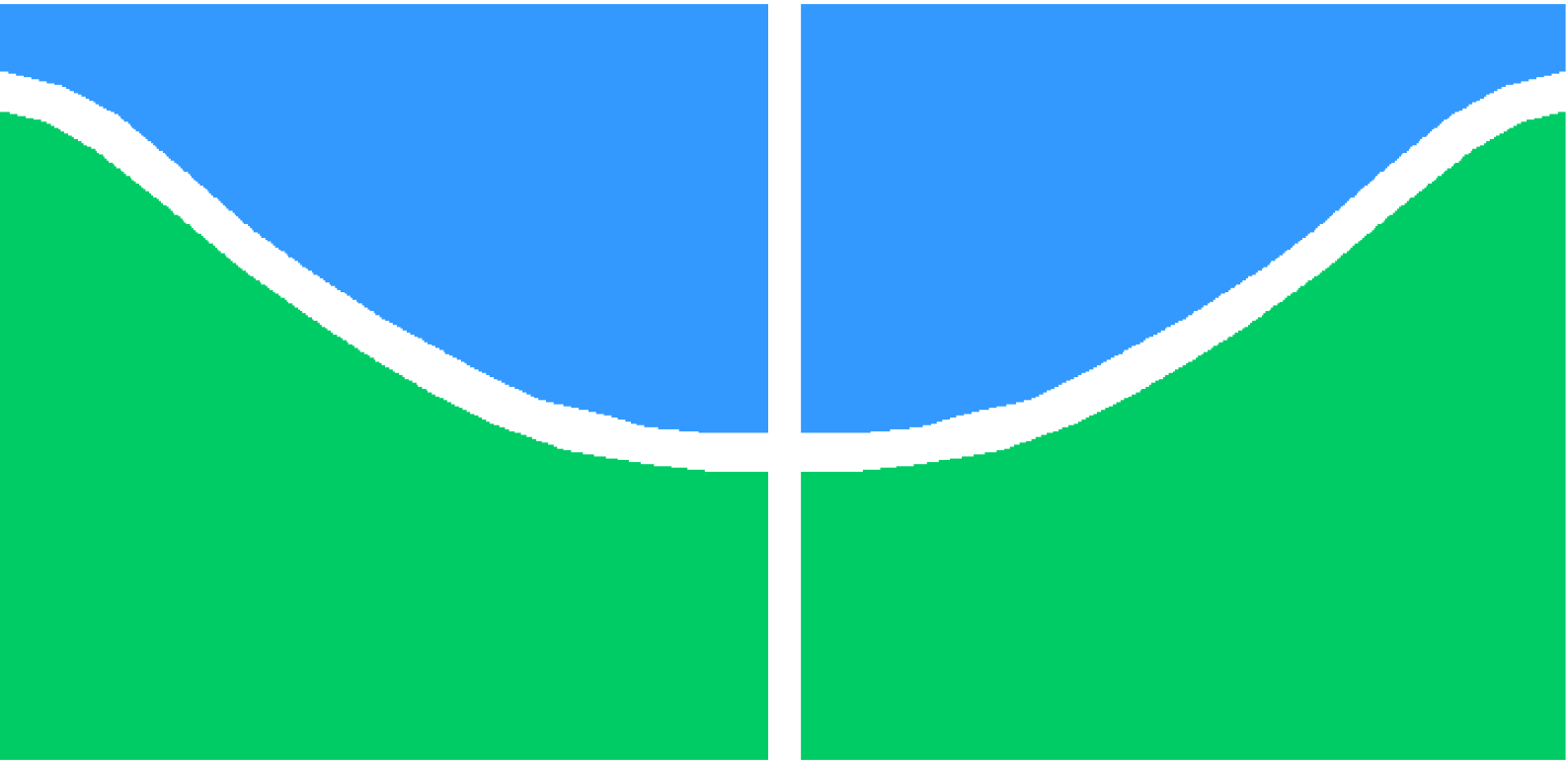


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Resolvendo Fórmulas Horn Mistas com um Algoritmo $O(2^{0.5284 \times N})$

Autor: André Lucas de Sousa Pinto
Orientador: Prof. Dr. Bruno César Ribas

Brasília, DF
2022

André Lucas de Sousa Pinto

Resolvendo Fórmulas Horn Mistas com um Algoritmo $O(2^{0.5284 \times N})$

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 05 de maio de 2022:

Prof. Dr. Bruno César Ribas
Orientador

Prof. Dr. John Lenon Cardoso Gardenghi
Convidado 1

Prof. Dr. Razer Anthom Nizer Rojas Montaño
Convidado 2

Brasília, DF
2022

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

PP659r

Pinto, André Lucas de Sousa
Resolvendo Fórmulas Horn Mistas com um Algoritmo O(2^(0.5284*n)) / André Lucas de Sousa Pinto; orientador Bruno César Ribas. -- Brasília, 2022.
63 p.

Monografia (Graduação - Engenharia de Software) -- Universidade de Brasília, 2022.

1. Satisfatibilidade Booleana. 2. Fórmula Horn. 3. Fórmula Horn Mista. 4. DIMACS FNC. 5. SAT-FHM. I. Ribas, Bruno César, orient. II. Título.

Resumo

A satisfatibilidade booleana é um problema que, dada uma expressão lógica, é decidido se existe algum conjunto de variáveis valoradas de modo a tornar a expressão verdadeira. Tal problema é de fundamental importância em áreas como ciência da computação, inteligência artificial e projetos de hardware. Embora satisfatibilidade booleana seja um problema NP-completo, um algoritmo de complexidade $O(2^{0.5284 \times N})$ foi definido para resolver especificamente a classe de fórmulas Horn mistas utilizando algoritmos de geração de conjuntos independentes máximos. O objetivo deste trabalho é construir um resolvedor de satisfatibilidade para fórmulas Horn mistas e comparar seu desempenho com resolvedores atuais. As fórmulas Horn mistas são um conjunto de fórmulas compostas de uma parte Horn e uma parte na forma normal conjuntiva contendo no máximo duas cláusulas.

Palavras-chaves: satisfatibilidade booleana, fórmula Horn mista, lógica proposicional, forma normal conjuntiva, DIMACS FNC, 2-SAT, SAT-FHM, conjunto independente máximo.

Abstract

Boolean satisfiability is a problem that, given a logical expression, it decides whether there is some set of variables valued in order to make the expression true. Such a problem is of fundamental importance in areas such as computer science, artificial intelligence and hardware projects. Although Boolean satisfiability is an NP-complete problem, an algorithm of complexity $O(2^{0.5284\times N})$ was defined to specifically solve the class of mixed Horn formulas using algorithms for generating maximum independent sets. The objective of this work is to build a satisfiability solver for mixed Horn formulas and compare its performance with current solvers. Mixed Horn formulas are a set of formulas composed of a Horn part and a conjunctive normal form part containing at most two clauses.

Keywords: boolean satisfiability, mixed Horn formula, propositional logic, conjunctive normal form, DIMACS CNF, 2-SAT, SAT-MHF, maximal independent set.

Lista de abreviaturas e siglas

UnB	Universidade de Brasília
FGA	Faculdade do Gama
FNC	Forma Normal Conjuntiva
HORNSAT	Satisfiabilidade de Horn
FHM	Fórmula Horn Mista
SAT	Satisfatibilidade Booleana
NP	Tempo Polinomial Não Determinístico
FNI	Forma Normal Implicativa
CFC	Componente Fortemente Conectado
TCC2	Trabalho de Conclusão de Curso 2
TCC1	Trabalho de Conclusão de Curso 1
CIM	Conjunto Independente Máximo
V	Verdade
F	Falsidade
DFS	Busca em Profundidade

Lista de ilustrações

Índice de algoritmos

Lista de tabelas

Sumário

1 Introdução

Segundo Cook (1971), o problema de Satisfatibilidade Booleana (SAT) é reconhecidamente um dos primeiros problemas NP-completo na teoria da complexidade computacional, sendo de fundamental importância em áreas como ciência da computação, inteligência artificial e projetos de hardware.

1.1 Contextualização

Apesar de SAT ser um problema NP-completo é possível resolvê-lo com uma complexidade exponencial um pouco menor para alguns subconjuntos de SAT. Porschen e Speckenmeyer (2007) definem um algoritmo para essa otimização mostrando como é possível alcancar a complexidade $O(2^{0.5284\times N})$ para Fórmulas Horn Mistas (FHM).

1.2 Objetivos

O objetivo desse trabalho é construir um resolvedor SAT de FHM e comparar seu desempenho com outros resolvedores modernos como o glucose (SIMON, 2016) baseado no MiniSat (EéN; SöRENSSON, 2005). Tal resolvedor terá como base os trabalhos de Porschen e Speckenmeyer (2007) e será construído utilizando a linguagem de programação C++.

1.3 Objetivos Específicos

- Implementar um resolvedor para FHM SAT;

1.4 Organização do Trabalho

Esse trabalho está dividido em quatro capítulos: Introdução, Referencial Teórico, Resolvedor FHM SAT e Conclusão. O segundo capítulo possui uma base teórica que será utilizada ao longo do trabalho. O capítulo três explica como funciona o resolvedor construído. Já no capítulo quatro é o resultado do trabalho.

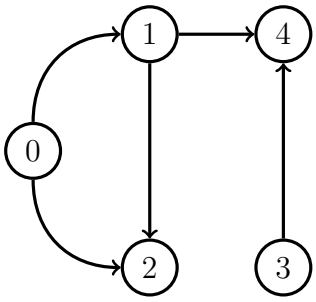
2 Referencial Teórico

A teoria desta seção será utilizada como ferramenta para a proposta do trabalho, mostrando os conhecimentos básicos para a compreensão do todo.

2.1 Grafos

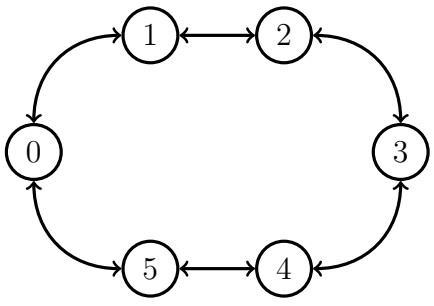
Segundo Sedgewick e Wayne (1983) um grafo $G(V, E)$ é um conjunto de vértices (ou nós) V e uma coleção de arestas E que conectam um par de vértices. Um grafo pode ser unidirecional ou bidirecional. O grafo unidirecional define um caminho que para cada aresta só existe o caminho de ida, já o grafo bidirecional existe sempre o caminho de ida e volta. A Figura 1 mostra um exemplo de um grafo unidirecional. A Figura 2 mostra um exemplo de um grafo bidirecional.

Figura 1 – Exemplo de um grafo unidirecional



Fonte: Autor (2022).

Figura 2 – Exemplo de um grafo bidirecional

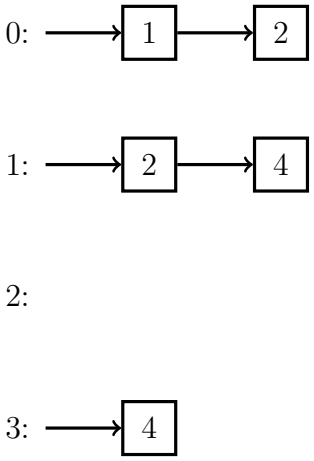


Fonte: Autor (2022).

2.1.1 Lista de Adjacências

Segundo Sedgewick e Wayne (1983) uma lista de adjacências consiste em um conjunto de listas indexadas por vértices dos nós adjacentes a cada vértice. Todo grafo pode ser representado por sua lista de adjacências. A Figura 3 mostra a lista de adjacências do grafo da Figura 1.

Figura 3 – Lista de adjacências do grafo da Figura 1



Fonte: Autor (2022).

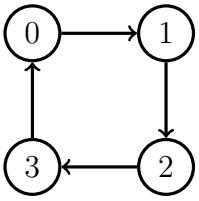
2.1.2 Busca em Profundidade

Segundo Sedgewick e Wayne (1983) a Busca em Profundidade (da sigla em inglês: *Depth-First Search*, DFS) é um algoritmo para realizar uma busca ou travessia em um grafo. O algoritmo consiste em escolher um nó do grafo e visitar seus nós vizinhos e os vizinhos dele, dando preferência sempre para os vizinhos do último nó visitado antes de visitar os demais, visitando todos os nós do grafo. No caso de uma busca o algoritmo se encerraria assim que encontrasse o nó que procura, no caso de uma travessia o algoritmo se encerra quando não houver mais nós.

2.1.3 Componente Fortemente Conectado

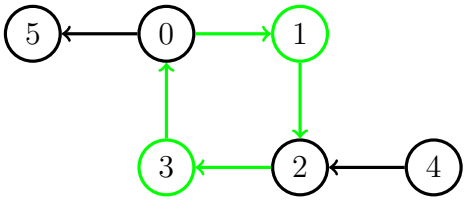
Segundo Sedgewick e Wayne (1983) dois vértices v e w são fortemente conectados se eles são mutualmente alcançados, ou seja, se existe um caminho de v até w e um caminho de w até v. Um grafo é fortemente conectado se todos seus vértices são fortemente conectados a todos os demais. A Figura 4 mostra um exemplo de um grafo fortemente conectado. A Figura 5 mostra um exemplo de dois vértices fortemente conectados.

Figura 4 – Exemplo de um grafo fortemente conectado



Fonte: Autor (2022).

Figura 5 – Exemplo de nós fortemente conectados em verde



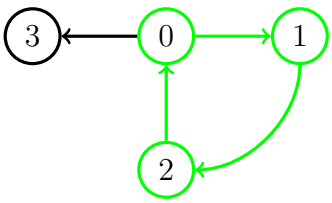
Fonte: Autor (2022).

Segundo Sedgewick e Wayne (1983) um Componente Fortemente Conectado (CFC) é um conjunto maximal de nós que segue três propriedades:

- Reflexivo: Todo vértice v é fortemente conectado a ele mesmo;
- Simétrico: Se um vértice v é fortemente conectado a um vértice w, então w é fortemente conectado a v;
- Transitivo: Se vértice um v é fortemente conectado a um vértice w e w é fortemente conectado a um vértice x, então v também é fortemente conectado a x.

A Figura 6 mostra um exemplo de um CFC.

Figura 6 – Exemplo de CFC em verde

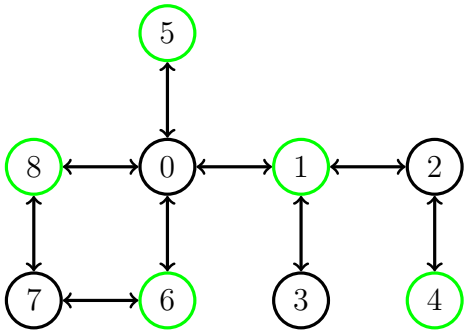


Fonte: Autor (2022).

2.1.4 Componente Independente Maximal

Segundo Johnson e Yannakakis (1987) um Componente Independente Máximo (CIM) é o conjunto máximo de nós V' pertencentes a um grafo $G(V, E)$ em que não existam dois nós em V' sendo ligados por uma aresta E e de modo que cada vértice em $V - V'$ seja ligado a um vértice em V'. A Figura 7 mostra um exemplo de um CIM.

Figura 7 – Exemplo de CIM representado pelos nós em verde

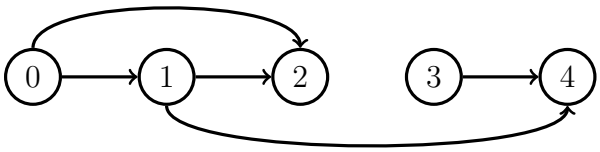


Fonte: Autor (2022).

2.1.5 Ordem Topológica

Segundo Sedgewick e Wayne (1983) a ordenação topológica de um grafo é a ordem linear em que cada nó vem antes de todos os nós para os quais este tenha arestas de saída. A Figura 8 mostra o grafo da Figura 1 visualmente em ordem topológica.

Figura 8 – Grafo da Figura 1 na ordem topológica



Fonte: Autor (2022).

2.2 Lógica Proposicional

A lógica proposicional é um sistema formal que representa proposições, atômicas ou compostas, possuindo valor Verdade (V) e Falsidade (F), mas nunca ambos. Uma proposição é uma sentença declarativa que se pode atribuir valor lógico ou verdade ou falsidade a ela.

Segundo Büning e Lettmann (1999), as operações sobre proposições, chamadas operações lógicas, obedecem a regras de um cálculo, denominado cálculo proposicional, semelhante ao da aritmética sobre números. As operações básicas da lógica proposicional são do tipo negação ($\neg$), conjunção ($\wedge$) e disjunção ($\vee$).

Dentro de lógica proposicional um literal (ou átomo) é uma variável proposicional ou sua negação (i.e., p, q, $\neg$p) e uma cláusula é uma disjunção de literais.

Segundo Büning e Lettmann (1999), as fórmulas bem formadas são indutivamente definidas por quatro regras:

- Todo átomo é uma fórmula;
- Se α é uma fórmula, então $(\neg\alpha)$ também é uma fórmula;
- Se α e β são fórmulas, então $(\alpha \vee \beta)$ e $(\alpha \wedge \beta)$ também são fórmulas;
- Apenas as expressões formadas pelas regras anteriores são fórmulas.

2.2.1 Tabela Verdade

Segundo Filho (2003), a tabela verdade figura todos os possíveis valores lógicos da proposição composta correspondentes a todas as possíveis atribuições de valores lógicos às proposições simples componentes. As Tabelas 1, 2, 3 mostram a tabela verdade das operações básicas disjunção, conjunção e negação.

Tabela 1 – Tabela verdade disjunção

p	**q**	**(p $\vee$ q)**
V	V	V
V	F	V
F	V	V
F	F	F

Fonte: Adaptado de Filho (2003).

Tabela 2 – Tabela verdade conjunção

p	**q**	**(p $\wedge$ q)**
V	V	V
V	F	F
F	V	F
F	F	F

Fonte: Adaptado de Filho (2003).

Tabela 3 – Tabela verdade negação

p	**¬ p**
V	F
F	V

Fonte: Adaptado de Filho (2003).

2.2.2 Tautologia

Segundo Filho (2003), tautologia é uma fórmula proposicional que é verdade independente da valoração de suas variáveis, em outras palavras, é impossível valorar uma tautologia de modo a obter falsidade. A Equação 2.1 mostra o exemplo mais básico de tautologia sendo visível em sua tabela verdade conforme a Tabela 4.

$$(q \vee \neg q) \tag{2.1}$$

Tabela 4 – Tabela verdade da Equação 2.1

q	**¬ q**	**(q ∨ ¬ q)**
V	F	V
F	V	V

Fonte: Autor (2022).

2.2.3 Forma Normal Conjuntiva

Segundo Biere et al. (2009), a Forma Normal Conjuntiva (FNC) é uma conjunção de cláusulas disjuntivas e pode ser vista como conjunção de disjunções. Em outras palavras, na lógica proposicional, uma fórmula está na sua FNC quando tal fórmula possui em sua composição apenas conjunções de cláusulas e cada uma é formada apenas por disjunções de literais. Toda e qualquer fórmula pode ser transformada em uma FNC equivalente. A Equação 2.2 representa uma fórmula FNC genérica. A Equação 2.3 representa um exemplo de FNC.

$$(A \vee B \vee \ldots \vee C) \wedge (D \vee E \vee \ldots \vee F) \wedge \ldots \wedge (X \vee Y \vee \ldots \vee Z) \tag{2.2}$$

$$(p \vee q \vee \neg r) \wedge (p \vee \neg q) \wedge (\neg q \vee r \vee s) \tag{2.3}$$

2.2.3.1 FNC monótona

Segundo Porschen e Speckenmeyer (2007), uma cláusula contendo apenas literais positivos (negativos) é chamada de monótona positiva (negativa). Uma FNC monótona é um caso especial de FNC com as mesmas propriedades e atributos.

A Equação 2.4 não é monótona pois existem literais não negados p, q e s e literais negados $\neg r$. A Equação 2.5 é monótona pois não existe nenhum literal negado entre os seus literais, sendo chamada de monótona positiva. Já a Equação 2.6 é monótona pois só existem literais negados, sendo chamada de monótona negativa.

$$(p \vee q) \wedge (p \vee \neg r \vee s) \wedge (s \vee q) \tag{2.4}$$

$$(p \vee q) \wedge (q \vee p \vee r \vee s) \wedge (q \vee r) \tag{2.5}$$

$$(\neg p \vee \neg q) \wedge (\neg q \vee \neg r) \wedge (\neg s \vee \neg p) \tag{2.6}$$

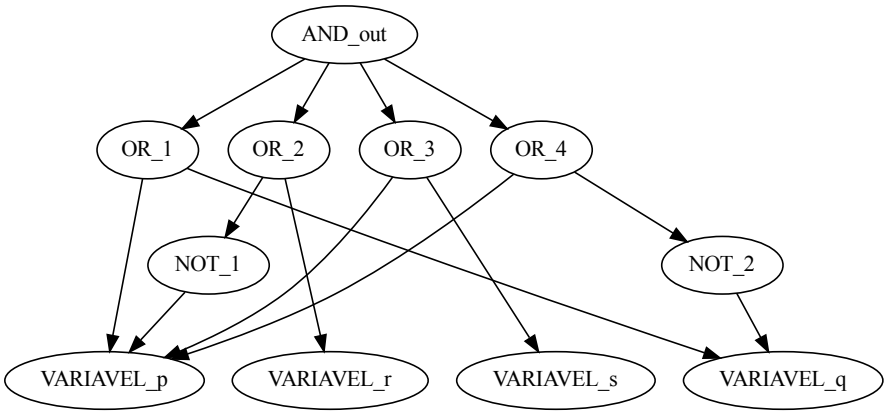
2.2.3.2 2-FNC

Segundo Aanderaa, Börger e Lewis (1982), uma 2-FNC, também conhecida como fórmula Krom, é um caso especial de FNC com as mesmas propriedades e atributos, porém com no máximo dois literais por cláusula. A Equação 2.7 representa uma 2-FNC genérica. A Equação 2.8 representa um exemplo de 2-FNC. A Figura 9 representa a forma em grafo da Equação 2.8 segundo Ribas (2011).

$$(A \vee B) \wedge (C \vee D) \wedge \ldots \wedge (Y \vee Z) \tag{2.7}$$

$$(p \vee q) \wedge (\neg p \vee r) \wedge (p \vee s) \wedge (p \vee \neg q) \tag{2.8}$$

Figura 9 – Representação em grafo da Equação 2.8



Fonte: Adaptado de Ribas (2011)

2.2.3.3 Fórmula Horn

A fórmula de Horn é uma fórmula na FNC que possui no máximo um literal positivo (cabeça da cláusula) e N literais negativos por cláusula (PORSCHEN; SPECKENMEYER, 2007). A Equação 2.9 representa uma de fórmula de Horn genérica. A Equação 2.10 representa um exemplo de uma fórmula de Horn.

$$(\neg A \vee \neg B \vee \neg C \vee ... \vee Z) \wedge ... \wedge (\neg D \vee \neg E \vee \neg F \vee ... \vee Y) \tag{2.9}$$

$$(\neg p \vee \neg q \vee r) \wedge (\neg p \vee \neg r \vee \neg s) \tag{2.10}$$

2.2.3.4 Fórmula Horn Mista

Segundo Porschen e Speckenmeyer (2007), uma FHM é composta por uma parte 2-FNC monótona positiva e uma parte Horn. Mais precisamente, para uma fórmula 2-FNC monótona positiva P e uma fórmula Horn H chamamos a conjunção de H e P de fórmula de Horn mista como mostrado na Equação 2.11.

$$FHM = H \wedge P \tag{2.11}$$

Onde:

H = parte de Horn
P = parte 2-FNC monótona positiva

2.2.3.5 DIMACS FNC

O propósito de utilizar um formato padrão é diminuir o esforço para testar e comparar algoritmos, oferecendo um melhor ambiente de testes para análises. Para tal fim, DIMACS (1993) define um formato que consiste em preâmbulos e cláusulas.

O preâmbulo é dividido em dois tipos: os comentários e a linha de problema. Os comentários sempre estarão no começo do arquivo e pode existir mais de um. A linha de problema é única e descreve o problema. A Figura 10 mostra o exemplo de um arquivo FNC completo.

- Preâmbulo: contém as informações iniciais do problema e é dividido em dois tipos:
  - Comentário: os comentários aparecem no começo do preâmbulo e começam com o caractere "c" e em seguida possuem uma linha contendo o comentário. Podem existir vários comentários. Ex:

```
c linha comentada.
```

  - Linha de Problema: começa com o caractere "p" e é seguido do formato, que precisa ser "cnf" (indicando FNC), quantidade de variáveis e quantidade de cláusulas separadas por um espaço em branco. Ex:

```
p cnf 339 2787
```

- Cláusulas: as cláusulas vêm logo após a linha de problema e podem estar no formato de Horn ou no formato 2-FNC. Cada variável da cláusula é escrita em um número, sendo sua versão negada o número oposto ou simétrico. O zero indica o final de uma cláusula. Podem existir várias cláusulas. Ex:

```
1 2 0
```

Figura 10 – Exemplo de arquivo DIMAS FNC completo

```
c linha comentada
c exemplo: (q or p) and (r or p)
c 3 variáveis (q, r, p)
c 2 cláusulas (q or p) e (r or p)
p cnf 3 2
1 2 0
3 2 0
```

Fonte: Autor (2022).

Um exemplo de leitura passo a passo do arquivo pode ser encontrada no anexo A.

2.3 Satisfatibilidade

SAT é um problema NP-completo que dado uma expressão lógica com variáveis booleanas, ele decide se existe alguma valoração de variáveis para tornar a expressão verdadeira. Logo, quando a formulação é dita como satisfazível existe um conjunto de variáveis valoradas que torna a expressão verdadeira e quando a formulação é dita como não satisfazível é impossível gerar um conjunto de variáveis valoradas que torna a expressão verdadeira (RIBAS, 2015).

2.3.1 Verificar Valoração de SAT

Para verificar a valoração de uma fórmula é possível aplicar o valor de cada literal na expressão e calcular o resultado. Para exemplificar a valoração será utilizada a Equação 2.12.

$$(p \vee q \vee \neg r) \wedge (\neg p \vee r) \wedge (p \vee q \vee \neg s) \tag{2.12}$$

Para o exemplo de verificação será utilizado dois conjuntos de valores possíveis $v1 = \{p : V, q : F, r : F, s : V\}$ e $v2 = \{p : V, q : F, r : V, s : V\}$. A Equação 2.13 mostra que o conjunto $v1$ não satisfaz a fórmula pois o resultado final é F. A Equação 2.14 mostra que o conjunto $v2$ satisfaz a fórmula pois o resultado final é V.

$$\begin{gathered}
(p \vee q \vee \neg r) \wedge (\neg p \vee r) \wedge (p \vee q \vee \neg s) \\
(V \vee F \vee \neg F) \wedge (\neg V \vee F) \wedge (V \vee F \vee \neg V) \\
(V \vee F \vee V) \wedge (F \vee F) \wedge (V \vee F \vee F) \\
(V \vee V) \wedge F \wedge (V \vee F) \\
V \wedge F \wedge V \\
F \wedge V \\
F
\end{gathered} \tag{2.13}$$

$$
\begin{gathered}
(p \vee q \vee \neg r) \wedge (\neg p \vee r) \wedge (p \vee q \vee \neg s) \\
(V \vee F \vee \neg V) \wedge (\neg V \vee V) \wedge (V \vee F \vee \neg V) \\
(V \vee F \vee F) \wedge (F \vee V) \wedge (V \vee F \vee F) \\
(V \vee F) \wedge V \wedge (V \vee F) \\
V \wedge V \wedge V \\
V \wedge V \\
V
\end{gathered}
\tag{2.14}
$$

2.3.2 2-SAT

O problema 2-SAT é uma restrição do SAT, tendo no máximo dois literais por cláusula, em outras palavras, 2-SAT é o problema de resolver SAT para 2-FNC.

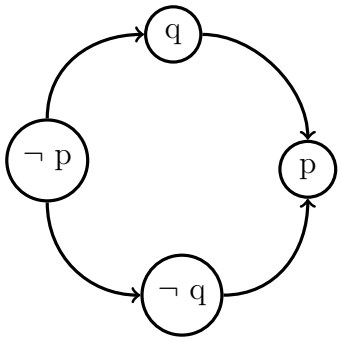
Existe um algoritmo definido em CPALGORITHMS (2017) para resolver 2-SAT. O primeiro passo do algoritmo é transformar a 2-FNC para uma forma diferente, a Forma Normal Implicativa (FNI), que consiste nas possíveis implicações lógicas da fórmula.

Em uma disjunção de dois literais existem duas implicações lógicas: a negação da primeira variável implicará logicamente na segunda e a negação da segunda variável implicará logicamente na primeira. Com a FNI é montado um grafo com as implicações. A Equação 2.15 representa um exemplo de 2-FNC.

$$
(p \vee q) \wedge (p \vee \neg q) \tag{2.15}
$$

A fórmula 2.15 é composta por uma 2-FNC de duas cláusulas. A primeira cláusula $(p \vee q)$ resulta nas implicações $(\neg p \Rightarrow q)$ e $(\neg q \Rightarrow p)$. A segunda cláusula $(p \vee \neg q)$ resulta nas implicações $(\neg p \Rightarrow \neg q)$ e $(q \Rightarrow p)$. Essas implicações estão representadas na Figura 11. O Algoritmo 1 implementa um pseudocódigo para transformar uma fórmula 2-FNC em um grafo de implicações.

Figura 11 – Grafo FNI da Equação 2.15



Fonte: Autor (2022).

Com o grafo de implicações construído o próximo passo é achar todos os CFC, que pode ser feito em $O(n + m)$ com o algoritmo de Kosaraju (SHARIR, 1981). Para cada variável P_i é verificado se ela se encontra no mesmo componente que seu complemento. Caso exista complementos no mesmo componente, pode-se dizer que eles se implicam logicamente conforme a Equação 2.16

$$\begin{gathered} P_i \Rightarrow \neg P_i \\ ou \\ \neg P_i \Rightarrow P_i \end{gathered} \tag{2.16}$$

Caso isso aconteça significa que existe uma contradição, pois tanto P_i quanto $\neg P_i$ precisariam ser verdadeiros conforme a Equação 2.17, então a FNC é não satisfazível.

$$P_i \equiv \neg P_i \equiv Verdade \tag{2.17}$$

Caso não exista nenhuma contradição é possível valorar as variáveis de modo a satisfazer a fórmula. Para a valoração os CFC são ordenados em ordem topológica e, feito isso, é atribuído verdade à variável se ela se encontrar em um componente de ordem maior que seu complemento, ou falso caso contrário. Outra solução possível é passar pelo grafo em ordem topológica reversa atribuindo verdade para cada variável não valorada, isso se dá pelo fato que em uma implicação lógica, caso a segunda fórmula seja verdade, a implicação resultará em verdade independentemente do valor da primeira fórmula, como mostrado na Tabela 5. O Algoritmo 2 implementa um pseudocódigo para resolver 2-SAT.

Tabela 5 – Tabela verdade de uma implicação lógica em tautologia

A	**V**	**(A ⇒ V)**
V	V	V
F	V	V

Fonte: Autor (2022).

Algoritmo 1: Pseudocódigo para transformar 2-FNC no grafo de implicações.

```
Data: fórmula 2-FNC
Result: grafo de implicações
grafoDeImplicações[ ][ ];
forall c : fórmula 2-FNC do
    grafoDeImplicações[complemento(c.primeiroAtomo)].add(c.segundoAtomo);
    grafoDeImplicações[c.segundoAtomo].add(complemento(c.primeiroAtomo));
end
return grafoDeImplicações;
```

Fonte: Autor (2022).

Algoritmo 2: Pseudocódigo para resolver 2-FNC.

```
Data: grafo de implicações, array de variáveis v[1...n]
Result: SAT|UNSAT sendo SAT satisfazível e UNSAT não satisfazível, array
        contendo uma possível valoração
resposta[ ];
componentes[ ] = kosaraju(grafo);
forall v : variáveis do
    if mesmoComponente(v, complemento(v)) then
        return UNSAT, null;
    end
    resposta[v] = componentePossuiOrdemTopológicaMaior(v, complemento(v));
end
return SAT, resposta;
```

Fonte: Adaptado (CPALGORITHMS, 2017).

Um exemplo de resolução passo a passo pode ser encontrada no anexo B.

2.3.3 Satisfatibilidade de Horn

Segundo Porschen e Speckenmeyer (2007) a Satisfatibilidade de Horn (HORNSAT) é um problema de decidir se uma fórmula Horn é satisfazível ou não.

É possível um resolver HORNSAT para uma fórmula de Horn f com n cláusulas com os seguintes passos.

1. Criar uma lista lf com todas as cláusulas de f.
2. Para cada cláusula atômica c_a, com um único literal a, atribua valor de modo a satisfazer a.
3. Remover todas as cláusulas que possuírem a de lf pois já possuem valor V.
4. Remover $\neg a$ das cláusulas restantes de lf.
5. Retornar ao passo 2 até não haver mais cláusulas atômicas ou alguma cláusula ficar sem literais.
6. Caso alguma cláusula fique sem literais, significa que foi encontrada uma contradição e f é UNSAT, caso contrário, f é SAT e os demais literais l podem ser valorados com F.

Tal solução se dá devido à natureza das fórmulas Horn, que possuem no máximo um literal positivo por cláusula. Em uma fórmula Horn sem cláusulas atômicas, é possível valorar todos os literais com F, pois é garantido que em cada cláusula exista um literal negado. Sendo assim, só é necessário forçar o valor de um literal caso ele se encontre em uma cláusula atômica ou se todos os demais literais de sua cláusula não satisfazerem a cláusula. O Algoritmo 3 implementa um pseudocódigo para resolver HORNSAT.

Algoritmo 3: Pseudocódigo para resolver HORNSAT.

```
Data: fórmula Horn H, array de variáveis v[1...n]
Result: SAT|UNSAT sendo SAT satisfazível e UNSAT não satisfazível, array
        contendo uma possível valoração
resposta[ ];
i = 1;
while i <= tamanho(H) do
    if tamanho(H[i]) == 1 then
        resposta[i] = H[i][0] > 0;
        removeCláusulasComLiteral(H, i);
        removeLiteralDeCláusulas(H, i*-1);
        i = 1;
    end
    else
        i++;
    end
end
forall c : H do
    if tamanho(c) == 0 then
        return UNSAT, null;
    end
end
forall l : v do
    if resposta[l] == null then
        resposta[l] = False;
    end
end
return SAT, resposta;
```

Fonte: Autor (2022).

Um exemplo de resolução passo a passo pode ser encontrada no Anexo C.

3 Resolvedor SAT-FHM

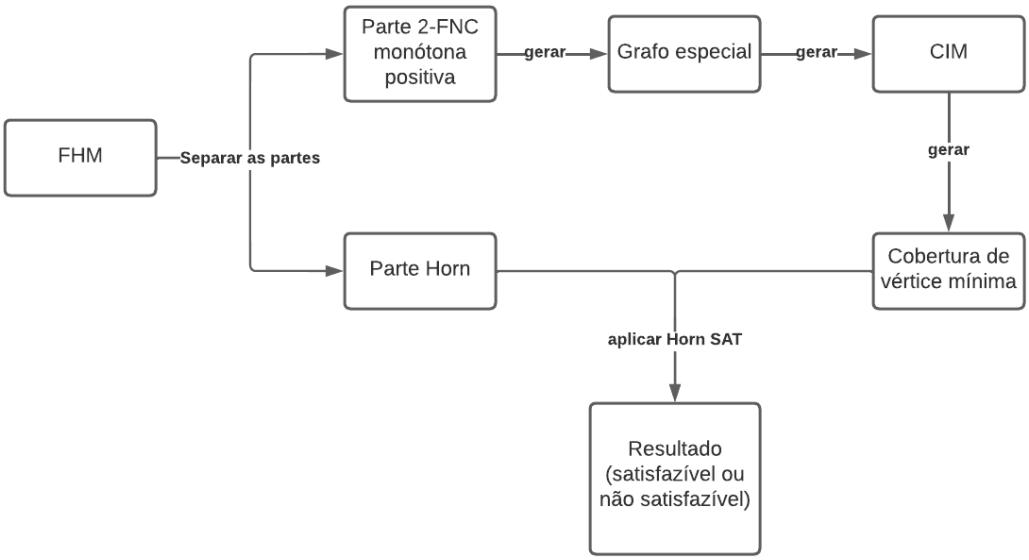
Segundo Porschen e Speckenmeyer (2007) dada uma fórmula $M = H \wedge P \in FHM$ sendo H a parte Horn e P a parte 2-FNC monótona positiva, é possível resolver SAT em $O(2^{0.5284 \times N})$. A Equação 3.1 mostra um exemplo de uma FHM. A Figura 12 mostra o fluxo do algoritmo.

$$M = (p \vee q) \wedge (\neg p \vee \neg r \vee \neg s) \wedge (p \vee r) \wedge (\neg q \vee \neg s \vee \neg u \vee \neg v) \tag{3.1}$$

Onde:

$$\begin{aligned}(\neg p \vee \neg r \vee \neg s) \wedge (\neg q \vee \neg s \vee \neg u \vee \neg v) &= \text{parte de Horn (polinomial)} \\ (p \vee q) \wedge (p \vee r) &= \text{parte 2-FNC monótona positiva (linear)}\end{aligned}$$

Figura 12 – Fluxo do resolvedor FHM SAT



Fonte: Autor (2022).

O primeiro passo para resolver SAT para FHM é separar as duas partes, a parte 2-FNC monótona positiva P e a parte Horn H. Com as duas partes separadas geramos todos os Conjuntos Independentes Máximos (CIM) que satisfazem P, deixando os demais literais livres, podendo ser atribuído qualquer valor a eles.

Com todos os CIM que satisfazem P é gerado a cobertura de vértice mínima, complemento do CIM, e testado Horn-SAT para cada conjunto extensível a H. Os literais

livres de P podem ser valorados de modo a satisfazer H e, consequentemente, satisfazer toda a instância M. O Algoritmo 4 implementa um pseudocódigo para resolver SAT-FHM.

Algoritmo 4: Pseudocódigo para resolver SAT-FHM.

```
Data: fórmula FHM
Result: SAT|UNSAT sendo SAT satisfazível e UNSAT não satisfazível, array
        contendo uma possível valoração
P = P(FHM);
if P == null then
    return HornSat(FHM)
end
vP = null;
vH = null;
cim = conjuntosIndependentesMaximos(P);
resposta = UNSAT;
while !cim.empty() do
    vP = minVertexCorver(cim.next());
    vH = HornSat(FHM(vP));
    if vH != null then
        resposta = SAT;
        break;
    end
end
return resposta, vP U vH;
```

Fonte: Adaptado (PORSCHEN; SPECKENMEYER, 2007).

Um exemplo de resolução passo a passo pode ser encontrada no anexo F.

3.1 Gerar Todos os Conjuntos Independentes Máximos em Ordem Lexicográfica da Parte 2-FNC

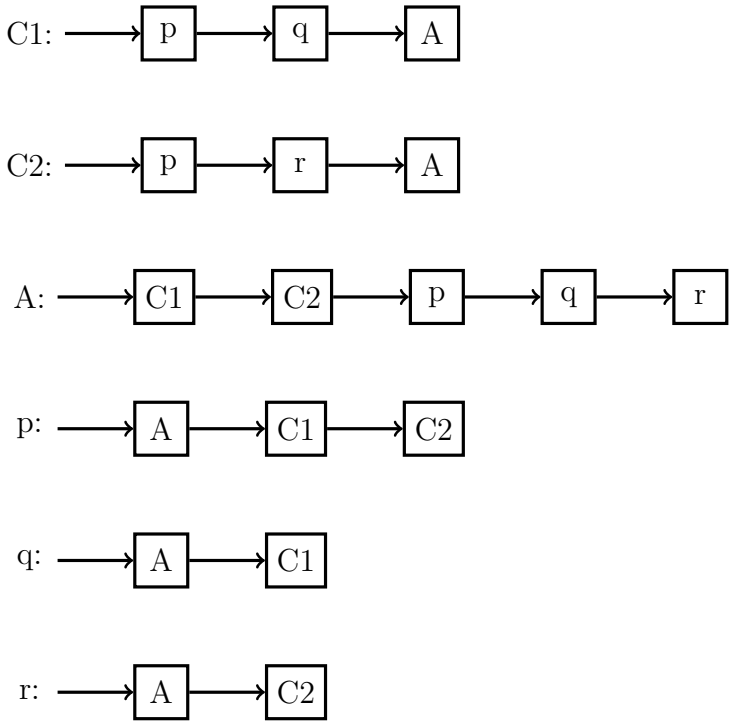
A solução é adaptada de Johnson e Yannakakis (1987). Para encontrar todos os CIM em ordem lexicográfica é necessário que o algoritmo receba um grafo. Porschen e Speckenmeyer (2007) utiliza um grafo cujo os vértices são os literais da fórmula, e cada vértice é ligado com outro caso os literais estejam na mesma cláusula. Para demonstrar como a geração é feita foi utilizado o grafo que o autor chama de grafo especial. Essa é parte de maior impacto na complexidade assintótica do algoritmo e usa exponencialmente a memória.

Para a solução será utilizada apenas a parte 2-FNC obtida através da FHM. Esse grafo especial é montado com uma ordem específica: os primeiros nós representam as cláusulas C_i, em seguida um nó A que o autor chama de nó especial e em sequência nós representando os literais x_j ou $\neg x_j$.

As adjacências do grafo se dão da seguinte forma: as cláusulas chegam nos literais pertencentes a elas, o nó especial chega em todos os nós e os literais chegam em seus opostos. A Figura 13 mostra a lista de adjacência da Equação 3.2 obtida através da parte 2-FNC da Equação 3.1. O Algoritmo 5 implementa um pseudocódigo para gerar os CIM. O anexo D mostra um exemplo passo a passo da montagem do grafo.

$$(p \vee q) \wedge (p \vee r) \tag{3.2}$$

Figura 13 – Lista de adjacências obtidas através da Equação 3.2



Fonte: Autor (2022).

Outro elemento que é necessário para o funcionamento do algoritmo é o primeiro conjunto independente máximo do grafo e ele pode ser obtido com os nós das cláusulas, visto que todas as cláusulas vão chegar em suas variáveis e que o nó especial é visto por todos os outros.

Com o grafo montado e seu primeiro conjunto independente máximo é possível começar o algoritmo. A ideia principal do algoritmo para gerar a ordem lexicográfica se

dá a partir de uma fila de prioridade dos conjuntos independentes máximos que respeite a ordem lexicográfica e pelo fato de que cada conjunto só gera conjuntos posteriores a ele.

Enquanto a fila de prioridades não estiver vazia, o próximo conjunto c_i em seu topo é dado como resposta e verificado se existe algum vértice v_g do grafo que possua adjacência a algum vértice v_p de c_i de ordem menor. Caso exista, é construído um conjunto cn_i cujos elementos são os n primeiros valores (em ordem lexicográfica) do conjunto, sendo n o grau de v_g, com exceção das adjacências de v_g e incluindo v_g, $cn_i = c_i \cap \{1, ..., n\} - adjacência(i) \cup i$. Caso tal conjunto seja um conjunto independente máximo (chegando em todas as cláusulas), adicionamos ele à nossa fila de prioridade. O Algoritmo 6 implementa um pseudocódigo para gerar todos os CIM em ordem lexicográfica.

Algoritmo 5: Pseudocódigo para gerar o grafo especial

```
Data: Array de cláusulas 2-FNC
Result: Grafo especial
grafo[ ][ ];
APos = clausulas.size()+1;
grafo.resize(APos);
for i = 0; i < APos; i++ do
    grafo[i].push_back(APos);
    grafo[APos].push_back(i);
end
forall c : clausulas do
    for l : c do
        if grafo[l] == null then
            grafo.push_back(l);
            grafo[l].push_back(APos);
            grafo[APos].push_back(l);
            if grafo[complemento(l) != null then
                grafo[l].push_back(complemento(l));
                grafo[complemento(l)].push_back(l);
            end
        end
        grafo[l].push_back(c);
        grafo[c].push_back(l);
    end
end
```

Fonte: Autor (2022).

Algoritmo 6: Pseudocódigo para gerar todos os conjuntos independentes máximos

```
Data: grafo especial, primeiro conjunto independente máximo do grafo
Result: array dos conjuntos independentes máximos em ordem lexigrográfica
resposta[ ];
filaDePrioridade.insert(primeiroConjuntoIndependenteMáximo);
while !filaDePrioridade.empty() do
    próximoConjunto = filaDePrioridade.retiraDoTopo();
    resposta.insert(próximoConjunto);
    forall vértice v_g do grafoEspecial adjacente a um vértice v_p do
     próximoConjunto onde v_p < v_g do
        próximoConjunto_n = primeirosNValores(próximoConjunto, v_g) -
        adjacência(v_g) ∪ v_g;
        if éConjuntoIndependenteMáximo(próximoConjunto_n) then
            filaDePrioridade.insert(próximoConjunto_n);
        end
    end
end
return resposta;
```

Fonte: Adaptado de (JOHNSON; YANNAKAKIS, 1987).

Um exemplo de geração passo a passo pode ser encontrada no anexo E.

4 Conclusão

Neste trabalho foi construído um Solver de FHM (2022) segundo os modelos de Porschen e Speckenmeyer (2007) e Johnson e Yannakakis (1987), que pode ser encontrado no link <https://github.com/UnB-SAT/tcc-andre-mhf-solver>, em um repositório do GitHub.

O algoritmo construído durante o trabalho não possui todas as simplificações e otimizações possíveis, tendo sido focado em entender e explicar os algoritmos utilizados.

Como resultado do trabalho foi obtido um estudo em grafos, lógica proposicional, satisfatibilidade e sobre os algoritmos utilizados, junto com uma explicação detalhada de cada algoritmo, abrindo possibilidade de evoluções futuras.

Uma proposta interessante para trabalhos futuros seria continuar no algoritmo descrito por Porschen e Speckenmeyer (2007) e implementar o FHM-SAT* descrito pelo autor, que adiciona o algoritmo descrito por Robson (2022) antes de aplicar o FHM-SAT para diminuir as fórmulas e realizar testes para verificar o impacto no desempenho.

Referências

AANDERAA, S.; BöRGER, E.; LEWIS, H. Conservative reduction classes of krom formulas. 1982. Disponível em: <https://doi.org/10.2307/2273385>. Citado na página 18.

BIERE, A. et al. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. [S.l.]: IOS Press, 2009. Citado na página 17.

BüNING, H.; LETTMANN, T. *Propositional Logic: Deduction and Algorithms*. [S.l.]: Cambridge University Press, 1999. Citado na página 16.

COOK, S. The complexity of theorem-proving procedures. 1971. Disponível em: <https://doi.org/10.1145/800157.805047>. Citado na página 11.

CPALGORITHMS. *2-SAT*. 2017. Disponível em: <https://cp-algorithms.com/graph/2SAT.html>. Acesso em: 05/08/2021. Citado 2 vezes nas páginas 22 e 24.

DIMACS. Satisfiability suggested format. 1993. Disponível em: <http://beyondnp.org/static/media/uploads/docs/satformat.pdf>. Acesso em: 15/08/2021. Citado na página 20.

EéN, N.; SöRENSSON, N. *MiniSat*. 2005. Disponível em: <http://minisat.se>. Acesso em: 21/10/2021. Citado na página 11.

FILHO, E. *Iniciação à lógica matemática*. [S.l.]: Nobel, 2003. Citado 2 vezes nas páginas 16 e 17.

JOHNSON, D.; YANNAKAKIS, M. On generating all maximal independent sets. 1987. Disponível em: <https://doi.org/10.1016/0020-0190(88)90065-8>. Acesso em: 02/09/2021. Citado 4 vezes nas páginas 15, 27, 30 e 31.

PINTO, A.; RIBAS, B. *TCC ANDRÉ P. MHF SOLVER*. 2022. Disponível em: <https://github.com/UnB-SAT/tcc-andre-mhf-solver>. Acesso em: 11/04/2022. Citado na página 31.

PORSCHEN, S.; SPECKENMEYER, E. Satisfiability of mixed horn formulas. 2007. Disponível em: <https://doi.org/10.1016/j.dam.2007.02.010>. Acesso em: 28/07/2021. Citado 7 vezes nas páginas 11, 18, 19, 24, 26, 27 e 31.

RIBAS, B. Satisfatibilidade não-clausal restrita às variáveis de entrada. 2011. Citado 2 vezes nas páginas 18 e 19.

RIBAS, B. Um método de pré-processamento de fórmulas sat e pseudo-boolean baseado em técnicas de programação linear inteira mista. 2015. Disponível em: <http://dx.doi.org/10.13140/RG.2.1.3852.9046>. Acesso em: 13/08/2021. Citado na página 21.

ROBSON, J. M. *Finding a maximum independent set in time* $O(2^{\frac{n}{4}})$. 2022. Disponível em: <https://www.labri.u-bordeaux.fr/perso/robson/mis/techrep.html>. Acesso em: 28/04/2022. Citado na página 31.

SEDGEWICK, R.; WAYNE, K. *Algorithms Fourth Edition*. [S.l.]: Princeton University, 1983. Citado 4 vezes nas páginas 12, 13, 14 e 15.

SHARIR, M. A strong-connectivity algorithm and its applications in data flow analysis. 1981. Disponível em: <https://doi.org/10.1016/0898-1221(81)90008-0>. Acesso em: 17/10/2021. Citado na página 23.

SIMON, L. *Glucose*. 2016. Disponível em: <https://www.labri.fr/perso/lsimon/glucose/>. Acesso em: 21/10/2021. Citado na página 11.

Anexos

ANEXO A – Exemplo Leitura de Arquivo FNC

A fórmula abaixo será utilizada para exemplificar a leitura.

$$(p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee \neg q \vee \neg r)$$

Que possuiria o seguinte formato DIMACS CNF (descrito na Seção 2.2.3.5):

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1 -2 -3 0
```

E seria lido da seguinte forma:

Realizar uma leitura por caractere caso, não seja um “p”, ignore a linha e leia novamente até encontrá-lo.

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1 -2 -3 0
```

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1 -2 -3 0
```

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1 -2 -3 0
```

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1 -2 -3 0
```

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1 -2 -3 0
```

Realizar a leitura do tipo, quantidade de variáveis e quantidade de cláusulas.

tipo = cnf

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1 -2 -3 0
```

quantidade de variáveis = 3

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1 -2 -3 0
```

quantidade de cláusulas = 4

```
c comment1
c comment2
p cnf 3  4
1  2  0
1  −2  0
−1  −3  0
−1  −2  −3  0
```

Realizar a leitura da quantidade de linhas necessárias (número de cláusulas) até encontrar zeros (indicando fim de cláusula) e separar as cláusulas Horn e as cláusulas 2-FNC em listas distintas.

```
c comment1
c comment2
p cnf 3 4
1  2  0
1  −2  0
−1  −3  0
−1  −2  −3  0
```

```
c comment1
c comment2
p cnf 3 4
1  2  0
1  −2  0
−1  −3  0
−1  −2  −3  0
```

Cláusulas 2-FNC = {{1, 2}}

```
c comment1
c comment2
p cnf 3 4
1 2  0
1  −2  0
−1  −3  0
−1  −2  −3  0
```

```
c comment1
c comment2
p cnf 3 4
1 2 0
1  −2  0
−1  −3  0
−1  −2  −3  0
```

```
c comment1
c comment2
p cnf 3 4
1 2 0
1  -2  0
−1  −3  0
−1  −2  −3  0
```

Cláusulas 2-FNC = {{1, 2}, {1, -2}}

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2  0
−1  −3  0
−1  −2  −3  0
```

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1  −3  0
−1  −2  −3  0
```

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1  -3  0
−1  −2  −3  0
```

Cláusulas 2-FNC = {{1, 2}, {1, -2}, {-1, -3}}

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3  0
−1  −2  −3  0
```

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1  −2  −3  0
```

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1  -2  −3  0
```

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1, -2  -3  0
```

Cláusulas Horn = {{-1, -2, -3}}

```
c comment1
c comment2
p cnf 3 4
1 2 0
1 -2 0
-1 -3 0
-1 -2, -3  0
```

Gerando o seguinte resultado:

Cláusulas 2-FNC = {{1, 2}, {1, -2}, {-1, -3}}

Cláusulas Horn = {{-1, -2, -3}}

ANEXO B – Exemplo Resolução 2-SAT

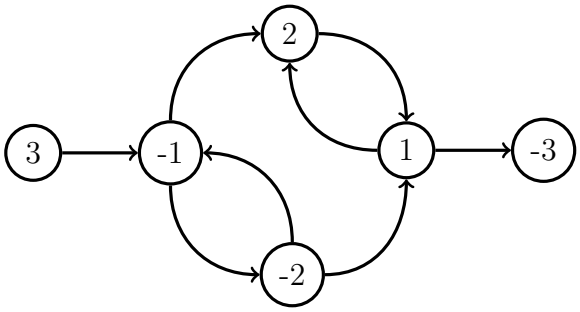
A fórmula abaixo será utilizada para exemplificar a resolução.

$$(p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee q)$$

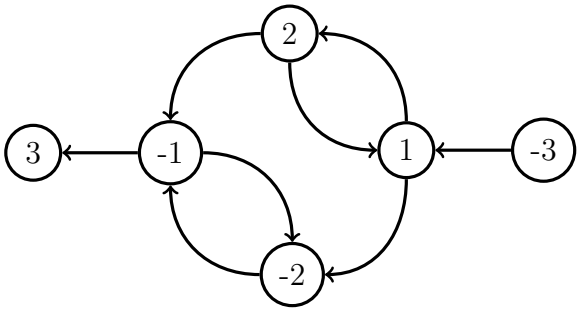
Que pode ser separada em quatro cláusulas com suas implicações:

- $(p \vee q)$: $\neg p \Rightarrow q$ e $\neg q \Rightarrow p$;
- $(p \vee \neg q)$: $\neg p \Rightarrow \neg q$ e $q \Rightarrow p$;
- $(\neg p \vee \neg r)$: $p \Rightarrow \neg r$ e $r \Rightarrow \neg p$;
- $(\neg p \vee q)$: $p \Rightarrow q$ e $\neg q \Rightarrow \neg p$.

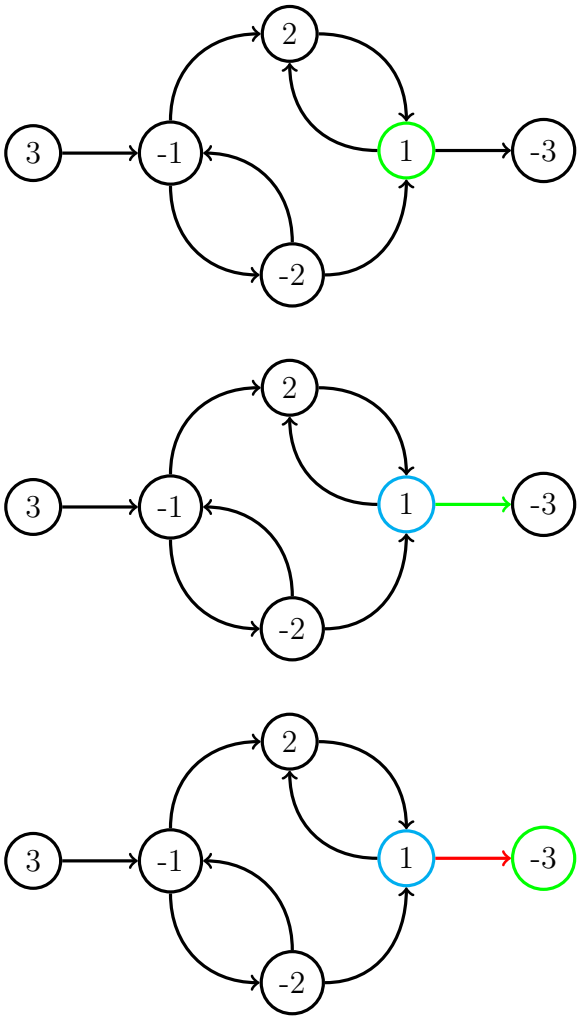
Gerando o seguinte grafo de implicações:



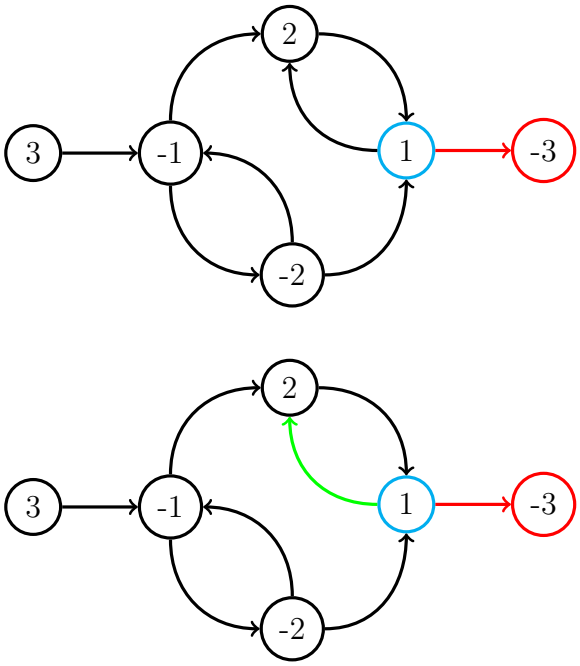
Com o seu grafo inverso:

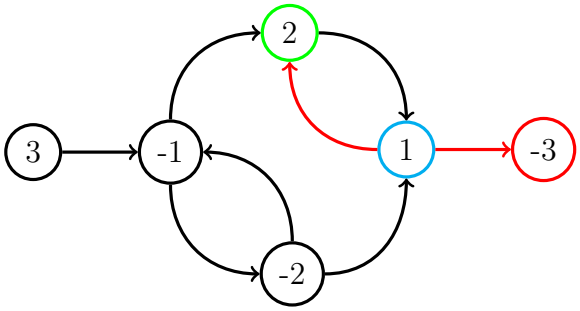


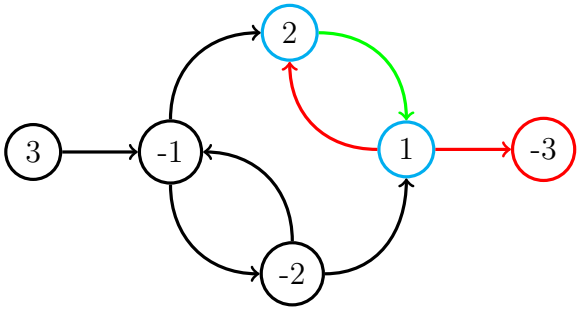
O algoritmo está descrito na Seção 2.3.2. O primeiro passo é fazer uma busca por profundidade no grafo guardando o término da busca de cada elemento. Nessa demonstração a cor verde representa o passo atual, a cor vermelha representa os passos já feitos, os nós em azuis são nós que não tiveram seu passo finalizado e as arestas em amarelo representam arestas não utilizadas pois o algoritmo já as utilizou antes.

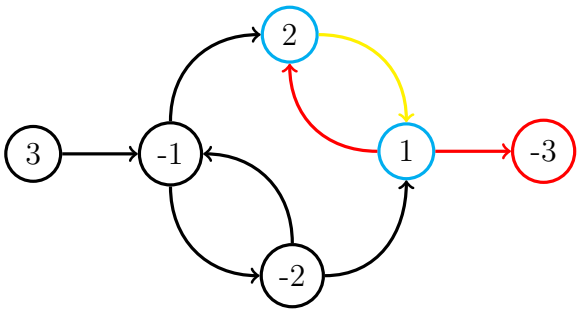


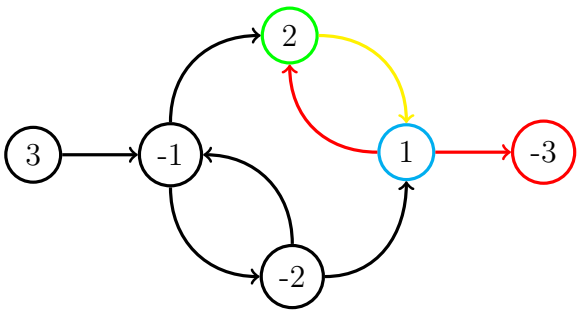
Ordem = {-3}



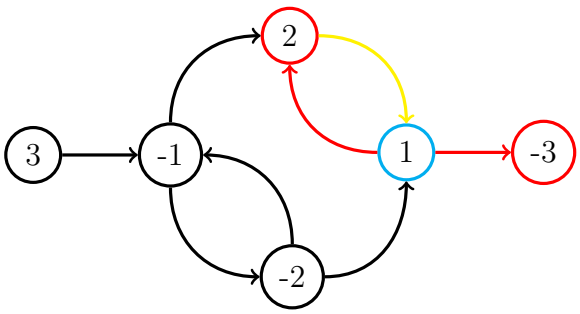


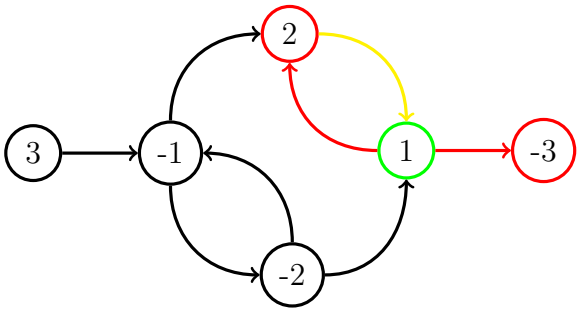




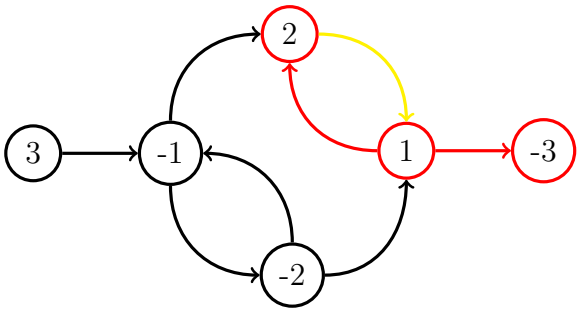


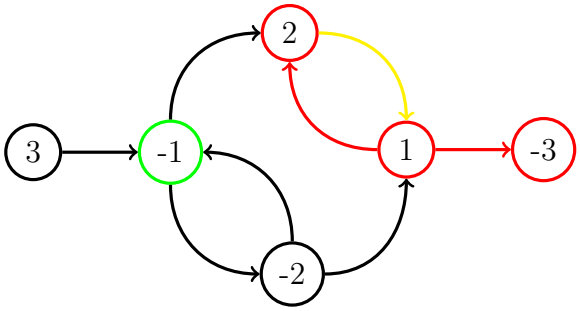
Ordem = {-3, 2}

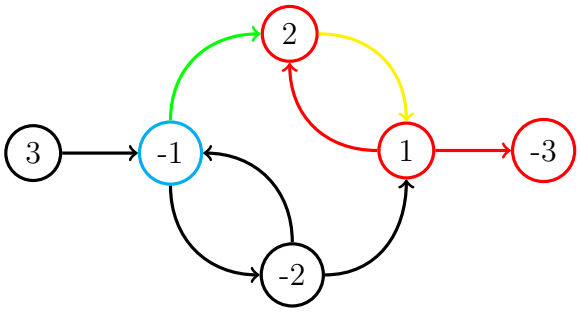


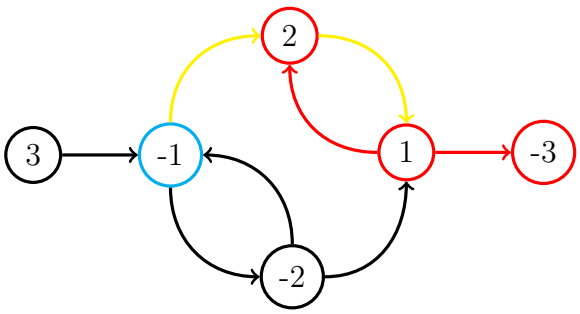


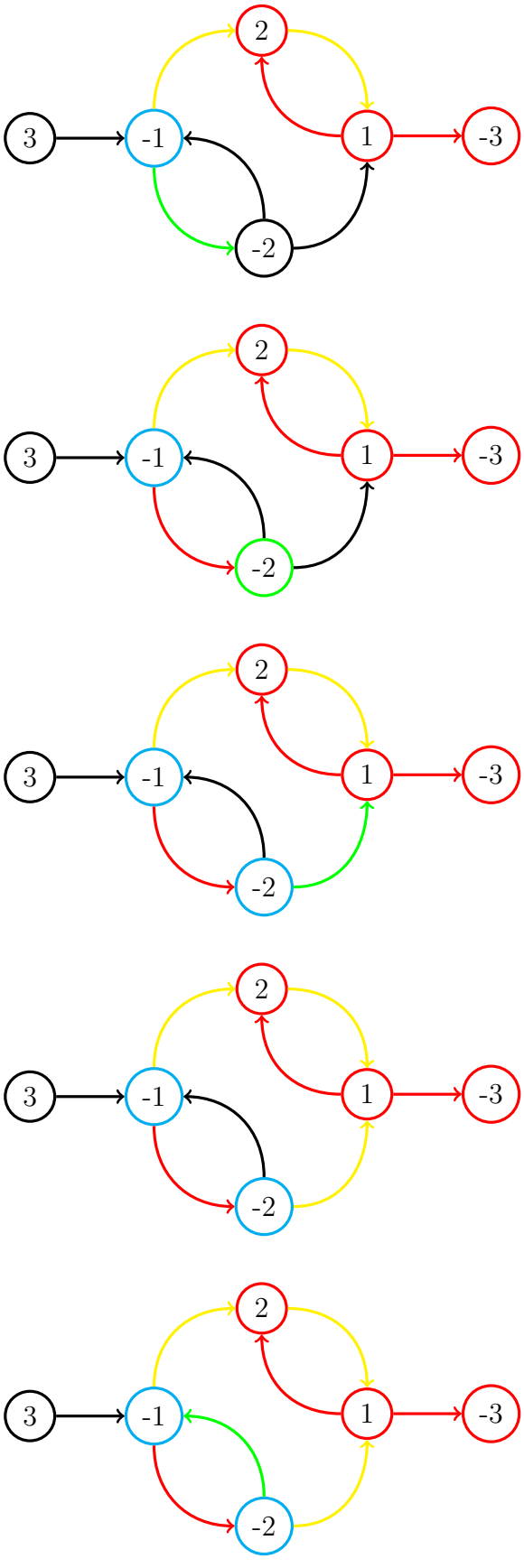
Ordem = {-3, 2, 1}

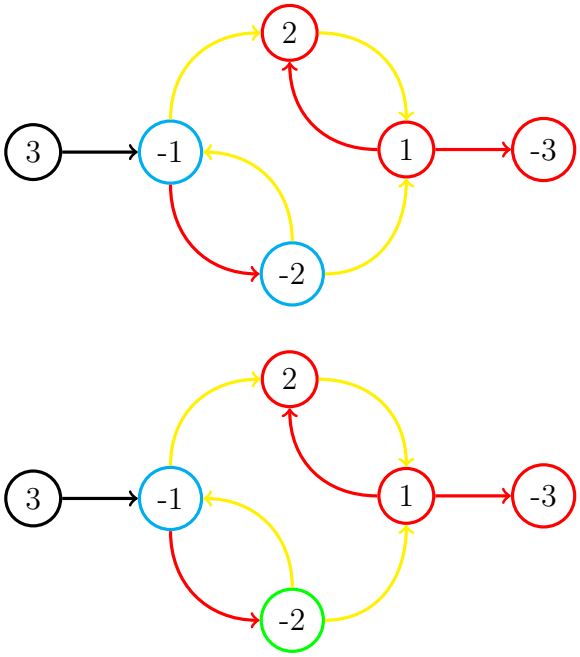




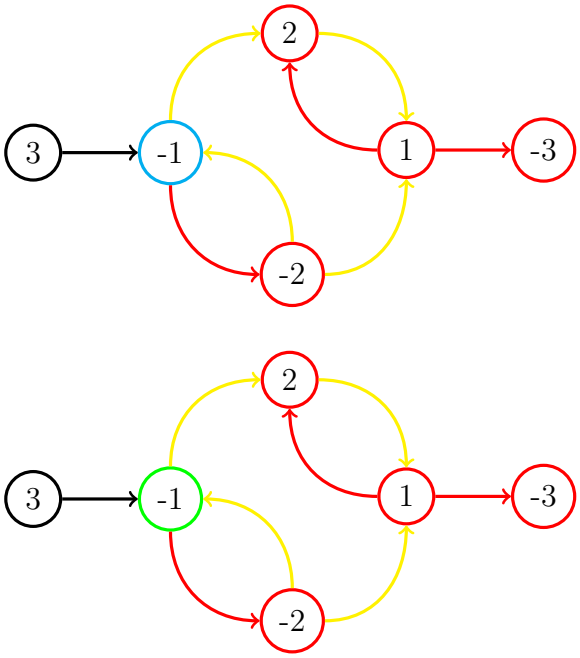




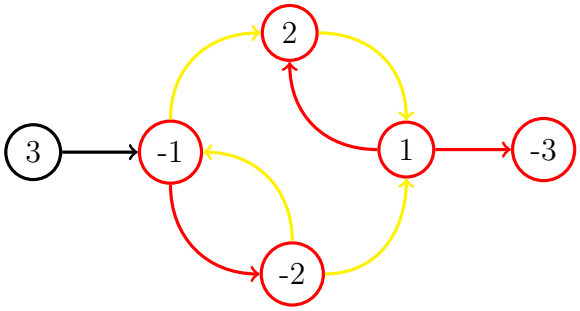

2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2

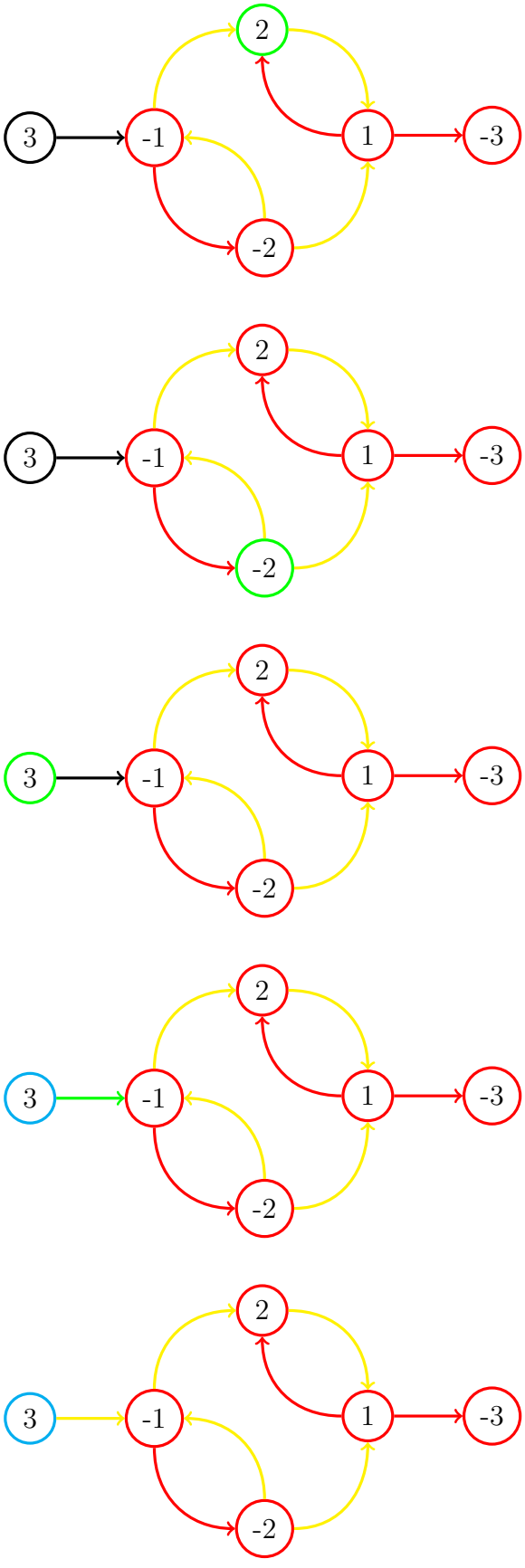


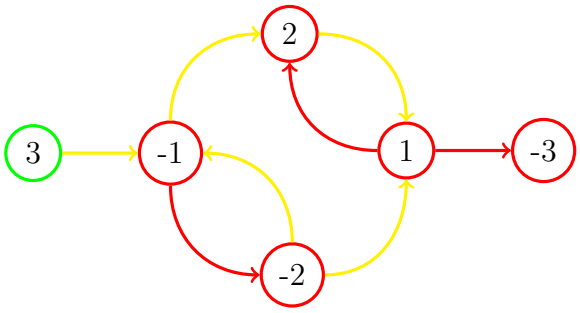
Ordem = {-3, 2, 1, -2}



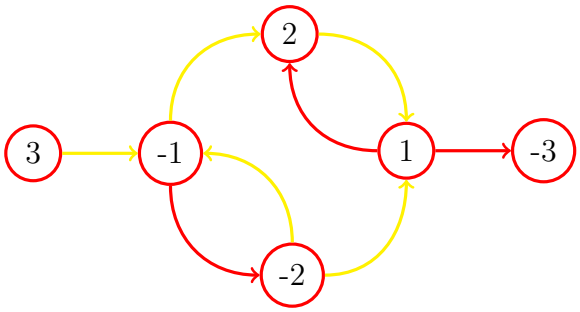
Ordem = {-3, 2, 1, -2, -1}

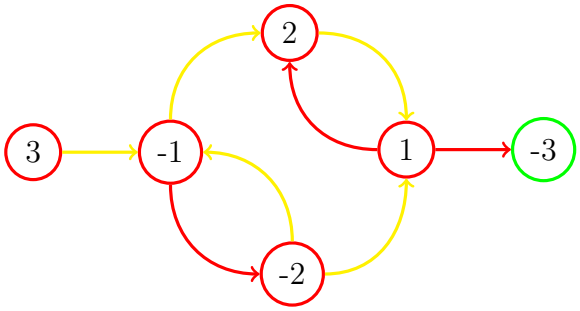


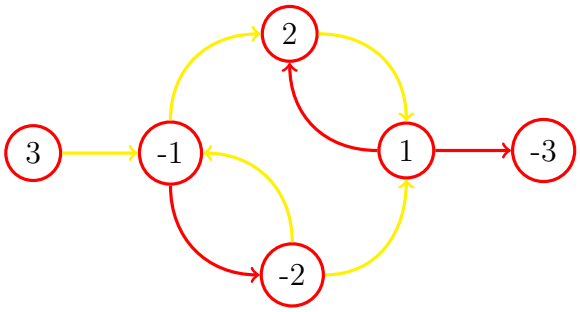

2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2



Ordem = {-3, 2, 1, -2, -1, 3}



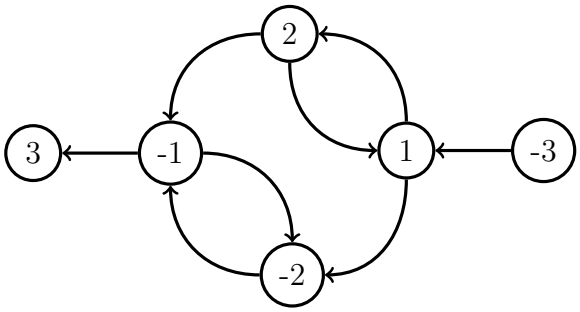


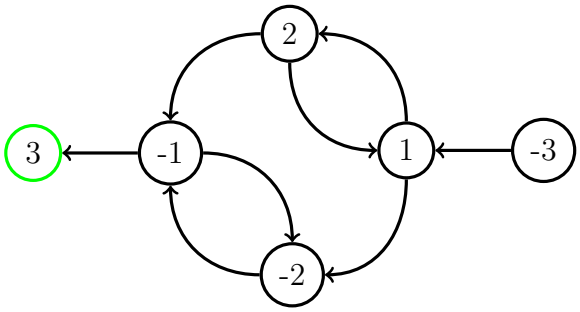


Após isso, é feito uma busca em profundidade no grafo inverso começando na ordem inversa da ordem encontrada no passo anterior e a partir de todo nó ainda não visitado, todos os nós não visitados que ele enxergar são componentes fortemente conectados.

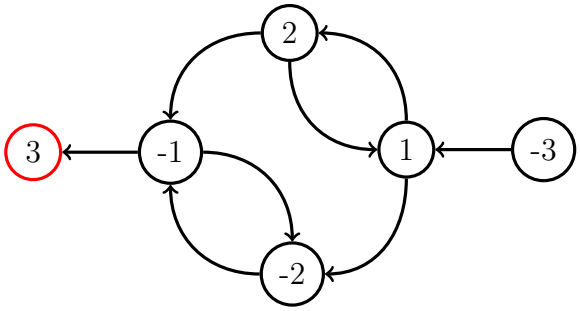
Ordem = {-3, 2, 1, -2, -1, 3}

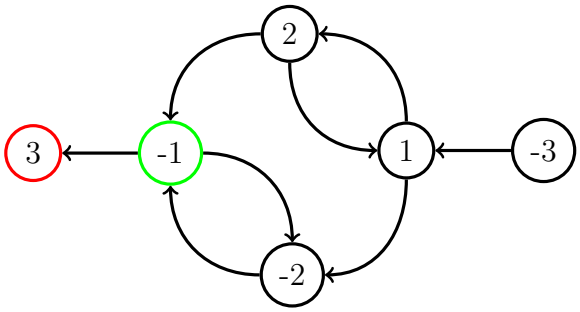
A ordem dos nós que vamos olhar será: 3, -1, -2, 1, 2, -3.

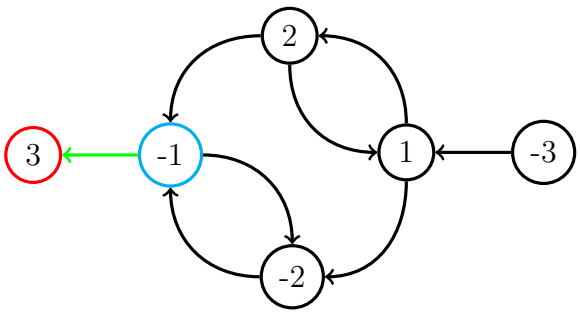


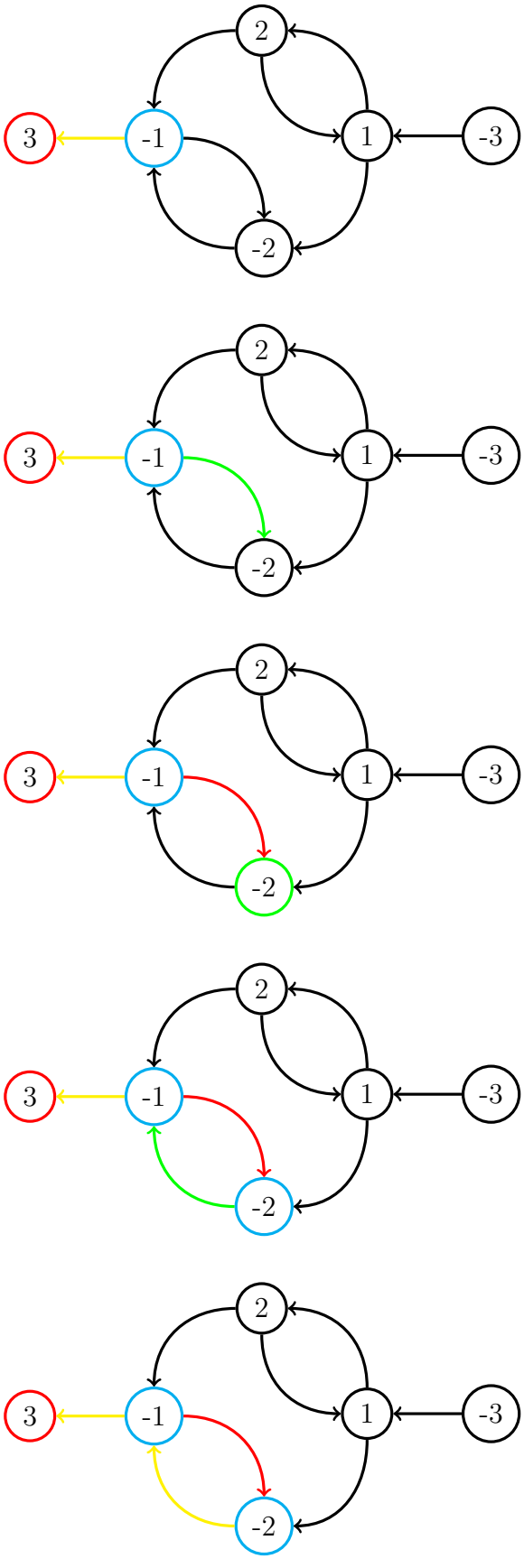


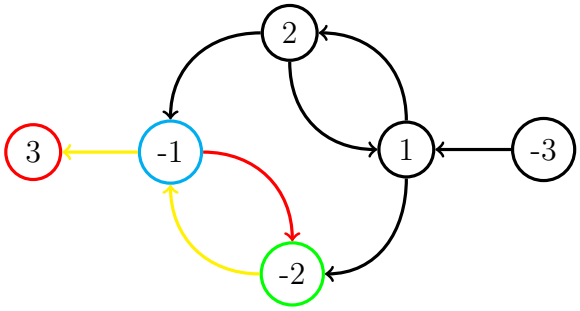
CFC = $\{C_1\{3\}\}$

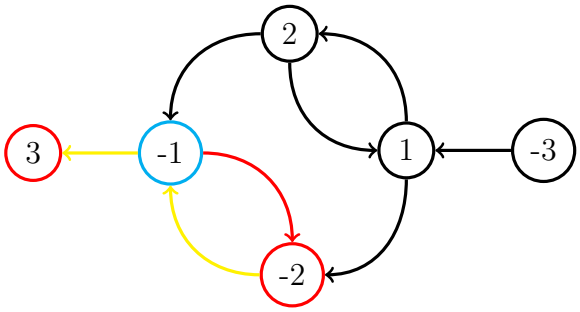


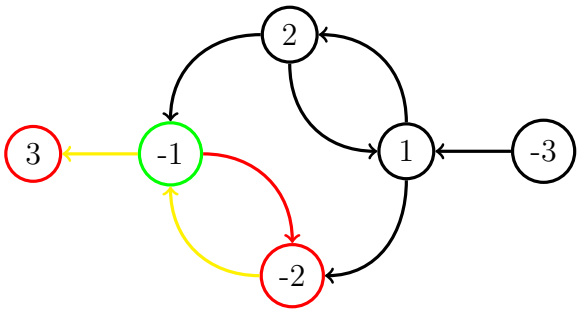




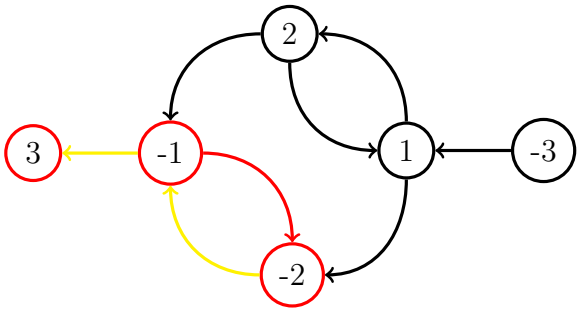

2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2

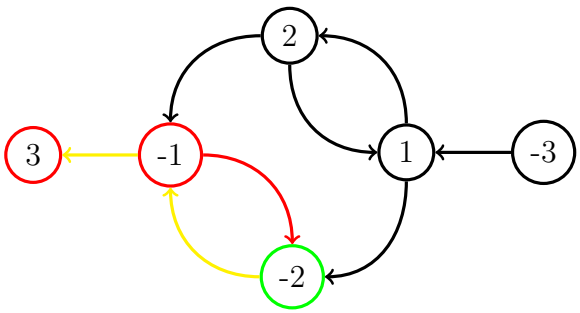


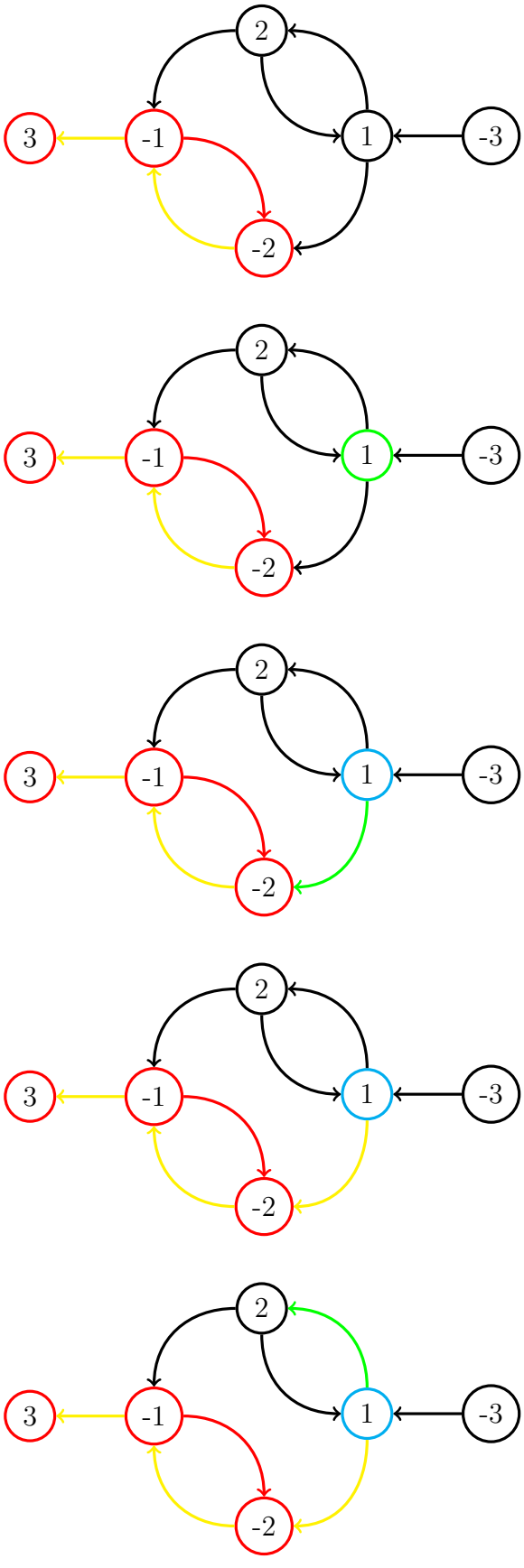


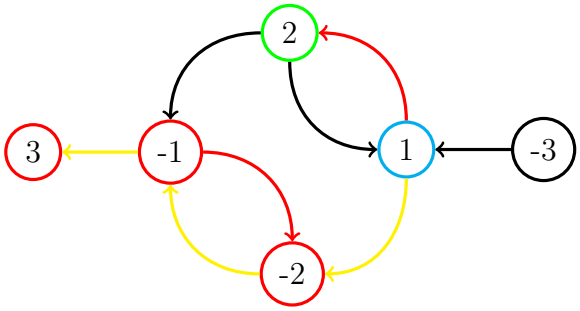


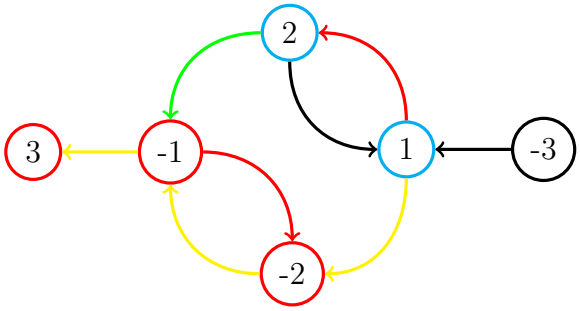
$$\text{CFC} = \{C_1\{3\}, C_2\{-1, -2\}\}$$

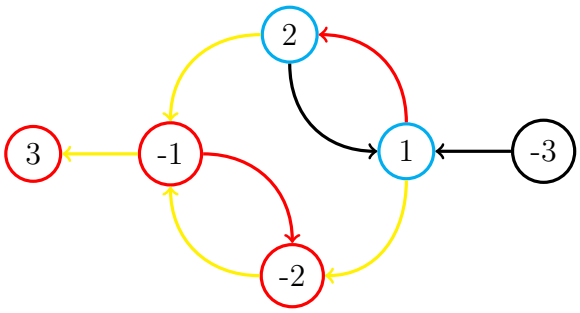


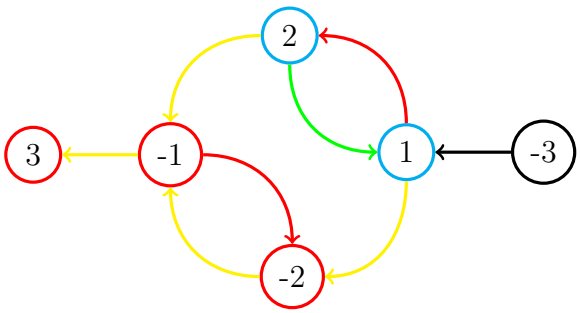


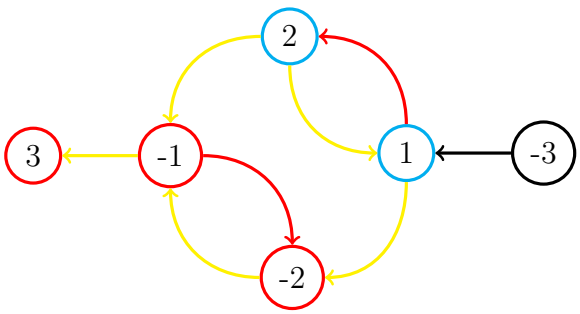

2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2
2
3
-1
1
-3
-2

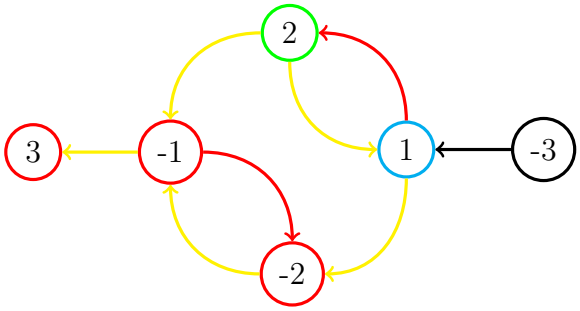

2
3
-1
1
-3
-2

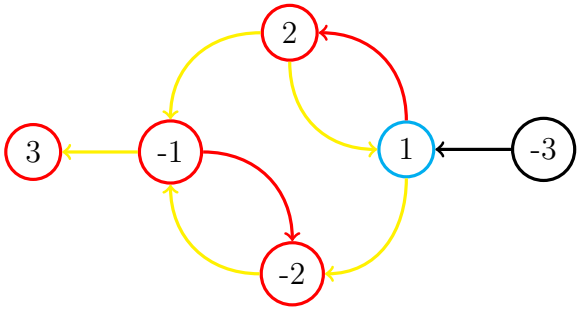

2
3
-1
1
-3
-2

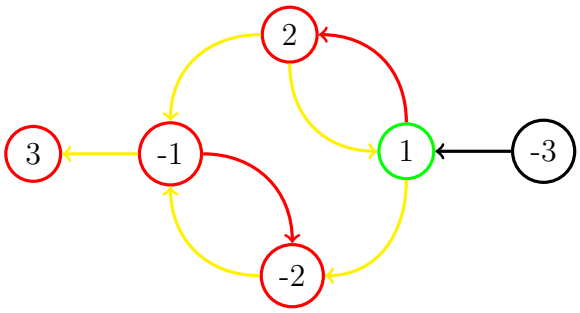

2
3
-1
1
-3
-2


2
3
-1
1
-3
-2

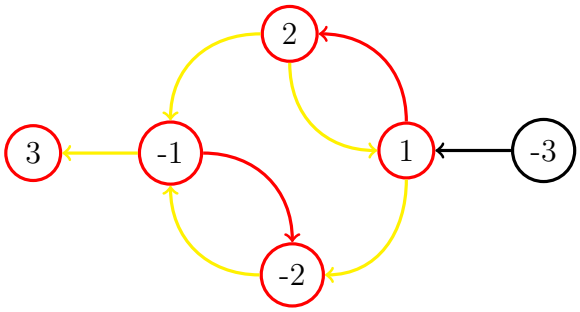

2
3
-1
1
-3
-2

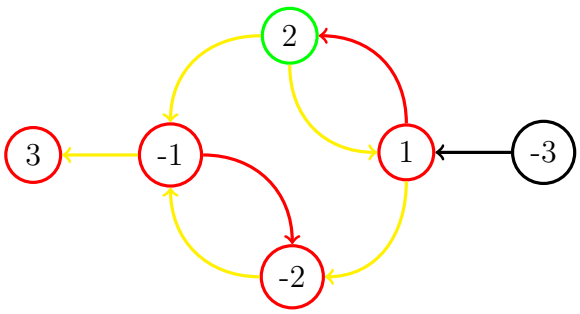


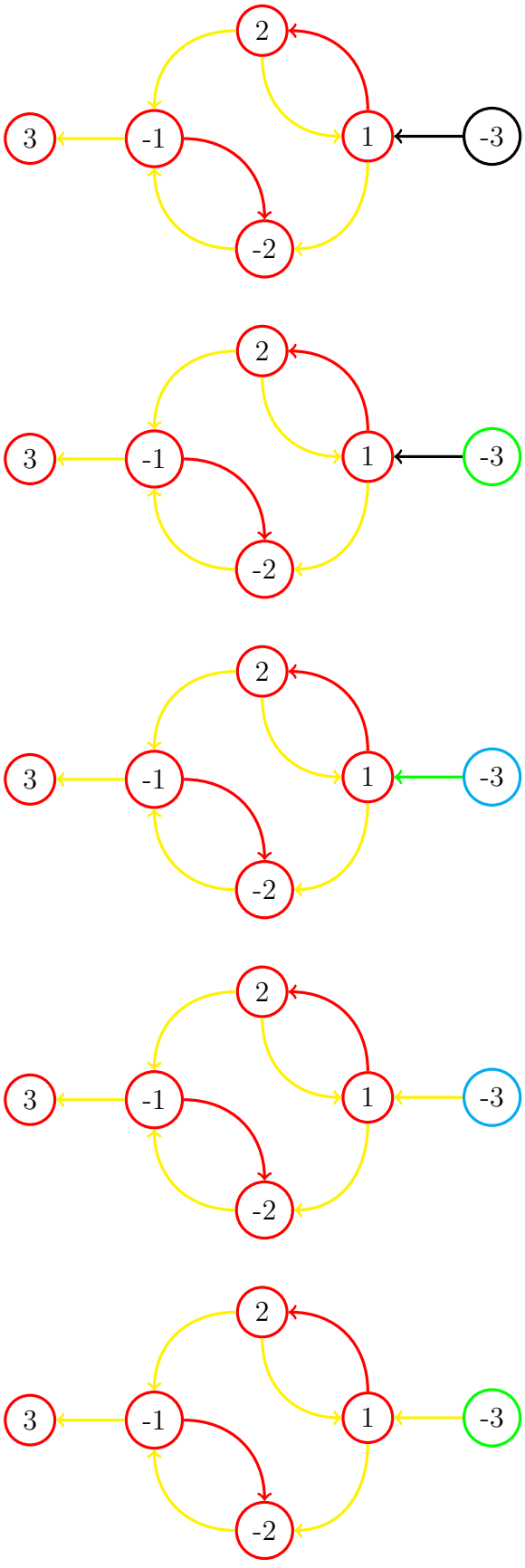




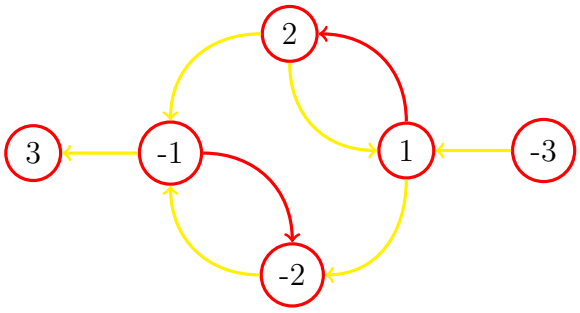
$$\text{CFC} = \{C_1\{3\}, C_2\{-1, -2\}, C_3\{1, 2\}\}$$





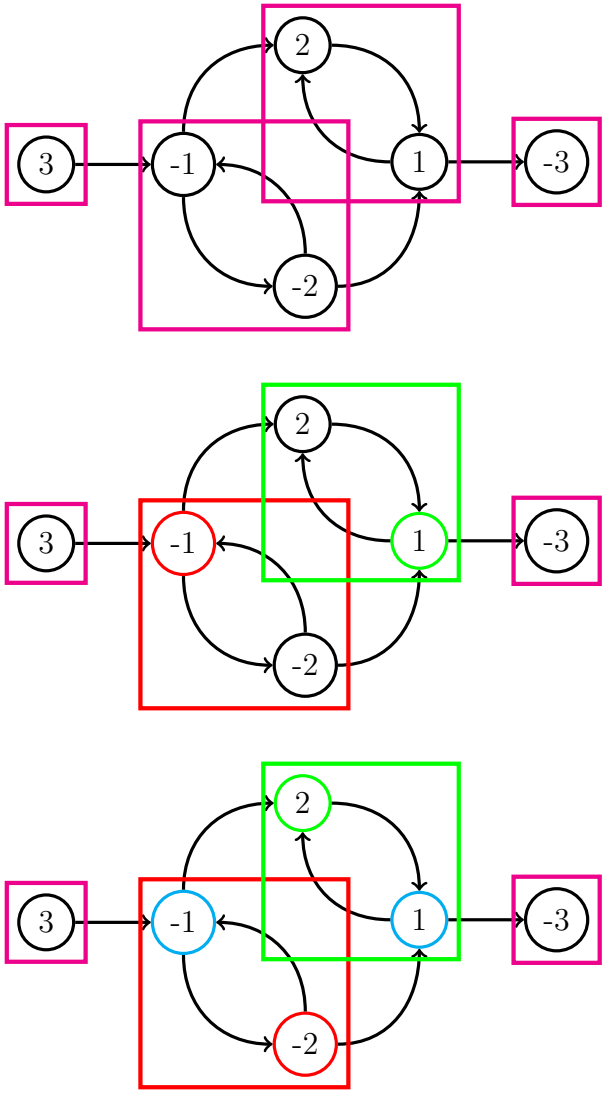


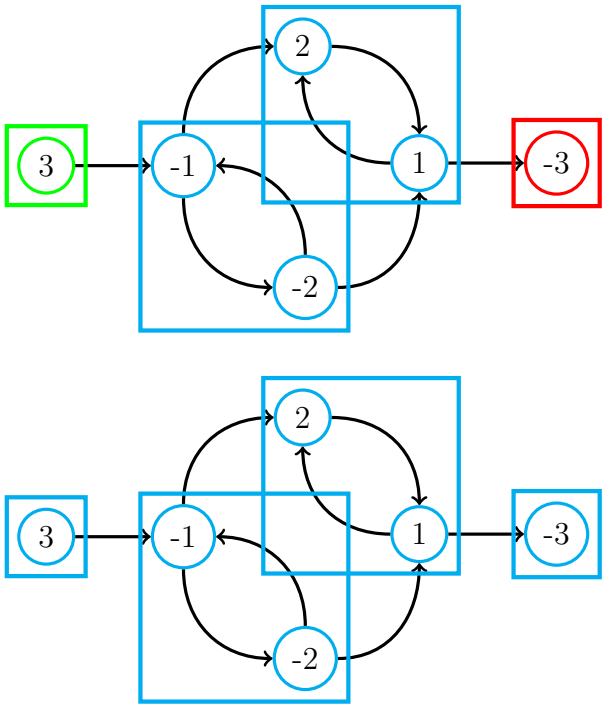
$$\text{CFC} = \{C_1\{3\}, C_2\{-1, -2\}, C_3\{1, 2\}, C_4\{-3\}\}$$



Com os CFC calculados, é verificado, para cada variável, se ela está no mesmo CFC que sua versão negada. Caso isso ocorra, é encontrada uma contradição e a FNC é não SAT.

$$\text{CFC} = \{C_1\{3\}, C_2\{-1, -2\}, C_3\{1, 2\}, C_4\{-3\}\}$$





Caso não existam contradições, é atribuído o valor da variável como verdade se ela estiver em um componente de maior grau que o componente de sua versão negada. Caso contrário, atribuímos falsidade.

$$C_1\{3\}, C_2\{-1, -2\}, C_3\{1, 2\}, C_4\{-3\}$$

$$1 = \text{V}$$

$$C_1\{3\}, C_2\{-1, -2\}, C_3\{1, 2\}, C_4\{-3\}$$

$$2 = \text{V}$$

$$C_1\{3\}, C_2\{-1, -2\}, C_3\{1, 2\}, C_4\{-3\}$$

$$3 = \text{F}$$

$$C_1\{3\}, C_2\{-1, -2\}, C_3\{1, 2\}, C_4\{-3\}$$

Para fins didáticos é possível testar o resultado:

$$(p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee q)$$
$$(V \vee V) \wedge (V \vee \neg V) \wedge (\neg V \vee \neg F) \wedge (\neg V \vee V)$$
$$(V \vee V) \wedge (V \vee F) \wedge (F \vee V) \wedge (F \vee V)$$
$$(V) \wedge (V) \wedge (V) \wedge (V)$$
$$(V) \wedge (V) \wedge (V)$$
$$(V) \wedge (V)$$
$$(V)$$

ANEXO C – Exemplo Resolução HORNSAT

A fórmula abaixo será utilizada para exemplificar a resolução.

$$(p) \wedge (\neg q) \wedge (\neg p \vee \neg q \vee \neg r) \wedge (\neg r \vee \neg s \vee t)$$

O primeiro passo para resolver HORNSAT é criar uma lista com todas as cláusulas, sendo assim, temos:

- (p)
- $(\neg q)$
- $(\neg p \vee \neg q \vee \neg r)$
- $(\neg r \vee \neg s \vee t)$

Em seguida é necessário olhar cara cláusula com apenas um literal e forçar o valor dele a ser V, removendo o literal oposto dele das demais cláusulas, pois sempre serão F. Em sequência deve-se tirar da lista as cláusulas já satisfeitas (que possuírem o literal forçado).

para (p):

- (p) -> $p = \text{V}$
- $(\neg q)$
- $(\neg p \vee \neg q \vee \neg r)$
- $(\neg r \vee \neg s \vee t)$

- $(\neg q)$
- $(\neg q \vee \neg r)$
- $(\neg r \vee \neg s \vee t)$

para $(\neg q)$:

- $(\neg q)$ -> $\neg q = \text{V}$ -> $q = \text{F}$
- $(\neg q \vee \neg r)$

- $(\neg r \vee \neg s \vee t)$

Sobrando:

- $(\neg r \vee \neg s \vee t)$

Ao remover os literais das cláusulas é possível gerar novas cláusulas com apenas um literal, portanto, sendo assim é necessário voltar no passo anterior enquanto houver novas cláusulas com apenas um literal.

Caso aconteça de ao remover um literal a cláusula ficar vazia significa que todos seus literais foram valorados com F, tornando a fórmula UNSAT. Caso contrário a fórmula é SAT.

Os demais literais que não foram valorados devem ser atribuídos o valor F a eles. No exemplo foi valorado p e q sobrando r, s e t, logo:

- $r = \mathrm{F}$
- $s = \mathrm{F}$
- $t = \mathrm{F}$

Para fins didáticos é possível testar o resultado:

$$(p) \wedge (\neg q) \wedge (\neg p \vee \neg q \vee \neg r) \wedge (\neg r \vee \neg s \vee t)$$
$$(V) \wedge (\neg F) \wedge (\neg V \vee \neg F \vee \neg F) \wedge (\neg F \vee \neg F \vee F)$$
$$(V) \wedge (V) \wedge (F \vee V \vee V) \wedge (V \vee V \vee F)$$
$$(V) \wedge (V) \wedge (V \vee V) \wedge (V \vee F)$$
$$(V) \wedge (V) \wedge (V) \wedge (V)$$
$$(V) \wedge (V) \wedge (V)$$
$$(V) \wedge (V)$$
$$(V)$$

ANEXO D – Exemplo Construção do Grafo Especial para Encontrar os CIM

A fórmula abaixo será utilizada para exemplificar a construção do grafo.

$$(p \vee q) \wedge (p \vee r)$$

O algoritmo está descrito na Seção 3.1. O primeiro passo é montar a estrutura do grafo na ordem definida. Os primeiros nós são referentes as cláusulas C_i, em seguida um nó especial A, seguido de nós representando os literais x_j ou $\neg x_j$. No exemplo existem duas cláusulas $C1 = (p \wedge q)$ e $C1 = (p \wedge r)$.

C1:

C2:

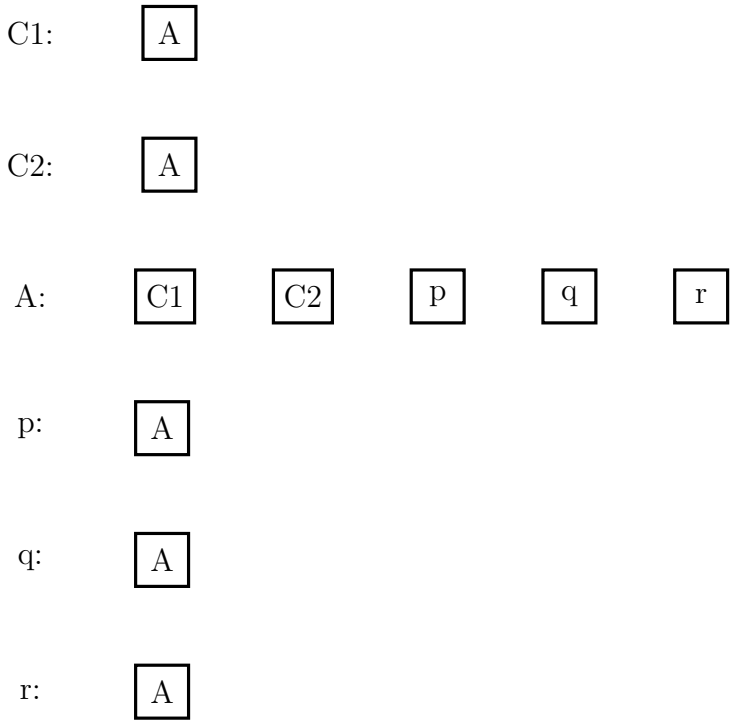
A:

p:

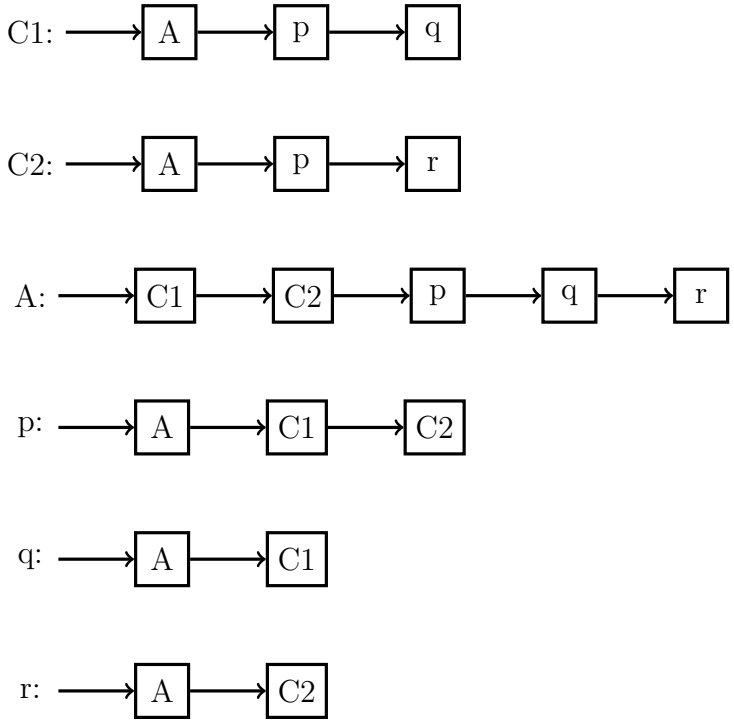
q:

r:

Em sequência vamos resolver as vizinhanças de A. Todo nó chega em A, então basta adicionar A à vizinhança dos demais nós e adicionar os demais nós à vizinhança de A, visto que se trata de um grafo bidirecional.



O próximo passo é adicionar a vizinha das cláusulas. Toda cláusula chega em seus literais, então basta adicionar os literais x_j à vizinhança de sua respectiva cláusula C_i, e vice e versa.



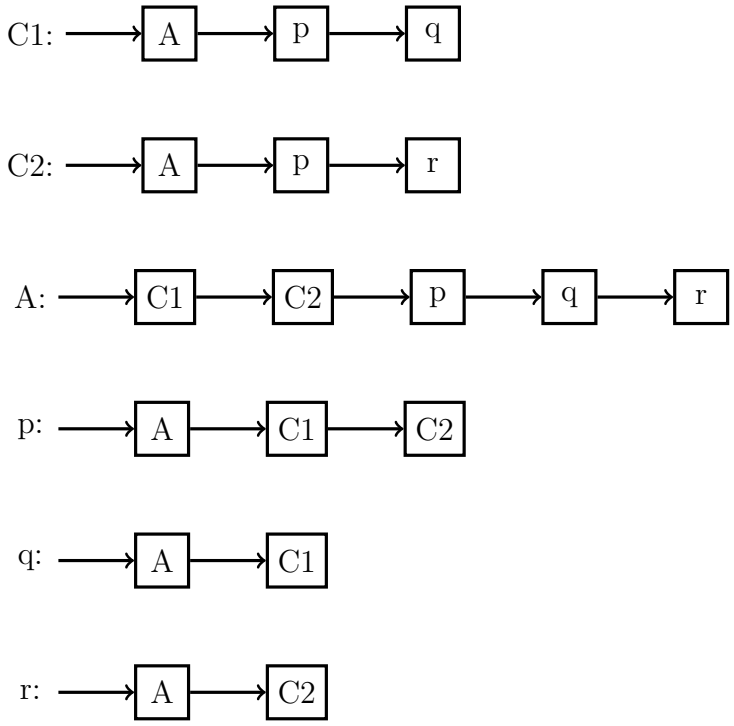
O último passo do algoritmo é adicionar a vizinha entre os literais e suas negações, como a Equação 3.2 se trata de um monótono positivo e não possui nenhum literal negado o algoritmo se encerra sem fazer alterações no grafo.

ANEXO E – Exemplo de Geração de todos os CIM em ordem lexicográfica

A fórmula abaixo será utilizada para exemplificar a geração de todos os CIM em ordem lexicográfica.

$$(p \vee q) \wedge (p \vee r)$$

Que teria o seguinte grafo especial g:



E o primeiro conjunto independente máximo seria $C = \{C1, C2\}$.

Fila de prioridades = {C1, C2}.

Para c_i = {C1, C2}:

É escolhido os nós de g adjacentes e com ordem maior do que algum elemento de c_i ({C2, A, p, q, r} são de ordem maior que C1) e calculado $cn_i = c_i \cap \{1, ..., n\} - adjacência(c_i) \cup c_i$ de cada um deles, caso cn_i chegue em todas as cláusulas é adicionado na fila de prioridades.

C_{C1} = ordem menor a todos os elementos.

C_{C2} = C1 não é adjacente e C2 é de ordem igual.

$C_A = \{C1, C2\} \cap \{C1, C2, A\} - \{C1, C2, p, q, r\} \cup A = \{A\}$

$C_p = \{C1, C2\} \cap \{C1, C2, A, p\} - \{A, C1, C2\} \cup p = \{p\}$

$C_q = \{C1, C2\} \cap \{C1, C2, A, p, q\} - \{A, C1\} \cup q = \{C2, q\}$

$C_r = \{C1, C2\} \cap \{C1, C2, A, p, q, r\} - \{A, C2\} \cup r = \{C1, r\}$

Dos conjuntos encontrados apenas os CIM (que enchergam o conjunto inicial) que não foram pra fila ainda.

Fila de prioridades = {{C1, r}, {C2, q}, {A}, {p}}.

Para c_i = {C1, r}:

C_{C1} = ordem menor a todos os elementos.

C_{C2} = C1 não é adjacente e r é de ordem maior.

$C_A = \{C1, r\} \cap \{C1, C2, A\} - \{C1, C2, p, q, r\} \cup A = \{A\}$

$C_p = \{C1, r\} \cap \{C1, C2, A, p\} - \{A, C1, C2\} \cup p = \{p\}$

$C_q = \{C1, r\} \cap \{C1, C2, A, p, q\} - \{A, C1\} \cup q = \{q\}$ (não maximal)

C_r = C1 não é adjacente e r é de ordem igual.

Fila de prioridades = {{C2, q}, {A}, {p}}.

Para c_i = {C2, q}:

C_{C1} = ordem menor a todos os elementos.

C_{C2} = ordem menor a todos os elementos.

$C_A = \{C2, q\} \cap \{C1, C2, A\} - \{C1, C2, p, q, r\} \cup A = \{A\}$

$C_p = \{C2, q\} \cap \{C1, C2, A, p\} - \{A, C1, C2\} \cup p = \{p\}$

C_q = C2 não é adjacente e q é de ordem igual.

$C_r = \{C2, q\} \cap \{C1, C2, A, p, q, r\} - \{A, C2\} \cup r = \{q, r\}$

Fila de prioridades = {{A}, {p}, {q, r}}.

Para c_i = {A}:

C_{C1} = ordem menor a todos os elementos.

C_{C2} = ordem menor a todos os elementos.

C_A = ordem menor a todos os elementos.

$C_p = \{A\} \cap \{C1, C2, A, p\} - \{A, C1, C2\} \cup p = \{p\}$

$C_q = \{A\} \cap \{C1, C2, A, p, q\} - \{A, C1\} \cup q = \{q\}$ (não maximal)

$C_r = \{A\} \cap \{C1, C2, A, p, q, r\} - \{A, C2\} \cup r = \{r\}$ (não maximal)

Fila de prioridades = {{p}, {q, r}}.

Para c_i = {p}:

C_{C1} = ordem menor a todos os elementos.

C_{C2} = ordem menor a todos os elementos.

C_A = ordem menor a todos os elementos.

C_p = ordem menor a todos os elementos.

C_q = não adjacente a todos os elementos.

C_r = não adjacente a todos os elementos.

Fila de prioridades = {{q, r}}.

Para c_i = {q, r}:

C_{C1} = ordem menor a todos os elementos.

C_{C2} = ordem menor a todos os elementos.

C_A = ordem menor a todos os elementos.

C_p = ordem menor a todos os elementos.

C_q = ordem menor a todos os elementos.

C_r = não adjacente a todos os elementos.

Fila de prioridades = {}.

Com a fila de prioridades vazia o algoritmo é encerrado.

CIM = {{C1, C2}, {C1, r}, {C2, q} {A}, {p}, {q, r}}

Para o algoritmo só é interessante os conjuntos com apenas literais.

CIM = {{p}, {q, r}}

ANEXO F – Exemplo Resolução MHFSAT

A fórmula abaixo será utilizada para exemplificar a resolução.

$$(p \vee q) \wedge (p \vee r) \wedge \neg p \wedge (\neg p \vee \neg s \vee \neg t)$$

O primeiro passo é separar as duas partes:

2-CNF = $(p \vee q) \wedge (p \vee r)$ e Horn = $\neg p \wedge (\neg p \vee \neg s \vee \neg t)$

Com a parte 2-CNF são gerados todos os CIM, CIM = $\{\{p\}, \{q, r\}\}$.

Para cada conjunto de c de CIM é aplicado HORNSAT, com a valoração V para cada literal l_i de c. Caso exista algum conjunto que satisfaça a parte Horn também, a fórmula é dita como SAT, caso contrário é dita como UNSAT.

Para $\{p\}$ HORNSAT será F.

Para $\{q, r\}$ HORNSAT será V com a seguinte valoração:

- p = F;
- q = V;
- r = V;
- s = F;
- t = F;

Tornando a fórmula SAT e encerrando o algoritmo.

Para fins didáticos é possível testar o resultado:

$$(p \vee q) \wedge (p \vee r) \wedge \neg p \wedge (\neg p \vee \neg s \vee \neg t)$$
$$(F \vee V) \wedge (F \vee V) \wedge \neg F \wedge (\neg F \vee \neg F \vee \neg F)$$
$$(F \vee V) \wedge (F \vee V) \wedge V \wedge (V \vee V \vee V)$$
$$(V) \wedge (V) \wedge V \wedge (V \vee V)$$
$$(V) \wedge (V) \wedge V \wedge (V)$$
$$(V) \wedge (V) \wedge V$$
$$(V) \wedge (V)$$
$$(V)$$