long. Plan constraints and preferences in PDDL3. Technical Report RT 2005-08-47, Department of Electronics for Automation, University of Brescia, Italy, 2005.

[112] Alfonso E. Gerevini, Alessandro Saetti, and Ivan Serina. An approach to temporal planning and scheduling in domains with predictable exogenous events. *Journal of Artificial Intelligence Research*, 25:187–231, 2006.

[113] Alfonso E. Gerevini and Lenhart K. Schubert. Accelerating partial-order planners: Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research*, 5:95–137, 1996.

[114] Sascha Geßler. Ein Framework zur interaktiven hybriden Handlungsplanung. Master's thesis, Ulm University, 2008. (A framework for interactive hybrid planning).

[115] Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors. *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, Toulouse, France, April 23–27 2002. AAAI Press, Menlo Park, CA, USA.

[116] Malik Ghallab and Hervé Laruelle. Representation and control in IxTeT, a temporal planner. In Hammond [126], pages 61–67.

[117] Malik Ghallab and Alfredo Milani, editors. *New Directions in AI Planning, Proceedings of the 3rd European Workshop on AI Planning (EWSP-95)*, volume 31 of *Frontiers in Artificial Intelligence*, Assisi, Italy, September 1996. IOS Press, Amsterdam, Oxford, Washington DC, Tokyo.

[118] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning - theory and practice*. Morgan Kaufmann, San Francisco, CA, USA, first edition, May 2004. ISBN 1-55860-856-7.

[119] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In Martha E. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1221–1226, Nagoya, Japan, August 1997. Morgan Kaufmann, San Francisco, CA, USA.

[120] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In Rich and Mostow [221], pages 623–630.

[121] Fausto Giunchiglia. Using Abstrips abstractions – where do we stand? IRST Technical Report 9607-10, Istituto Trentino di Cultura, Povo, 38100 Trento, Italy, January 1997.

[122] Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–389, 1992.

[123] Robert P. Goldman. Durative planning in htns. In Long et al. [172], pages 382–385.

[124] C. Cordell Green. Application of theorem proving to problem solving. In Donald E. Walker and Lewis M. Morton, editors, *Proceedings of the 1st International Joint Conference on Artificial Intelligence (IJCAI-69)*, pages 219–240, Bedford, Massachusetts, 1969. William Kaufmann.

[125] Volker Haarslev and Ralf Möller. Description of the racer system and its applications. In Carole Goble, Deborah L. McGuinness, Ralf Möller, and Peter F. Patel-Schneider, editors, *Working Notes*

of the 2001 International Description Logics Workshop (DL-2001), volume 49 of *CEUR Workshop Proceedings*, 2001.

[126] Kirk Hammond, editor. *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, Chicago, IL, June 13–15 1994. AAAI Press, Menlo Park, CA, USA.

[127] Kristian J. Hammond. Case-based planning: A framework for planning from experience. *Cognitive Science*, 14(3):385–443, 1990.

[128] David Harel, Dexter Kozen, and Jerzy Tiuryn. Dynamic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic Volume II — Extensions of Classical Logic*, pages 497–604. D. Reidel Publishing Company: Dordrecht, The Netherlands, 1984.

[129] Perter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics (SSC)*, 4(2):100–107, 1968.

[130] Patrik Haslum. Quality of solutions to IPC5 benchmark problems: Preliminary results. In Minh Do, Alan Fern, Malte Helmert, and Ioannis Refanidis, editors, *ICAPS 2007 Workshop "International Planning Competition: Past, Present & Future"*, 2007.

[131] Barbara Hayes-Roth and Richard E. Korf, editors. *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, Seattle, Washington, USA, July 1994. AAAI Press, Menlo Park, CA, USA.

[132] Malte Helmert. On the complexity of planning in transportation domains. In Cesta and Borrajo [48], pages 349–360. Preprint.

[133] Malte Helmert. *Understanding Planning Tasks: Domain Complexity and Heuristic Decomposition*, volume 4929 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Heidelberg, New York, etc., 2008. ISBN 978-3-540-77722-9.

[134] James Hendler, editor. *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems (AIPS-92)*, College Park, Maryland, June 15–19 1992. Morgan Kaufmann, San Francisco, CA, USA.

[135] Charles A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 583, October 1969.

[136] Jörg Hoffmann. FF: The fast-forward planning system. *AI Magazine*, 22(3):57–62, 2001.

[137] Jörg Hoffmann and Stefan Edelkamp. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579, 2005.

[138] Jörg Hoffmann, Stefan Edelkamp, Sylvie Thiébaux, Roman Englert, Frederico dos Santos Liporace, and Sebastian Trüg. Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4. *Journal of Artificial Intelligence Research*, 26:453–541, 2006.

[139] Ian Horrocks, Frank van Harmelen, and Peter F. Patel-Schneider. DAML+OIL Specification (March 2001), 2001. `http://www.daml.org/2001/03/daml+oil-index.html`.

[140] Mark D. Johnston and Glenn E. Miller. SPIKE: Intelligent scheduling of Hubble space telescope observations. In Zweben and Fox [299], chapter 14, pages 391–422. ISBN 1-55860-260-7.

[141] Albert Jones and Luis C. Rabelo. Survey of job shop scheduling techniques. NISTIR 9820, National Institute of Standards and Technology, Gaithersburg, MD, 1998.

[142] Ari K. Jónsson, Paul H. Morris, Nicola Muscettola, Kanna Rajan, and Ben Smith. Planning in interplanetary space: Theory and practice. In Chien et al. [56], pages 177–186.

[143] David Joslin and Martha E. Pollack. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In Hayes-Roth and Korf [131], pages 1004–1009.

[144] David Joslin and Martha E. Pollack. Is "early commitment" in plan generation ever a good idea. In Clancey and Weld [59], pages 1188–1193.

[145] Subbarao Kambhampati. On the utility of systematicity: Understanding tradeoffs between redundancy and commitment in partial-ordering planning. In Bajcsy [14], pages 1380–1387.

[146] Subbarao Kambhampati. A comparative analysis of partial order planning and task reduction planning. *SIGART Bulletin*, 6(1):16–25, 1995.

[147] Subbarao Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18(2):67–97, Summer 1997.

[148] Subbarao Kambhampati, Craig A. Knoblock, and Qiang Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence*, 76(1-2):167–238, 1995.

[149] Subbarao Kambhampati, Amol Dattatraya Mali, and Biplav Srivastava. Hybrid planning for partially hierarchical domains. In Rich and Mostow [221], pages 882–888.

[150] Subbarao Kambhampati and Dana S. Nau. On the nature and role of modal truth criteria in planning. *Artificial Intelligence*, 82(2):129–155, 1996.

[151] Henry Kautz and Bart Selman. Planning as satisfiablility. In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92)*, pages 359–363, Vienna, Austria, August 3–7 1992. John Wiley and Sons, Chichester, UK.

[152] Henry A. Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In Thomas [265], pages 318–325.

[153] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.

[154] Craig A. Knoblock. Generating parallel execution plans with a partial-order planner. In Hammond [126], pages 98–103.

[155] Craig A. Knoblock, Josh D. Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In Maybury [176], pages 692–697.

[156] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The graph isomorphism problem: Its Structural Complexity*. Progress in Theoretical Computer Science. Birkhäuser Verlag, Boston-Basel-Berlin-Stuttgart, 1993. ISBN 978-0-8176-3680-7.

[157] Richard E. Korf. Depth-first iterative-deepening – an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, September 1985.

[158] Richard E. Korf. Artificial intelligence search algorithms. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 36. CRC Press, 1998.

[159] Jana Köhler. Planning under resource constraints. In Henri Prade, editor, *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 489–493, Brighton, UK, August 23–28 1998. John Wiley and Sons, Chichester, UK.

[160] Jana Köhler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, 2000.

[161] Philippe Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143(2):151–188, February 2003.

[162] Philippe Laborie. Resource temporal networks: Definition and complexity. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 948–953, Acapulco, Mexico, August 2003. Morgan Kaufmann, San Francisco, CA, USA.

[163] Philippe Laborie and Malik Ghallab. IxTeT: an integrated approach for plan generation and scheduling. In *4th IEEE-INRIA Symposium on Emerging Technologies and Factory Automation (ETFA'95)*, pages 485–495, Paris, October 1995.

[164] Philippe Laborie and Malik Ghallab. Planning with sharable resource constraints. In Mellish [190], pages 1643–1651.

[165] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[166] Pat Langley. Systematic and nonsystematic search strategies. In Hendler [134], pages 145–152.

[167] David B. Leake and Andrew Kinley. Integrating cbr components within a case-based planner. In *Proceedings of the AAAI-98 Workshop on Case-Based Reasoning Integrations*, San Mateo, CA, 1998. AAAI Press, Menlo Park, CA, USA.

[168] Olivier Lhomme. Consistency techniques for numeric CSPs. In Bajcsy [14], pages 232–238.

[169] Vladimir Lifschitz. On the semantics of STRIPS. In Georgeff and Lansky [108], pages 1–9.

[170] A. R. Lingard and E. B. Richards. Planning parallel actions. *Artificial Intelligence*, 99(2):261–324, 1998.

[171] Derek Long and Maria Fox. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, December 2003. Special Issue on the 3rd International Planning Competition.

[172] Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey, editors. *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS-06)*, Ambleside, The English Lake District, UK, June 2006. AAAI Press, Menlo Park, CA, USA.

[173] Amol Mali and Subbarao Kambhampati. Encoding HTN planning in propositional logic. In Simmons et al. [238], pages 190–198.

[174] Frank Manola and Eric Miller. RDF primer, 2003. `http://www.daml.org/2001/03/daml+oil-index.html`.

[175] Eliseo Marzal, Eva Onaindia, and Laura Sebastia. An incremental temporal partial-order planner. In Fox and Coddington [98], pages 26–32.

[176] Mark T. Maybury, editor. *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)*, Anaheim, California, USA, July 1991. AAAI Press, Menlo Park, CA, USA.

[177] David McAllester and David Rosenblitt. Systematic nonlinear planning. In Maybury [176], pages 634–639.

[178] Brian McBride. Making software agent technology available to enterprise applications, 2000. `http://www-uk.hpl.hp.com/people/bwm/papers/20001221-paper/`.

[179] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[180] Thomas L. McCluskey. Object transition sequences: A new form of abstraction for HTN planners. In Chien et al. [56], pages 216–225.

[181] Thomas L. McCluskey. PDDL: A language with a purpose? In Derek Long, Drew McDermott, and Sylvie Thiebaux, editors, *Proceedings of ICAPS'03 Workshop on PDDL*, Trento, Italy, June 2003.

[182] Thomas L. McCluskey and Diane E. Kitchin. A tool-supported approach to engineering HTN planning models. In *Proceedings of the tenth IEEE International Conference on Tools with Artificial Intelligence (ICTAI'98)*, pages 272–279, Taipei, Taiwan, November 1998. IEEE Press, Piscataway, NJ, USA.

[183] Thomas L. McCluskey, Diane E. Kitchin, and Julie M. Porteous. Object-centred planning: Lifting classical planning from the literal level to the object level. In *Proceedings of 11th IEEE International Conference on Tools with Artificial Intelligence*, Toulouse, 1996. IEEE Press, Piscataway, NJ, USA.

[184] Thomas L. McCluskey, Donghong Lui, and R.M. Simpson. Using knowledge engineering and state-space planning techniques to optimise an HTN planner. In *Proceedings of PlanSig 2002*, pages 1–11, Delft University of Technology, The Netherlands, 2002. ISBN 1-3685-70-8.

[185] Drew McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[186] Drew McDermott. Reasoning about plans. In Jerry R. Hobbs and Robert C. Moore, editors, *Formal Theories of the Commonsense World*, pages 269–317. Ablex Publishing Corp., Norwood, NJ, 1985.

[187] Drew McDermott. PDDL, the planning domain definition language. Technical report, Yale Center for Computational Vision and Control, 1998. `ftp://ftp.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz`.

[188] Drew McDermott. PDDL2.1 – the art of the possible? *Journal of Artificial Intelligence Research*, 20:145–148, December 2003. Special Issue on the 3rd International Planning Competition.

[189] Drew McDermott. *The Opt Manual*. Yale University, Faculty for Computer Science, November 2005. Version 1.7.7, Draft.

[190] Chris S. Mellish, editor. *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Cananda, August 1995. Morgan Kaufmann, San Francisco, CA, USA.

[191] Steven Minton, John Bresina, and Mark Drummond. Commitment strategies in planning: A comparative analysis. In Mylopoulos and Reiter [196], pages 259–265.

[192] Steven Minton, John Bresina, and Mark Drummond. Total-order and partial-order planning: A comparative analysis. *Journal of Artificial Intelligence Research*, 2:227–262, November 30 1994.

[193] Nicola Muscettola. HSTS: Integrating planning and scheduling. In Zweben and Fox [299], pages 169–212. ISBN 1-55860-260-7.

[194] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: to go boldly where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.

[195] Hector Muñoz-Avila, D. W. Aha, L. Breslow, and Dana S. Nau. HICAP: an interactive case-based planning architecture and its application to noncombatant evacuation operations. In Ramasamy Uthurusamy and Barbara Hayes-Roth, editors, *The Eleventh Innovative Applications of Artificial Intelligence Conference on Artificial Intelligence (IAAI-99)*, pages 870–875. American Association on Artificial Intelligence, AAAI Press, Menlo Park, CA, USA, 1999.

[196] John Mylopoulos and Raymond Reiter, editors. *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, August 1991. Morgan Kaufmann, San Francisco, CA, USA.

[197] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *WWW '02: Proceedings of the 11th International Conference on World Wide Web*, pages 77–88, New York, NY, USA, 2002. ACM Press.

[198] Dana S. Nau. Current trends in automated planning. *AI Magazine*, 28(4):43–58, 2007.

[199] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, December 2003. Special Issue on the 3rd International Planning Competition.

[200] Dana S. Nau, Yue Cao, Amnon Lotem, and Hector Muñoz-Avila. SHOP: Simple hierarchical ordered planner. In Thomas [265], pages 968–975.

[201] Dana S. Nau, Yue Cao, Amnon Lotem, and Hector Muñoz-Avila. SHOP and M-SHOP: Planning with ordered task decomposition. Technical Report CS TR 4157, University of Maryland, College Park, MD, June 2000.

[202] Dana S. Nau, Hector Muñoz-Avila, Yue Cao, Amnon Lotem, and Steven Mitchell. Total-order planning with partially ordered subtasks. In Nebel [204], pages 425–430.

[203] Bernhard Nebel. Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ORD-Horn class. In Wahlster [277], pages 38–42.

[204] Bernhard Nebel, editor. *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle, Washington, USA, August 2001. Morgan Kaufmann, San Francisco, CA, USA.

[205] Bernhard Nebel and Hans-Jürgen Bürckert. Reasoning about temporal relations: A maximal tractable subclass of allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.

[206] Bernhard Nebel, Charles Rich, and William R. Swartout, editors. *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, Cambridge, Massachusetts, October 1992. Morgan Kaufmann, San Francisco, CA, USA.

[207] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In Nebel [204], pages 459–466.

[208] H. Penny Nii. The blackboard model of problem solving. *AI Magazine*, 7(2):38–53, 1986.

[209] Nils Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

[210] Jeff Orkin. Three states and a plan: The AI of F.E.A.R. In *Proceedings of the Game Developer's Conference*, 2006.

[211] Edwin P. D. Pednault. Formulating multiagent, dynamic-world problems in the classical planning framework. In Georgeff and Lansky [108], pages 47–82.

[212] Edwin P. D. Pednault. Generalizing nonlinear planning to handle complex goals and actions with context-dependent effects. In Mylopoulos and Reiter [196], pages 240–245.

[213] Edwin P.D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors, *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 324–332, Toronto, Ontario, Canada, May 1989. Morgan Kaufmann, San Francisco, CA, USA.

[214] J. Scott Penberthy and Daniel S. Weld. UCPOP - a sound, complete, partial order planner for ADL. In Nebel et al. [206], pages 103–114.

[215] M. A. Peot and D. Smith. Threat removal strategies for partial-order planning. In Richard Fikes and Wendy Lehnert, editors, *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pages 492–499, Washington, D.C., USA, July 1993. AAAI Press, Menlo Park, CA, USA.

[216] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, second edition, 2001. ISBN 0-13-028138-7.

[217] PLANFORM: An open environment for building planners. `http://scom.hud.ac.uk/planform/`. Project web site.

[218] Martha E. Pollack, David Joslin, and Massimo Paolucci. Flaw selection strategies for partial-order planning. *Journal of Artificial Intelligence Research*, 6:223–262, 1997.

[219] Raj Reddy, editor. *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI-77)*, Cambridge, MA, August 1977. Morgan Kaufmann, San Francisco, CA, USA.

[220] Raymond Reiter. Natural actions, concurrency and continuous time in the situation calculus. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR-96)*, pages 2–13, Cambridge, Massachusetts, USA, November 1996. Morgan Kaufmann, San Francisco, CA, USA.

[221] Charles Rich and Jack Mostow, editors. *Proceedings of the 15th National Conference on Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 98)*, Madison, Visconsin, USA, July 1998. AAAI Press, Menlo Park, CA, USA.

[222] Jussi Rintanen and Hartmut Jungholt. Numeric state variables in constraint-based planning. In Biundo and Fox [23], pages 109–121.

[223] Stuart Russell and Peter Norvig. *Artifical Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, first edition, 1995. ISBN 0-13-103805-2.

[224] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[225] Earl D. Sacerdoti. The nonlinear nature of plans. In Reddy [219], pages 206–214.

[226] Norman Sadeh. Micro-opportunistic scheduling: The micro-boss factory scheduler. In Zweben and Fox [299], chapter 4, pages 99–135. ISBN 1-55860-260-7.

[227] Jürgen Sauer and Rene Schumann. Modelling and solving workforce scheduling problems. In Stefan Edelkamp, Jürgen Sauer, and Bernd Schattenberg, editors, *Proceedings of the 21st Workshop "Planen, Scheduling und Konfigurieren, Entwerfen" (PUK 2007)*, pages 93–101, Osnabrück, Germany, September 2007.

[228] Michael Schalk, Thorsten Liebig, Thorsten Illmann, and Frank Kargl. Combining FIPA ACL with DAML+OIL - a case study. In Stephen Cranefield, Tim Finin, and Steve Willmott, editors, *Proceedings of the Second International Workshop on Ontologies in Agent Systems*, Bologna, Italy, July 2002.

[229] Bernd Schattenberg, Steffen Balzer, and Susanne Biundo. Knowledge-based middleware as an architecture for planning and scheduling systems. In Long et al. [172], pages 422–425.

[230] Bernd Schattenberg, Steffen Balzer, and Susanne Biundo. Semantic web technology as a basis for planning and scheduling systems. In Jürgen Sauer, editor, *Proceedings of the 20th Workshop "Planen und Konfigurieren´´ (PuK'06)*, pages 26–36, University of Bremen, 2006. ISBN 3-88722-670-4.

[231] Bernd Schattenberg, Julien Bidot, and Susanne Biundo. On the construction and evaluation of flexible plan-refinement strategies. In Joachim Hertzberg, Michael Beetz, and Roman Englert, editors, *KI 2007: Advances in Artificial Intelligence, Proceedings of the 30th German Conference on Artificial Intelligence*, volume 4667 of *Lecture Notes in Artificial Intelligence*, pages 367–381, Osnabrück, Germany, September 2007. Springer Verlag, Berlin, Heidelberg, New York, etc.

[232] Bernd Schattenberg and Susanne Biundo. On the identification and use of hierarchical resources in planning and scheduling. In Ghallab et al. [115], pages 263–272.

[233] Bernd Schattenberg and Susanne Biundo. A unifying framework for hybrid planning and scheduling. In Christian Freksa, Michael Kohlhase, and Kerstin Schill, editors, *KI 2006: Advances in Artificial Intelligence, Proceedings of the 29th German Conference on Artificial Intelligence*, volume 4314 of *Lecture Notes in Artificial Intelligence*, pages 361–373, Bremen, Germany, June 2007. Springer Verlag, Berlin, Heidelberg, New York, etc.

[234] Bernd Schattenberg, Andreas Weigl, and Susanne Biundo. Hybrid planning using flexible strategies. In Ulrich Furbach, editor, *KI 2005: Advances in Artificial Intelligence, Proceedings of the 28th German Conference on Artificial Intelligence*, volume 3698 of *Lecture Notes in Artificial Intelligence*, pages 258–272, Koblenz, Germany, September 2005. Springer Verlag, Berlin, Heidelberg, New York, etc.

[235] Robert Schrag, Mark Boddy, and Jim Carciofini. Managing disjunction for practical temporal reasoning. In Nebel et al. [206], pages 36–46.

[236] Lenhart K. Schubert and Alfonso E. Gerevini. Accelerating partial order planners by improving plan and goal choices. In *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence*, pages 442–450, Herndon, Virginia, 1995. IEEE Computer Society Press.

[237] Amílcar Sernadas, Cristina Sernadas, and José Félix Costa. Object specification logic. *Journal of Logic and Computation*, 5(5):603–630, 1995.

[238] Reid Simmons, Manuela Veloso, and Stephen Smith, editors. *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, Pittsburgh, PA, USA, June 7–10 1998. AAAI Press, Menlo Park, CA, USA.

[239] David E. Smith, Jeremy Frank, and Ari K. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1):47–83, 2000.

[240] David E. Smith and Mark A. Peot. Suspending recursion in causal-link planning. In Drabble [75], pages 182–190.

[241] David E. Smith and Daniel S. Weld. Temporal planning with mutual exclusion reasoning. In Thomas [265], pages 326–337.

[242] Stephen J. J. Smith, Kiran Hebbar, Dana S. Nau, and Ioannis Minis. Integrating electrical and mechanical design and process planning. In Martti Mäntylä, Susan Finger, and Tetsuo Tomiyama, editors, *Knowledge Intensive CAD*, volume 2, pages 269–288. Chapman and Hall, 1997.

[243] Stephen J.J. Smith, Dana S. Nau, and Thomas A. Throop. Computer bridge: A big win for AI planning. *AI Magazine*, 19(2):93–105, 1998.

[244] Biplav Srivastava and Subbarao Kambhampati. Scaling up planning by teasing out resource scheduling. In Biundo and Fox [23], pages 172–186.

[245] Scott Stark. *JBoss Administration and Development*. JBoss Group, LLC, Atlanta, USA, second edition, January 2003. JBoss Version 3.0.5.

[246] Sam Steel and Rachid Alami, editors. *Recent Advances in AI Planning, Proceedings of the 4th European Conference on Planning (ECP-97)*, volume 1348 of *Lecture Notes in Computer Science*, Toulouse, France, September 24–26 1997. Springer Verlag, Berlin, Heidelberg, New York, etc.

[247] Werner Stephan and Susanne Biundo. A new logical framework for deductive planning. In Bajcsy [14], pages 32–38.

[248] Werner Stephan and Susanne Biundo. Deduction-based refinement planning. In Drabble [75], pages 213–220.

[249] Oliviero Stock, editor. *Spatial and Temporal Reasoning*. Springer Verlag, Berlin, Heidelberg, New York, etc., 1997. ISBN 978-0-7923-4644-9.

[250] Jeroen Groenendijkand Martin Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14(1):39–100, 1991.

[251] Sun Microsystems. *Simplified Guide to the Java 2 Platform, Enterprise Edition*, 1999. `http://java.sun.com/j2ee/white/j2ee_guide.pdf`.

[252] Sun Microsystems. *Enterprise JavaBeans 2.1 Documentation*, 2003. `http://java.sun.com/products/ejb/docs.html`.

[253] Gerald J. Sussman. *A Computational Model of Skill Acquisition*. American Elsevier, New York , NY , USA, 1975.

[254] Austin Tate. INTERPLAN: A plan generation system which can deal with interactions between goals. memorandum MIP-R-109, Machine Intelligence Research Unit, University of Edinburgh, December 1974.

[255] Austin Tate. Generating project networks. In Reddy [219], pages 888–893.

[256] Austin Tate. Intelligible AI planning. In *Research and Development in Intelligent Systems XVII, Proceedings of the ES2000, The 20th British Computer Society Special Group on Expert Systems International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, pages 3–16, Cambridge, UK, December 2000. Springer Verlag, Berlin, Heidelberg, New York, etc.

[257] Austin Tate and Jeff Dalton. O-plan: a common lisp planning web servide. In *Proceedings of the International Lisp Conference*, pages 12–25, New York, USA, October 2003.

[258] Austin Tate, Brian Drabble, and Jeff Dalton. The use of condition types to restrict search in an AI planner. In Hayes-Roth and Korf [131], pages 1129–1134.

[259] Austin Tate, Brian Drabble, and Jeff Dalton. O-Plan: a knowledge-based planner and its application to logistics. In Austin Tate, editor, *Advanced Planning Technology*. AAAI Press, Menlo Park, CA, USA, May 1996. ISBN 0-929280-98-9.

[260] Austin Tate, Brian Drabble, and Richard Kirby. Spacecraft command and control using AI planning techniques – the O-Plan2 projcect. Technical Report RL-TR-92-217, Rome Laboratory, Air Force Systems Command, August 1992.

[261] Austin Tate, Brian Drabble, and Richard Kirby. O-Plan2: An architecture for command, planning and control. In Zweben and Fox [299], pages 213–240. ISBN 1-55860-260-7.

[262] Austin Tate, John Levine, Peter Jarvis, and Jeff Dalton. Using AI planning technology for army small unit operations. In Chien et al. [56], pages 379–386.

[263] Snehal Thakkar, José Luis Ambite, and Craig A. Knoblock. Composing, optimizing, and executing plans for bioinformatics web services. *The VLDB Journal*, 14(3):330–353, 2005.

[264] Sylvie Thièbaux, Jörg Hoffmann, and Bernhard Nebel. In defense of PDDL axioms. *Artificial Intelligence*, 168(1-2):38–69, 2005.

[265] Dean Thomas, editor. *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden, July 31–August 6 1999. Morgan Kaufmann, San Francisco, CA, USA.

[266] Romain Trinquart, Solange Lemai, and Stephane Cambon. One step on the left, one step on the right, and back to the middle: Exploring temporal domains in a POP fashion. In Fox and Coddington [98], pages 82–90.

[267] Grigorios Tsoumakas, Georgios Meditskos, Dimitris Vrakas, Nick Bassiliades, and Ioannis Vlahavas. Web services for adaptive planning. In Castillo et al. [41], pages 159–168. ISBN 1-58603-484-7.

[268] Reiko Tsuneto, Kutluhan Erol, James A. Hendler, and Dana S. Nau. Commitment strategies in hierarchical task network planning. In Clancey and Weld [59], pages 536–542.

[269] Reiko Tsuneto, James A. Hendler, and Dana S. Nau. Analyzing external conditions to improve the efficiency of HTN planning. In Rich and Mostow [221], pages 913–920.

[270] Reiko Tsuneto, James A. Hendler, Dana S. Nau, and Leliane Nunes De Barros. Matching problem features with task selection for better performance in HTN planning. In Simmons et al. [238]. Workshop on Knowledge Engineering and Acquisition for Planning.

[271] Reiko Tsuneto, Dana S. Nau, and James A. Hendler. Plan-refinement strategies and search-space size. In Steel and Alami [246], pages 414–426.

[272] Peter van Beek and Robin Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence Journal*, 6:132–144, 1990.

[273] Manuela M. Veloso and Jim Blythe. Linkability: Examining causal link commitments in partial-order planning. In Hammond [126], pages 170–175.

[274] Manuela M. Veloso, Jaime Carbonell, Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1), 1995.

[275] Manuela M. Veloso and Peter Stone. FLECS: Planning with a flexible commitment strategy. *Journal of Artificial Intelligence Research*, 3:25–52, 1995.

[276] Lluis Vila. A survey on temporal reasoning in artificial intelligence. *AI Communications*, 7(1):4–28, 1994.

[277] Wolfgang Wahlster, editor. *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, Budapest, Hungary, August 12–16 1996. John Wiley and Sons, Chichester, UK.

[278] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999. ISBN 0-585-10830-7.

[279] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[280] Daniel S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.

[281] David E. Wilkins. *Practical Planning: Extending the AI Planning Paradigm*. Morgan Kaufmann, San Francisco, CA, USA, 1988. ISBN 0-934613-94-X.

[282] David E. Wilkins. Can AI planners solve practical problems? *Computational Intelligence Journal*, 6(4):232–246, 1990.

[283] David E. Wilkins. *Using the SIPE-2 Planning System: A Manual for SIPE-2, Version 6.1*. SRI International Artificial Intelligence Center, 333 Ravenswood Ave., Menlo Park, California 94025, July 1999.

[284] David E. Wilkins and Roberto V. Desimone. Applying an AI planner to military operations planning. In Zweben and Fox [299], pages 685–709. ISBN 1-55860-260-7.

[285] David E. Wilkins and Marie desJardins. A call for knowledge-based planning. *AI Magazine*, 22(1):99–115, 2001.

[286] David E. Wilkins and Karen L. Myers. A multiagent planning architecture. In Simmons et al. [238], pages 154–163.

[287] Matthew L. Ginsberg William D. Harvey. Limited discrepancy search. In Mellish [190], pages 607–615.

[288] Mike Williamson and Steve Hanks. Flaw selection strategies for value-directed planning. In Drabble [75], pages 237–244.

[289] Michael Wooldridge and Nicholas R. Jennings. Pitfalls of agent-oriented development. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 385–391, New York, 9-13, 1998. ACM Press.

[290] Dan Wu and Dana S. Nau. UM-Translog-2: A planning domain designed for AIPS-2002. Technical Report CS-TR-4402, UMIACS-TR-2002-82, University of Maryland, 2002.

[291] Qiang Yang. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, 6:12–24, 1990.

[292] Qiang Yang. *Intelligent Planning: A Decomposition and Abstraction Based Approach to Classical Planning*. Springer Verlag, Berlin, Heidelberg, New York, etc., 1997. ISBN 3-540-61901-1.

[293] Qiang Yang, Philip W. L. Fong, and Edward Kim. Design patterns for planning systems. In Leliane Nunes de Barros, Richard Benjamins, Yuval Shahar, Austin Tate, and Andre Valente, editors, *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98) Workshop on Knowledge Engineering and Acquisition for Planning: Bridging Theory and Practice*, pages 104–112, Pittsburgh, PA, USA, 1998. AAAI Press, Menlo Park, CA, USA.

[294] Qiang Yang, Josh Tenenberg, and Steven Woods. On the implementation and evaluation of Abtweak. *Computational Intelligence Journal*, 12(2):295–318, 1996.

[295] Neil Yorke-Smith. Exploiting the structure of hierarchical plans in temporal constraint propagation. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 20th National Conference on Artificial Intelligence and 17th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI 05)*, pages 1223–1228, Pittsburgh, Pennsylvania, USA, July 2005. AAAI Press, Menlo Park, CA, USA.

[296] Håkan L. S. Younes and Reid G. Simmons. On the role of ground actions in refinement planning. In Ghallab et al. [115], pages 54–62.

[297] Håkan L. S. Younes and Reid G. Simmons. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, December 2003. Special Issue on the 3rd International Planning Competition.

[298] R. Michael Young, Martha E. Pollack, and Johanna D. Moore. Decomposition and causality in partial order planning. In Hammond [126], pages 188–193.

[299] Monte Zweben and Mark Fox, editors. *Intelligent Scheduling*. Morgan Kaufmann, San Francisco, CA, USA, San Mateo, CA, 1994. ISBN 1-55860-260-7.

[300] Éric Jacopin, Claude Le Pape, and Jean-François Puget. A theoretical analysis of the "uselessness" of white knights. Technical Report 92/27, LAFORIA, October 1992.