Solução de problemas em Grafos através da Lógica Monádica de Segunda Ordem e da Decomposição em Árvore

Glasielly Demori Proença

PROGRAMA DE PÓS-GRADUAÇÃO DA FACOM-UFMS

Data de Depósito:

Assinatura:_____

Glasielly Demori Proença

Orientador: *Prof. Dr. Vagner Pedrotti*

Dissertação apresentada à Faculdade de Computação - FACOM-UFMS como parte dos requisitos necessários à obtenção para o título de Mestre em Ciências de Computação.

UFMS - Campo grande
Fevereiro/2017

Aos meus pais,
Gervásio e Leila,

Aos meus irmãos,
Rony e Gleyscol.

Agradecimentos

Agradeço a Deus por ter me dado capacidade e condições para realizar essa pesquisa.

Meu agradecimento especial à minha família, meus pais Gervásio Proença e Leila Demori, meu irmão Rony Demori, por confiarem e acreditarem em mim e por muitas vezes se deslocarem da sua cidade para estarem comigo nos períodos mais difíceis. Um agradecimento ao meu irmão Gleyscol Demori, que hoje não está aqui, mas que também acreditava na minha força de vontade e me apoiava em minhas decisões, com certeza também contribuiu para que eu chegasse onde estou.

Um agradecimento ao meu professor orientador Vagner Pedrotti pela paciência, conhecimento compartilhado e auxílio no desenvolvimento dessa pesquisa.

Agradeço os professores do Programa de Mestrado, os professores da banca que se interessaram em avaliar meu trabalho e os técnicos que sempre foram prestativos e atenciosos.

Agradeço os amigos que estiveram comigo dando apoio e incentivo, em especial Lauro Maycon, Juliana Galete e Ana Caroline.

Agradeço a todas as pessoas que tiveram participação direta ou indiretamente, incentivando essa conquista.

Abstract

The Courcelle's Theorem states that every problem definable in Monadic Second Order Logic (MSOL) can be solved in graphs that present a tree decomposition with limited treewidth by a constant in linear running time through a fixed parameter algorithm. But, as in any algorithm, its running time depends on a constant, which, in turn, depends on the limit of treewidth and the size of the MSO expression, presenting super-exponential growth as the treewidth increases. Some authors have presented approaches to avoid this huge constant problem. In this work, we explore research described in [12], which deals with this problem using game theory to evaluate the MSO expression using the tree decomposition of the input graph. In this work are described the concepts of fixed parameter tractability, tree decomposition and treewidth, and the approach to the evaluation of an MSO expression with a focus on experimental validation of the running time complexity of the algorithm.

Resumo

O Teorema de Courcelle afirma que todo problema definível em Lógica Monádica de Segunda Ordem (LMSO) pode ser resolvido em grafos que apresentam uma decomposição em árvore com *treewidth* limitada por uma constante, em tempo de execução linear, através de um algoritmo de parâmetro fixo. Mas como todo algoritmo, seu tempo de execução depende de uma constante, a qual depende do limite da *treewidth* e do tamanho da expressão MSO, apresentando crescimento super exponencial à medida que a *treewidth* aumenta. Alguns autores têm apresentado abordagens para evitar esse problema da constante. Neste trabalho, exploramos uma pesquisa feita em [12], que trata esse problema utilizando teoria dos jogos para a avaliação da expressão MSO, a partir da decomposição em árvore do grafo de entrada. São descritos neste trabalho, os conceitos sobre a tratabilidade por parâmetro fixo, sobre decomposição em árvore e *treewidth*, e a abordagem para a avaliação de uma expressão MSO com um foco na validação experimental da complexidade do tempo de execução do algoritmo.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO

1

Introdução

Grafos podem ser usados para modelar problemas práticos, que geralmente se aplicam na vida real. Infelizmente, muitos destes problemas não possuem algoritmos eficientes.

Os problemas para os quais não existem algoritmos eficientes, ou seja, algoritmo de tempo polinomial, são chamados de problemas intratáveis. Na década de 70, foi descoberto que muitos destes problemas se tornam fáceis, se aplicados em árvores. Isso levou os pesquisadores a estudarem formas para medir o quanto um grafo se assemelha a uma árvore. Notou-se que quanto mais um grafo se parece com uma árvore, mais fácil se torna a solução de vários problemas intratáveis sobre esse grafo. Com base nisso, foi introduzido o conceito de *treewidth* [14], um inteiro que mede o quanto um grafo se assemelha a uma árvore.

A *treewidth* de um grafo é calculada a partir de uma *decomposição em árvore do grafo*, como veremos na Seção 1.1.1.

Alguns anos depois foi mostrado que grafos com *treewidth* limitada admitem algoritmos de tempo linear para certos problemas, como Cobertura de Vértices, Conjunto Independente Máximo e k−Coloração [1, 2, 5].

Alguns resultados importantes sobre problemas em grafos, foram mostrados usando Lógica Monádica de Segunda Ordem (LMSO). Courcelle [7] utilizou tais resultados sobre decomposição em árvore e mostrou que dada uma expressão em LMSO para um problema de decisão em grafos e um grafo com *treewidth* limitada, existe um algoritmo Tratável por Parametro Fixo (TPF) que resolve tal problema no grafo dado. Este foi um resultado célebre, conhecido como Teorema de Courcelle, conforme Seção 2.2.

A Tratabilidade por Parâmetro Fixo (TPF) é uma outra abordagem para

lidar com problemas intratáveis na prática, afim de encontrar uma solução de tempo polinomial que depende da instância do problema.

Algumas tentativas práticas de desenvolver o algoritmo do Teorema originalmente proposto por Courcelle, mostraram que construir uma estrutura que represente a expressão *MSO* computacionalmente e que reconheça-a como verdadeira ou falsa, não é uma tarefa fácil. Vários experimentos têm mostrado diferentes abordagens para a implementação de tal algoritmo que têm apresentado bons resultados. Dentre essas abordagens estão a Teoria dos Jogos e Programação Dinâmica.

Outros resultados, estendem a lógica *MSO* para melhor aplicá-la a diferentes problemas em grafos, juntamente com outros tipos de decomposições, como decomposições em caminhos e decomposições em cliques [13].

Neste trabalho, apresentamos uma pesquisa sobre aplicações da decomposição em árvores para reduzir um problema em grafos a instâncias menores deste problema e combiná-las para resolver o problema original. Listamos também alguns problemas de decisão em grafos que já foram expressos na *LMSO*.

Focamos no estudo de um algoritmo que utiliza a programação dinâmica para construir uma solução para o Teorema de Courcelle a partir da decomposição em árvore do grafo e uma expressão *MSO* que define um problema em grafos. Tal algoritmo utiliza uma função de minização que permite sua aplicação em problemas de otimização. A técnica utilizada nesse algoritmo é a construção de jogos que avaliam a expressão *MSO* e responde se tal expressão sobre um dado grafo juntamente com sua decomposição é verdadeira ou falsa.

Implementamos o algoritmo citado anteriormente desenvolvido em [12] e executamos experimentos afim de avaliar o tempo de execução.

1.1 Definições

Denota-se por $G = (V,E)$ um grafo não orientado, onde V é um conjunto de vértices, e E é um conjunto de pares de vértices xy, tal que $xy \in E$ se, e somente se, existe uma aresta entre x e y em G [6].

Um *laço* em um grafo G, é uma aresta xx, tal que $x \in V(G)$. Duas ou mais arestas que ligam o mesmo par de vértices em um grafo são chamadas *arestas paralelas*. Um grafo é *simples* quando não possui laços nem arestas paralelas.

Um *caminho* em um grafo G, que sai do vértice x até o vértice y, é uma sequência de vértices $(a_0, a_1, ..., a_n)$, onde $x = a_0$ e $y = a_n$, sem repetir vértices, e cada par $a_i a_{i+1}$ com $0 \leq i \leq n$, pertence a $E(G)$.

Sejam $G = (V,E)$ um grafo arbitrário e S um subconjunto de vértices de G. O *subgrafo induzido* por S, denotado por $G[S]$, é o subgrafo cujo conjunto de

vértices é S e xy é uma aresta de $G[S]$ se, e somente se, xy é uma aresta em G e $\{x,y\} \subseteq S$.

1.1.1 Decomposição em Árvore e Treewidth

Definição 1. ([14]) Dado um grafo $G = (V,E)$, um par $D = (X,T)$ é uma decomposição em árvore de G se, $X = \{X_i : i \in I\}$ tal que X_i é um subconjunto de vértices, definido como *bags* e $T = (I,F)$, é uma árvore, onde I é o conjunto de *nós* de T, e F é o conjunto de arestas de T. Temos que D deve satisfazer as seguintes propriedades:

1. $\bigcup_{i \in I} X_i = V(G)$, ou seja, cada vértice em G pertence a pelo menos um nó em T;
2. Para toda aresta $uv \in E(G)$, existe $i \in I$ tal que $\{u,v\} \subseteq X_i$;
3. Para i, j e $k \in I$, se k está no caminho único de i a j em T, então $X_i \cap X_j \subseteq X_k$.

O último item pode ser reescrito da seguinte forma:

- $T[X^v]$ é uma subárvore induzida por X^v, sendo $X^v = \{X_i \mid X_i \in X \text{ e } v \in X_i\}$.

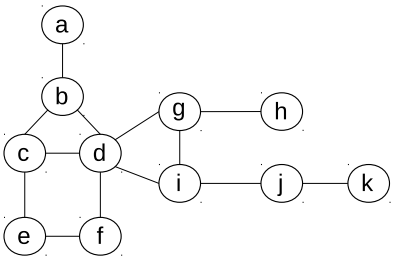


Figura 1.1: Grafo G.

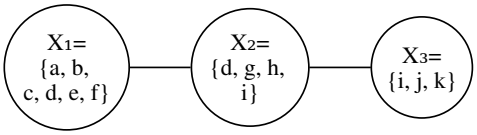


Figura 1.2: Uma decomposição em árvore para G.

Todo grafo tem uma decomposição em árvore trivial, isto é, um único nó raiz, contendo $V(G)$.

A *treewidth* $Tw_G(D)$, ou *largura da decomposição em árvore* de um grafo G, é o tamanho do maior subconjunto X_i de D menos um, ou seja, $Tw_G(D) = máximo\{|X_i| - 1\}$.

Definimos a *treewidth* $Tw(G)$ de um grafo G, como sendo a menor *treewidth* de todas as possíveis decomposições em árvore para G, ou seja, $Tw(G) = mínima\{Tw_G(D) \mid \text{D é uma decomposição em árvore em } G\}$.

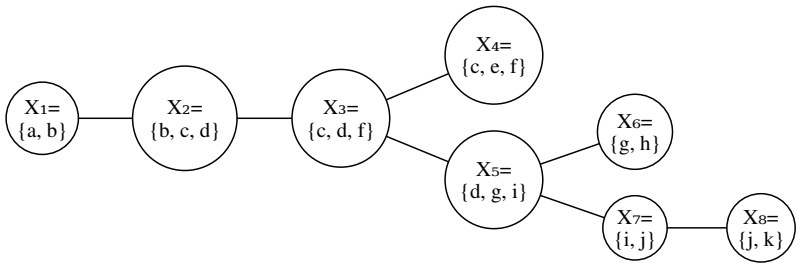


Figura 1.3: Outra possível decomposição em árvore para G.

Todo grafo que é uma árvore apresenta *treewidth* de valor 1.

Na Figura 1.1, temos um grafo G com 11 vértices, para o qual mostramos duas possíveis decomposições em árvore. Ressaltamos que a decomposição trivial de G seria uma *bag* X_1, contendo todos os vértices de G.

A Figura 1.2 apresenta uma decomposição em árvore para G, tal que a a maior *bag*, X_1, têm tamanho 6, logo, essa decomposição apresenta uma *treewidth* $Tw_G(D) = 5$.

A Figura 1.3 apresenta outra decomposição em árvore para G, tal que as maiores *bags* X_2, X_3, X_4 e X_5 tem tamanho 3. Logo, essa decomposição apresenta uma *treewidth* $Tw_G(D) = 2$, ou seja, uma decomposição com largura menor do que a apresentada na Figura 1.2. Quanto menor o tamanho das *bags*, a *treewidth* se aproxima mais da ótima.

O problema de encontrar a *treewidth* de um grafo qualquer é NP-Completo [4, 1], porém existe um algoritmo de tempo linear que, dada uma constante k e um grafo G, determina se a *treewidth* de G é no máximo uma constante k, além disso, encontra uma decomposição em árvore de G com *treewidth* $\leq k$ [3]. Esse k é uma constante pequena, que não cresce arbitrariamente.

1.1.2 Tratabilidade em Parâmetro Fixo

A tratabilidade em parâmetro fixo (TPF) é uma das abordagens utilizadas para solucionar problemas intratáveis. A ideia principal dessa abordagem é observar as instâncias do problema para identificar um parâmetro relacionado a sua complexidade, pelo qual pode-se separar da entrada o componente que causa a explosão combinatória.

Nas abordagens clássicas, temos como argumentos para a solução de um problema, a entrada do problema e a questão a ser resolvida. A entrada geralmente é um inteiro n ou é do tamanho de um inteiro n. Na abordagem TPF, temos também como argumento o parâmetro k que está associado com a instância do problema. Alguns problemas TPF podem ser vistos em [10].

Definição 2. *(Tratabilidade por parâmetro fixo).* Um problema parametrizável P é uniformemente tratável por parâmetro fixo se existe uma constante α e um

algoritmo que resolve o problema P em tempo $f(k)n^{\alpha}$, onde n é o tamanho da entrada, k é um parâmetro da entrada e $f : \mathbb{N} \to \mathbb{N}$ é uma função arbitrária.

CAPÍTULO

2

Fundamentação Teórica

Neste capítulo serão introduzidos os conceitos e alguns exemplos da lógica MSO, que é utilizada no Teorema de Courcelle. Uma abordagem para verificar a validade de uma expressão em tal lógica é apresentada aqui com um nível maior de detalhe, pois é de grande relevância para este trabalho.

A Seção 2.1 apresenta alguns exemplos de problemas em grafos que já foram definidos em LMSO. A Seção 2.2 trata de um modelo de verificação de fórmula MSO através da teoria dos jogos.

2.1 Lógica Monádica de Segunda Ordem (LMSO)

A Lógica de Segunda Ordem difere da Lógica de Primeira Ordem nas variáveis e quantificadores dos elementos do universo. Enquanto a Lógica de Primeira Ordem inclui somente variáveis e quantificadores individuais para elementos do universo, a Lógica de Segunda Ordem adiciona-os para relações sobre elementos do universo.

O termo *monádica*, restringe a lógica *MSO* a permitir a quantificação somente sobre relações unárias.

Definição 3. A LMSO consiste de uma linguagem na qual os predicados podem ser construídos usando:

1. Os conectivos lógicos $\wedge$ (e), $\vee$ (ou), $\neg$ (negação), $\Rightarrow$ (implicação) e $\Leftrightarrow$ (se, e somente se);

2. Dois tipos de variáveis que podem ser: as simples, que assumem como valores um elemento; e as de conjunto, que assumem como valores conjuntos de elementos;

3. Os quantificadores existencial ($\exists$) e universal ($\forall$);

4. Teste de pertinência $x \in Y$, onde x e Y são variáveis simples e de conjunto, respectivamente;

5. Teste de pertinência $(x, y, ..., z) \in R$, onde $(x, y, ..., z)$ é uma tupla contendo variáveis simples e R é uma relação fornecida como entrada para o predicado;

6. Teste de identidade $x = y$.

Expressar um problema em uma linguagem lógica, através de suas regras sintáticas, torna possível sua análise por um algoritmo. Ao longo do tempo, pesquisadores têm estudado a LMSO como uma linguagem poderosa para expressar problemas em grafos. Um resultado forte nessa área é o Teorema de Courcelle, que afirma que um problema em grafos expressível em LMSO pode ser resolvido em tempo linear sobre grafos que admitem uma *decomposição em árvore* de largura limitada por uma constante. É com base nesse Teorema que motivamos essa pesquisa.

Um problema de decisão em grafos pode ser definido em LMSO se existir uma expressão que define tal problema.

Definição 4. *[13].* Uma propriedade Π de um grafo é chamada MSO-definível se existe uma sentença MSO φ tal que para um grafo $G = (V, E)$ é verdade que $G \in \Pi \Leftrightarrow G \models \varphi$ [1].

Considerando V como conjunto universo, a sentença abaixo expressa o bem conhecido problema NP-completo da 3-coloração. A fórmula $3col$ em MSO tem como entrada um conjunto de arestas E.

$$\begin{aligned} 3col(E) \equiv & \exists X \exists Y \exists Z \ (\ Part(X,Y,Z) \wedge \forall x \forall y \ (\ xy \in E \wedge x \neq y \Rightarrow \\ & \neg(x \in X \wedge y \in X) \wedge \neg(x \in Y \wedge y \in Y) \wedge \neg(x \in Z \wedge y \in Z) \ ) \ ) \end{aligned}$$

O predicado $Part(X,Y,Z)$ expressa que (X,Y,Z) particiona o conjunto de vértices (o universo) em 3 partes (ou cores). O restante da expressão garante que dois vértices adjacentes não podem estar na mesma parte (ou com a mesma cor). O predicado $Part(X,Y,Z)$ é definido da seguinte forma:

$$\begin{aligned} Part(X,Y,Z) \equiv & \forall \ x \ ((\ x \in X \vee x \in Y \vee x \in Z) \wedge \\ & (\neg(x \in X \wedge x \in Y) \wedge \neg(x \in Y \wedge x \in Z) \wedge \neg(x \in X \wedge x \in Z))) \end{aligned}$$

[1] $G \models \varphi$ (lê-se φ é verdadeira em G)

O próximo exemplo é uma expressão para grafos não conexos em LMSO com universo V e tendo como entrada o conjunto de arestas E. Outras expressões podem ser vistas em [8].

$$Disconn(E) \equiv \exists X(\exists x(x \in X) \wedge \exists y(y \notin X) \wedge \forall s \forall t(st \in E \Rightarrow (s \in X \Leftrightarrow t \in X)))$$

A expressão $Disconn(E)$ é válida se existe uma componente X com pelo menos um vértice, e existe um vértice y fora de X, e para toda aresta st do grafo, os vértices s e t pertencem à mesma componente. Isto é, não existe uma aresta que conecta o elemento y à componente X. Logo o grafo é desconexo com pelo menos duas componentes.

Seja $G = (V,E)$ um grafo. O predicado $conn(X,F)$ definido a seguir é válido se o subgrafo H do grafo G, tal que X é o conjunto de vértices de H e F é o conjunto de arestas de H, é conexo.

Seja $F = \{e_i \mid i \in \mathbb{N}\}$ um conjunto de arestas, e $inc(x,e_i)$, um predicado que é válido se a aresta e_i incide no vértice x.

$$\begin{aligned} conn(X,F) = \forall Y(\ & (\forall y(y \in Y \Rightarrow y \in X) \wedge \exists y(y \in Y) \wedge \exists x(x \notin Y\ \wedge\ x \in X)) \Rightarrow \\ & \exists e \exists x \exists y(e \in F\ \wedge inc(x,e)\ \wedge\ inc(y,e)\ \wedge\ x \in X\ \wedge\ x \notin Y\ \wedge\ y \in Y)) \end{aligned}$$

A expressão $conn(X,F)$ é válido se para todo subconjunto $Y \subseteq X$, existe uma aresta que conecta o subconjunto Y ao subconjunto $X \setminus Y$. A sentença $(\forall y(y \in Y \Rightarrow y \in X) \wedge \exists y(y \in Y) \wedge \exists x(x \notin Y\ \wedge\ x \in X))$ é verdadeira sempre que Y é um subconjunto de X e existe pelo menos um elemento y que está em Y e pelo menos um elemento x que está em $X \setminus Y$. O restante da expressão verifica se toda essa primeira parte da expressão for verdadeira então existe uma aresta e que conecta um elemento de Y em um elemento de $X \setminus Y$. Em outras palavras se houver um subconjunto Y de X que não está conectado com $X \setminus Y$, então $conn(X,F)$ será falsa.

Nem todos os problemas em grafos podem ser expressos por LMSO. Portanto pesquisadores já têm estudado algumas extensões da lógica MSO para uso na solução de problemas em grafos. A lógica MSO_2 permite a quantificação sobre o conjunto de arestas, e também é chamada de lógica MSO classificação 2 [9]. Na MSO_2 o conjunto universo é formado pela união do conjunto dos vértices e conjunto das arestas. Assim, podemos afirmar que a lógica que vimos até agora é conhecida como MSO_1 ou de classificação 1. Essa extensão é ainda mais poderosa que a lógica que vimos até agora. Por exemplo, podemos expressar que um grafo tem Ciclo Hamiltoniano, o que não pode ser expresso em MSO_1, utilizando quantificadores para conjunto de arestas (estes exemplos para a lógica MSO_2 podem ser verificados em [13]).

O predicado $ham'(F,X)$, apresentado a seguir, avalia se o conjunto de arestas F forma um Ciclo Hamiltoniano no grafo G. Consideramos que $X = V(G)$, e uma aresta do universo e x um vértice do universo.

$$ham'(F,X) \equiv (conn(X,F) \wedge \forall x \exists e_1 \exists e_2 (e_1 \in F \wedge e_2 \in F \wedge inc(x,e_1) \wedge inc(x,e_2) \ \wedge \neg(e_1 = e_2) \wedge \\ \forall e_3((inc(x,e_3) \wedge e_3 \in F) \Rightarrow (e_3 = e_1 \vee e_3 = e_2))))$$

O predicado $ham''(F)$ a seguir é válido se $F \subseteq E$ for um Ciclo Hamiltoniano no grafo, enquanto a expressão ham é válida se existir tal subconjunto F no grafo.

$$ham''(F) \equiv \forall X(\forall x(x \in X) \Rightarrow ham'(F,X))$$
$$ham \equiv \exists F ham''(F)$$

A sentença $\forall X(\forall x(x \in X)$ é necessária, pois X deve ser igual ao conjunto de vértices do grafo, além disso, é parâmetro do predicado $conn(X,F)$ utilizado no predicado $ham'(F,X)$.

Note que, podemos reescrever a expressão ham da seguinte forma:

$$ham \equiv \exists F \forall X(\forall x(x \in X) \Rightarrow conn(X,F) \ \wedge \\ \forall x \exists e_1 \exists e_2(\ e_1 \in F \ \wedge \ e_2 \in F \ \wedge \ inc(x,e_1) \ \wedge \ inc(x,e_2) \ \wedge \\ \forall e_3(\ (\ inc(x,e_3) \ \wedge \ e_3 \in F \ ) \Rightarrow (\ e_3 = e_1 \vee e_3 = e_2 \ ))))$$

2.2 O Teorema de Courcelle

O Teorema de Courcelle afirma que todo problema definido em LMSO pode ser verificado em tempo linear sobre estruturas com *treewidth* limitada.

Teorema 1. *([7]). Seja P um problema definido por uma fórmula MSO φ e seja w um inteiro positivo. Existe um algoritmo A e uma função $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ tal que para todo grafo $G = (V,E)$ de ordem $n := |V(G)|$ e treewidth no máximo w, A resolve P sobre a entrada $G = (V,E)$ em tempo $f(\| \varphi \|, w) \cdot n$, onde $\| \varphi \|$ é o comprimento de φ.*

O problema P do Teorema 1 é um problema TPF, portanto o algoritmo A é de parâmetro fixo, e tal parâmetro é a *treewidth* w da decomposição em árvore do grafo G.

Algumas abordagens para a construção do algoritmo definido no Teorema de Courcelle foram estudadas, como por exemplo, a construção de um autômato que aceita uma decomposição em árvore de um dado grafo $G = (V,E)$ se,

e somente se, $G = (V,E)$ satisfaz a fórmula MSO φ que expressa o problema P. Existem algumas ferramentas que fazem a construção desse autômato, mas o espaço de memória utilizado é infactível na prática, conforme [13, 12].

O tempo de execução do problema P do Teorema de Courcelle é limitado por $O(n)f(\| \varphi \|,w)$. A função $f(\| \varphi \|,w)$ é uma iteração exponencial de altura $O(\| \varphi \|)$,

$$f(\| \varphi \|,w) = 2^{2^{\cdot^{\cdot^{2^w}}}} \Big\} O(\| \varphi \|)$$

De acordo com [11] é impossível fornecer qualquer tempo de execução eficiente quando a função f for limitada superiormente por uma exponencial iterada de altura limitada em termos de φ e w para fórmulas φ arbitrárias. Porém, para w e φ fixos o problema P do Teorema de Courcelle se torna TPF.

Kneis em [12] apresenta uma abordagem para o modelo de checagem de uma expressão *MSO* considerando uma determinada instância de entrada, baseada na teoria dos jogos, que na prática tem mostrado bons resultados. Nesta abordagem, basicamente a fórmula *MSO* é avaliada no grafo usando um algoritmo recursivo para a verificação. Tal algoritmo é um jogo entre dois jogadores chamados *verificador* e *falsificador*, o primeiro tenta provar que a fórmula vale para o grafo de entrada, enquanto o segundo tenta provar que a fórmula não é verdadeira.

Para facilitar o entendimento, assumimos nesse primeiro momento que a posição de um jogo é parte da expressão *MSO*, por exemplo, na expressão

$$\forall x \exists y(adj(x,y))$$

as expressões $\forall x$, $\exists y$ e $adj(x,y)$ são posições do jogo. Uma posição do jogo é definida formalmente como uma tupla na Definição 8 da Seção 2.2.2.

Neste jogo, o *falsificador* é o jogador universal e seus movimentos são baseados nas fórmulas universais $(\wedge, \forall)$, isto é, cada posição do jogo é uma parte da expressão *MSO* e quando essa parte da expressão é um $\wedge$ ou $\forall$ então essa posição é do *falsificador*. Enquanto o *verificador* é o jogador existencial e seus movimentos são baseados nas fórmulas existenciais $(\vee, \exists)$, isto é, quando a parte da expressão é um $\vee$ ou $\exists$ essa posição é do *verificador*.

Na próxima seção é apresentada a estrutura de uma instância a ser expandida na expressão *MSO*. Por exemplo, para expandir a expressão do problema do conjunto dominante para um determinado grafo G, devemos definir uma estrutura para o grafo. Expandir uma expressão significa atribuir uma entrada para a expressão assumindo valores do universo dessa entrada para as variáveis da expressão, com o objetivo de avaliá-la como verdadeira ou falsa para a entrada considerada.

2.2.1 Estrutura

Definimos estrutura, de acordo com a notação de [12]. Para tanto, alguns conceitos elementares são necessários e são apresentados a seguir.

Um vocabulário τ é um conjunto de símbolos, tal que cada símbolo possuiuma aridade. Para aridade zero, cada símbolo pode ser mapeado em um elemento do universo. Para as outras aridades, cada símbolo é mapeado em um conjunto de tuplas de tamanho igual a aridade. Chamamos de $nulário(\tau)$ o conjunto dos símbolos nulários em τ, que possuem aridade zero, $relação(\tau)$ o conjunto dos símbolos relacionais em τ, que possuem aridade maior ou igual que 1, e $unário(\tau)$ o conjunto dos símbolos unários em τ, que possuem aridade 1.

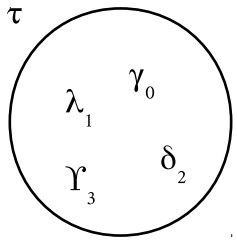


Figura 2.1: Ilustração de um vocabulário τ.

A Figura 2.1 apresenta um exemplo de um vocabulário τ, que possui os símbolos: δ_2, um símbolo de relação binária, ou seja, com aridade 2; λ_1, um símbolo de relação unária; γ_0, um símbolo nulário; e Υ_3, um símbolo relacional com aridade 3.

Uma estrutura $\mathscr{A}$ sobre τ é uma tupla $\mathscr{A} = \left(A, (R^{\mathscr{A}})_{R \in relação(\tau)}, (c^{\mathscr{A}})_{c \in nulário(\tau)}\right)$, onde A é o universo de $\mathscr{A}$, $(R^{\mathscr{A}})_{R \in relação(\tau)}$ é a interpretação do símbolo relacional R contida no universo de $\mathscr{A}$ e $(c^{\mathscr{A}})_{c \in nulário(\tau)}$ é a interpretação do símbolo nulário c contida no universo de $\mathscr{A}$.

Utilizamos aqui letras caligráficas para denotarem uma estrutura, por exemplo, $\mathscr{A}$ e $\mathscr{H}$, e a mesma letra em fonte normal e maiúscula para denotar o universo de tal estrutura. Seguindo o exemplo temos respectivamente os universos A e H.

Para um símbolo de relação $R \in relação(\tau)$, temos que $R^{\mathscr{A}} \subseteq A^{aridade(R)}$. Isto é, o símbolo de relação R é mapeado para uma relação sobre elementos do universo A, mantendo sua aridade em τ. Por exemplo, sejam $A = \{1,2,3\}$ o universo e R_2 um símbolo relacional com aridade 2 no vocabulário τ. Nesse caso, R_2 pode ser interpretado em $\mathscr{A}$ como qualquer relação binária sobre A que possua aridade 2, por exemplo, $\{(1,3),(3,2)\}$.

Para um símbolo nulário $c \in nulário(\tau)$, temos que ou $c^{\mathscr{A}} \in A$, ou seja, c é mapeado para algum elemento do universo A em $\mathscr{A}$, ou $c^{\mathscr{A}} = nulo$ e dizemos

que c não é interpretado em $\mathscr{A}$. O conjunto de símbolos nulários que são interpretados em $\mathscr{A}$ é definido como $interpretado(\mathscr{A})$.

Para conjuntos de símbolos de relação $\bar{R} = \{R_1, ..., R_l\} \subseteq relação(\tau)$, denotamos $\bar{R}^{\mathscr{A}} := \{R^{\mathscr{A}} \mid R \in \bar{R}\}$, isto é, todo elemento do conjunto $\bar{R}^{\mathscr{A}}$ é a interpretação em $\mathscr{A}$ de um símbolo relacional em $\bar{R}$. E para conjuntos de símbolos nulários $\bar{c} = \{c_1, ..., c_m\} \subseteq nulário(\tau)$, temos que $\bar{c}^{\mathscr{A}} := \{c^{\mathscr{A}} \mid c \in \bar{c} \cap interpretado(\mathscr{A})\}$, isto é, todo elemento do conjunto $\bar{c}^{\mathscr{A}}$ é a interpretação em $\mathscr{A}$ de um símbolo nulário, ou seja, excluem-se de $\bar{c}$ os símbolos nulários que não são interpretados em $\mathscr{A}$.

Seja $\mathscr{A}$ uma estrutura sobre τ (que pode ser denominada por $\tau-$estrutura) e $\bar{a} = \{a_1, ..., a_m\} \subseteq A$. Então $\mathscr{A}[\bar{a}]$ é a subestrutura de $\mathscr{A}$ induzida por $\bar{a}$, onde $\mathscr{A}[\bar{a}]$ tem universo $\bar{a}$ e para cada símbolo relacional $R \in \tau$ temos que $R^{\mathscr{A}[\bar{a}]} = R^{\mathscr{A}} \cap \bar{a}^{aridade(R)}$, ou seja, excluem-se das interpretações dos símbolos relacionais de $\mathscr{A}$ todas as tuplas com elementos não contidos em $\bar{a}$, e símbolos nulários c são interpretados como $c^{\mathscr{A}[\bar{a}]} = c^{\mathscr{A}}$ se $c^{\mathscr{A}} \in \bar{a}$, caso contrário permanecem não interpretados.

Por exemplo, um grafo $G = (V, E)$ pode ser identificado facilmente como uma estrutura $\mathscr{G}^*$ sobre o vocabulário $\tau_{Grafo} = \{(adj)\}$, onde adj é um símbolo binário único em τ_{Grafo}, e interpretado por $\mathscr{G}^*$ para $E(G)$. O universo de $\mathscr{G}^*$ é V, e todo $xy \in E$ se, e somente se, $\{(x,y),(y,x)\} \subseteq adj^{\mathscr{G}^*}$. O grafo P_3 da Figura 2.2 é uma estrutura $\mathscr{G}^*$, seu universo é $V = \{a,b,c\}$, o símbolo adj do seu vocabulário é interpretado como $E = \{ab, bc\}$.

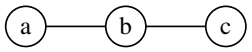


Figura 2.2: Um grafo P_3.

Operações sobre Estruturas. Definimos aqui as operações de , e sobre estruturas. Essas definições serão utilizadas no Capítulo 3.

Definição 5. *(Isomorfismo entre estruturas).* Duas τ-estruturas $\mathscr{A}$ e $\mathscr{B}$ são isomorfas, denotado por $\mathscr{A} \cong \mathscr{B}$, se existir uma bijeção $h : A \to B$ entre os universos A e B, e;

- para todo $c \in nulário(\tau)$, c é interpretado em $\mathscr{A}$ se e somente se, c é interpretado em $\mathscr{B}$;
- para todo símbolo c que é interpretado em τ, a interpretação de c em $\mathscr{A}$ é mapeada para a interpretação de c em $\mathscr{B}$, por h;
- para todo símbolo relacional $R \in \tau$ e $a_1, ..., a_p \in A$, onde p é a aridade de R, temos que $(a_1, ..., a_p)$ está contido na interpretação de R em $\mathscr{A}$, se e somente se, $(h(a_1), ..., h(a_p))$ estiver na interpretação de R em $\mathscr{B}$.

Na Tabela 2.1 as estruturas $\mathscr{A}$ e $\mathscr{B}$ são isomorfas, tomando a bijeção entre os universos $h : A \mapsto B = \{(2,6),(1,1),(3,5),(4,3)\}$. Os símbolos c,d,e e f são nulários e o símbolo Y é relacional.

Estrutura $\mathscr{A}$	
τ	$\{Y,c,d,e,f\}$
A	$\{1,2,3,4\}$
Interpretações	
c	2
d	1
e	*nulo*
f	3
Y	$\{2,3,4\}$

Estrutura $\mathscr{B}$	
τ	$\{Y,c,d,e,f\}$
B	$\{1,3,5,6\}$
Interpretações	
c	6
d	1
e	*nulo*
f	5
Y	$\{6,5,3\}$

Tabela 2.1: Exemplo de estruturas isomorfas.

Definição 6. *(Estruturas compatíveis).* Duas τ-estruturas $\mathscr{A}$ e $\mathscr{B}$ são compatíveis se para todo símbolo nulário c interpretado em ambas estruturas, temos que a interpretação de c em $\mathscr{A}$ é igual a interpretação de c em $\mathscr{B}$ e a função identidade $x \mapsto x$ é um isomorfismo entre $\mathscr{A}[A \cap B]$ e $\mathscr{B}[A \cap B]$.

Observe as estruturas $\mathscr{A}$ e $\mathscr{B}$ da Tabela 2.2. Todos os símbolos nulários que são interpretados em ambas as estruturas têm a mesma interpretação. Note que as estruturas induzidas $\mathscr{A}[A \cap B]$ e $\mathscr{B}[A \cap B]$ são isomorfas, tomando a bijeção entre os universos $h : A_a \mapsto B_b = \{(2,2),(1,1),(3,3)\}$. Portanto $\mathscr{A}$ e $\mathscr{B}$ são estruturas compatíveis.

$\mathscr{A}$	
τ	$\{Y,c,d,e,f\}$
A	$\{1,2,3,4\}$
Interpretações	
c	2
d	1
e	4
f	3
Y	$\{2,3,4\}$

$\mathscr{B}$	
τ	$\{Y,c,d,e,f\}$
B	$\{1,2,3,5,6\}$
Interpretações	
c	2
d	1
e	*nulo*
f	3
Y	$\{2,3,5,6\}$

$\mathscr{A}[A \cap B]$	
τ	$\{Y,c,d,e,f\}$
A_a	$\{1,2,3\}$
Interpretações	
c	2
d	1
e	*nulo*
f	3
Y	$\{2,3\}$

$\mathscr{B}[A \cap B]$	
τ	$\{Y,c,d,e,f\}$
B_b	$\{1,2,3\}$
Interpretações	
c	2
d	1
e	*nulo*
f	3
Y	$\{2,3\}$

Tabela 2.2: Exemplo de compatibilidade entre as estruturas $\mathscr{A}$ e $\mathscr{B}$.

Definição 7. *(União entre estruturas).* A união de τ-estruturas $\mathscr{A}_1$ e $\mathscr{A}_2$ compatíveis, denotada por $\mathscr{A}_1 \cup \mathscr{A}_2$ é definida como uma τ-estrutura com universo $A = A_1 \cup A_2$ e para todo símbolo relacional $R \in \tau$ em $\mathscr{A}_1 \cup \mathscr{A}_2$, sua interpretação é a união das interpretações de R em $\mathscr{A}_1$ e $\mathscr{A}_2$. Os símbolos nulários c que não são interpretados em ambas as estruturas, permanecem não interpretados

em $\mathscr{A}_1 \cup \mathscr{A}_2$, por outro lado, se c é interpretado em $\mathscr{A}_1$ ou $\mathscr{A}_2$, então c mantém a interpretação de uma das estruturas em $\mathscr{A}_1 \cup \mathscr{A}_2$.

2.2.2 Jogo para Avaliação da Expressão MSO

Um jogo $\mathscr{G} = (P, M, P_0, P_1, p_0)$ entre dois jogadores é formado por um conjunto de posições P, dois conjuntos disjuntos $P_0, P_1 \subseteq P$ de posições atribuídas para cada jogador, digamos Jogador 0 e Jogador 1, uma posição inicial $p_0 \in P$ e uma relação binária acíclica $M \subseteq P \times P$, que corresponde aos movimentos válidos no jogo. Os movimentos são permitidos a partir da posição atribuída a um dos dois jogadores.

Para $p \in P$ seja $\mathit{próximo}_{\mathscr{G}}(p) = \{p' \in P \mid (p, p') \in M\}$ o conjunto de posições que podem ser alcançadas a partir de p através de um movimento em M. Para qualquer posição $p \in P$, seja $\mathit{subjogo}_{\mathscr{G}}(p) = (P, M, P_0, P_1, p)$ um subjogo de $\mathscr{G}$ que inicia da nova posição inicial p.

Regras

A sequência de posições do jogo $\mathscr{G}$ é uma sequência maximal, tal que entre duas posições subsequentes existe um movimento válido. A regra diz que o jogador atribuído para a i-ésima posição (p_i) deve jogar um movimento válido, ou seja, deve escolher a próxima posição $p_{i+1} \in \mathit{próximo}(p_i)$. Se a posição p_{i+1} não existir, o jogo termina. Nesse caso, o jogador atribuído para a posição p_i perde o jogo. O objetivo do jogo é forçar o jogador adversário a entrar em uma posição em que ele não possa se mover, isto é, forçá-lo a perder o jogo.

A lógica MSO

Definimos como $MSO(\tau)$, o conjunto de sentenças Monádicas de Segunda Ordem para o vocabulário τ. O conjunto $MSO(\tau)$ contém a fórmula *atômica* $R(c_1, ..., c_p)$ para todo símbolo de relação $R \in \tau$ e quaisquer símbolos nulários $c_1, ..., c_p \in \tau$. Também podemos denotar $R(c_1, ..., c_p)$ como $(c_1, ..., c_p) \in R$.

Se φ, ψ estão em $MSO(\tau)$, então podemos aplicar os conectivos lógicos: $\neg$ (negação), $\vee$ (ou), $\wedge$ (e), tal que $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$ estão em $MSO(\tau)$.

Se $\varphi \in MSO(\tau \cup \{c\})$ para algum símbolo nulário c, tal que $c \notin \tau$, então ambos $\forall c\varphi$ e $\exists c\varphi$ estão em $MSO(\tau)$.

Se $\varphi \in MSO(\tau \cup \{R\})$ para algum símbolo de relação R, tal que $R \notin \tau$, então ambos $\forall R\varphi$ e $\exists R\varphi$ estão em $MSO(\tau)$.

Para definição do jogo, o símbolo de negação $\neg$ ocorre somente sobre fórmulas atômicas. Sobre uma τ-estrutura $\mathscr{A}$ que interpreta totalmente τ, isto é, todos os símbolos de τ são interpretados em $\mathscr{A}$ e uma fórmula $\varphi \in MSO(\tau)$, escrevemos que $\mathscr{A} \models \varphi$ se, e somente se, φ é verdade em $\mathscr{A}$.

Tomamos, como exemplo, o problema do conjunto dominante para um grafo $G = (V,E)$, que é um subconjunto S de V tal que cada vértice de V ou está em S ou é adjacente a pelo menos um vértice em S. Considere a fórmula $cd(S) \in MSO(\tau \cup \{S\})$, tal que S é uma variável de conjunto livre fornecida como entrada para a fórmula $cd(S)$:

$$cd(S) = \forall x(x \in S \ \vee \ \exists y(y \in S \wedge adj(x,y)))$$

A Figura 2.3 ilustra a árvore da expressão $cd(S)$, e cada *nó* interno da árvore é uma fórmula atômica, uma fórmula atômica negada, uma variável quantificada, ou um operador lógico e todas as folhas são fórmulas atômicas ou fórmulas atômicas negadas.

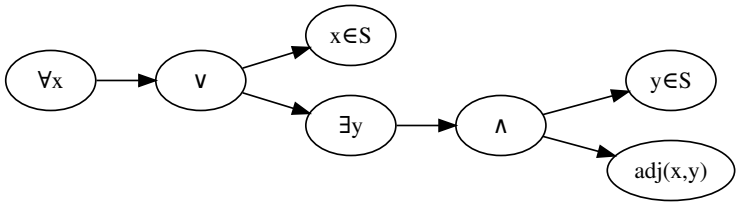


Figura 2.3: Árvore da fórmula $cd(S)$.

Construção do Jogo

Definição 8. (Jogo de Avaliação [12]). O clássico *Jogo de Avaliação* $MC(\mathscr{A},\varphi) = (P,M,P_0,P_1,p_0)$ sobre uma τ-estrutura $\mathscr{A}$ que interpreta totalmente τ e uma fórmula $\varphi \in MSO(\tau)$ é definido recursivamente sobre a fórmula φ como segue. Seja $p_0 = (\mathscr{A}[\bar{c}^{\mathscr{A}}],\varphi)$, onde $\bar{c} = nulário(\tau)$.

Vamos denotar Jogador 0 como *falsificador* e Jogador 1 como *verificador*. Portanto, P_0 são as posições do *falsificador* e P_1 são as do *verificador*.

1. Se φ é uma fórmula atômica ou negada, então $MC(\mathscr{A},\varphi) = (\{p_0\},\emptyset,P_0,P_1,p_0)$, onde:

   - A posição inicial p_0 pertence ao *falsificador* se, e somente se
     - $\varphi = (c_1,...,c_p) \in R$ e $(c_1^{\mathscr{A}},...,c_p^{\mathscr{A}}) \in R^{\mathscr{A}}$; ou
     - $\varphi = (c_1,...,c_p) \notin R$ e $(c_1^{\mathscr{A}},...,c_p^{\mathscr{A}}) \notin R^{\mathscr{A}}$.
       Nesses casos $P_1 = \emptyset$.
   - A posição inicial p_0 pertence ao *verificador* se, e somente se
     - $\varphi = (c_1,...,c_p) \in R$ e $(c_1^{\mathscr{A}},...,c_p^{\mathscr{A}}) \notin R^{\mathscr{A}}$; ou
     - $\varphi = (c_1,...,c_p) \notin R$ e $(c_1^{\mathscr{A}},...,c_p^{\mathscr{A}}) \in R^{\mathscr{A}}$.
       Nesses casos $P_0 = \emptyset$.

Por exemplo, considere que a estrutura $\mathscr{G}^*$ é baseada no grafo G_2 da Figura 2.4 e que $\varphi = cd(S)$, tal que $S = \{a,b\}$ e $x = a$ e $y = b$. Para avaliarmos $\psi = adj(x,y)$ temos que $\psi = (c_1,...,c_p) \in R$ e $(c_1^{\mathscr{A}},...,c_p^{\mathscr{A}}) \in R^{\mathscr{A}}$, pois a e $b \in adj$ e ab é uma aresta em G.

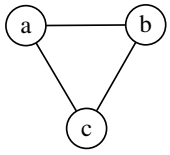


Figura 2.4: Um grafo G_2.

Podemos concluir que $p_0 \in P_0$ sempre que a fórmula atômica ou negada φ é verdadeira e, nesse caso, o *falsificador* perde o jogo e o *verificador* vence. Enquanto que $p_0 \in P_1$ sempre que a fórmula atômica ou negada φ é falsa e, nesse caso, *verificador* perde o jogo e o *falsificador* vence.

2. Se $\varphi \in \{\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2\}$, sejam $MC(\mathscr{A},\psi_1) = (P_{\psi_1}, M_{\psi_1}, P_{0,\psi_1}, P_{1,\psi_1}, p_{\psi_1})$ o jogo de avaliação sobre $\mathscr{A}$ e ψ_1, e $MC(\mathscr{A},\psi_2) = (P_{\psi_2}, M_{\psi_2}, P_{0,\psi_2}, P_{1,\psi_2}, p_{\psi_2})$ o jogo de avaliação sobre $\mathscr{A}$ e ψ_2. Então $MC(\mathscr{A},\varphi) = (P,M,P_0,P_1,p_0)$, onde:

   - P é a união de p_0 com $P_{\psi_1} \cup P_{\psi_2}$;
   - M é a união de M_{ψ_1} e M_{ψ_2} com $\{(p_0,p_{\psi_1}),(p_0,p_{\psi_2})\}$;
   - $P_0 = P_{0,\psi_1} \cup P_{0,\psi_2} \cup \{p_0\}$ se, e somente se $\varphi = \psi_1 \wedge \psi_2$; ou $P_0 = P_{0,\psi_1} \cup P_{0,\psi_2}$ se $\varphi = \psi_1 \vee \psi_2$; e
   - $P_1 = P_{1,\psi_1} \cup P_{1,\psi_2} \cup \{p_0\}$ se, e somente se $\varphi = \psi_1 \vee \psi_2$; ou $P_1 = P_{1,\psi_1} \cup P_{1,\psi_2}$ se $\varphi = \psi_1 \wedge \psi_2$.

   Ou seja, $MC(\mathscr{A},\varphi)$ é a união dos dois subjogos $MC(\mathscr{A},\psi_1)$ e $MC(\mathscr{A},\psi_2)$, com uma nova posição inicial p_0 e mais dois movimentos $\{(p_0,p_{\psi_1}),(p_0,p_{\psi_2})\}$.

3. Se $\varphi \in \{\forall c\psi, \exists c\psi\}$, para algum símbolo nulário c, seja $MC(\mathscr{A}_a,\psi) = (P_a,\ M_a,\ P_{0,a},\ P_{1,a},\ p_a)$ o jogo de avaliação sobre $\mathscr{A}_a$ e ψ, tal que $\mathscr{A}_a$ é uma estrutura expandida de $\mathscr{A}$, isto é, o símbolo c foi adicionado ao vocabulário de $\mathscr{A}$ em $\mathscr{A}_a$, tal que a interpretação de c na estrutura $\mathscr{A}_a$ é igual a a, para $a \in A$. Então $MC(\mathscr{A},\varphi) = (P,M,P_0,P_1,p_0)$, onde:

   - P é a união de p_0 com P_a para todo $a \in A$;
   - M é a união de M_a com $\{(p_0,p_a)\}$ para todo $a \in A$;
   - P_0 é união de p_0 com $P_{0,a}$ para todo $a \in A$ se, e somente se $\varphi = \forall c\psi$; ou P_0 é união de $P_{0,a}$ para todo $a \in A$ se $\varphi = \exists c\psi$; e
   - P_1 é união de p_0 com $P_{1,a}$ para todo $a \in A$ se, e somente se $\varphi = \exists c\psi$; ou P_1 é união de $P_{1,a}$ para todo $a \in A$ se $\varphi = \forall c\psi$.

Ou seja, $MC(\mathscr{A},\varphi)$ é a união de todos os subjogos $MC(\mathscr{A}_a,\psi)$ para todo $a \in A$, com uma nova posição inicial p_0, mais um movimento $\{(p_0,p_a)\}$ para cada $a \in A$. A posição p_0 é do *falsificador* se $\varphi = \forall c\psi$, ou é do *verificador* se $\varphi = \exists c\psi$.

No exemplo da estrutura $\mathscr{G}^*$, temos que seu universo é $V(G)$. Nesse caso, devemos calcular um jogo para cada vértice de G, e fazer a união de todos esses jogos para obtermos $MC(\mathscr{A},\varphi)$, quando $\varphi = \forall x\psi$ ou $\varphi = \exists y\psi$

4. Se $\varphi \in \{\forall R\psi, \exists R\psi\}$, para algum símbolo de relação R, seja $MC(\mathscr{A}_U,\psi) = (P_U, M_U, P_{0,U}, P_{1,U}, p_U)$ o jogo de avaliação sobre $\mathscr{A}_U$ e ψ, tal que $\mathscr{A}_U$ é uma estrutura expandida de $\mathscr{A}$, isto é, o símbolo U foi adicionado ao vocabulário de $\mathscr{A}$ em $\mathscr{A}_U$, tal que a interpretação de R na estrutura $\mathscr{A}_U$ é igual a U, onde $U \subseteq A$. Então $MC(\mathscr{A},\varphi) = (P, M, P_0, P_1, p_0)$, onde:

   - P é a união de p_0 com P_U para todo $U \subseteq A$;
   - M é a união de M_U com $\{(p_0,p_U)\}$ para todo $U \subseteq A$;
   - P_0 é união de p_0 com $P_{0,U}$ para todo $U \subseteq A$ se, e somente se $\varphi = \forall R\psi$; ou P_0 é união de $P_{0,U}$ para todo $U \subseteq A$ se $\varphi = \exists R\psi$; e
   - P_1 é união de p_0 com $P_{1,U}$ para todo $U \subseteq A$ se, e somente se $\varphi = \exists R\psi$; ou P_1 é união de $P_{1,U}$ para todo $U \subseteq A$ se $\varphi = \forall R\psi$.

   Ou seja, $MC(\mathscr{A},\varphi)$ é a união de todos os subjogos $MC(\mathscr{A}_U,\psi)$ para todo $U \subseteq A$, com uma nova posição inicial p_0, mais um movimento $\{(p_0,p_U)\}$ para cada $U \subseteq A$. E a posição p_0 é do *falsificador* se $\varphi = \forall R\psi$, ou é do *verificador* se $\varphi = \exists R\psi$.

   Por exemplo, para fórmula $Disconn(E)$ da Seção 2.1 devemos calcular um jogo para cada $X \subseteq A$, e fazer a união de todos esses jogos para obtermos $MC(\mathscr{A},\varphi)$, quando $\varphi = \exists X\psi$

A Figura 2.5 é uma árvore que representa o jogo de avaliação para a fórmula $cd(S)$ da Figura 2.3, com $S = \{a,b\}$, sobre o grafo G_2 da Figura 2.4. Cada círculo representa o Jogador 0, ou *falsificador*, e cada caixa retangular representa o Jogador 1, ou *verificador*. Um *nó* da árvore é uma posição $p_i \in P$ do jogo. O *nó* inicial $\forall x$ pertence ao *falsificador*, de acordo com a definição. Como G possui 3 vértices, então existem 3 possíveis valores para x, ou seja, existem 3 movimentos possíveis para o *falsificador* jogar. Mas qualquer um dos movimentos passam a jogada para o *verificador*, que por sua vez é um operador lógico $\vee$, possibilitando 2 movimentos para o *verificador*. Toda vez que uma folha da árvore é do *falsificador*, a expressão φ é verdadeira, e quando a folha

é do *verificador*, a expressão φ é falsa. Portanto, no próximo passo, se o *verificador* escolher uma folha que seja do *falsificador*, o *verificador* ganha o jogo, mas se escolher uma folha que seja também do *verificador*, ele perde o jogo.

Dizemos que um jogador tem uma estratégia vencedora sobre $\mathscr{G}$ se, e somente se, ele sempre ganha todas as partidas, independente das escolhas do jogador oponente. Por exemplo, o *falsificador* tem uma estratégia vencedora sobre $\mathscr{G}$ se, e somente se:

- A posição p_0 pertence ao *falsificador* e existe um movimento $(p_0, p_1) \in M$ tal que o *falsificador* tem uma estratégia vencedora sobre o $subjogo_{\mathscr{G}}(p_1)$; ou
- A posição p_0 pertence ao *verificador* e o *falsificador* tem uma estratégia vencedora sobre o $subjogo_{\mathscr{G}}(p_1)$, para todos os movimentos $(p_0, p_1) \in M$ (incluindo o caso em que o *verificador* não pode mover-se).

Os *nós* na árvore da Figura 2.5, possuem um símbolo ☺, isto é, aquele jogador possui uma estratégia vencedora a partir da posição atual, ou um símbolo ☹, isto é, aquele jogador não possui estratégia vencedora a partir da posição atual. No exemplo do jogo da Figura 2.5, o *falsificador*, que é o jogador que inicia a partida, perde o jogo, pois a próxima jogada é do *verificador*, que possui uma estratégia vencedora para todas as jogadas a partir da posição atual.

O Algoritmo 2.1, define qual jogador possui uma estratégia vencedora. O laço da linha 3 a 6 percorre todas as posições do jogo até chegar nas posições que não possuem movimento. A partir daí, o algoritmo decide se o jogador da posição atual possui ou não uma estratégia vencedora. Isso acontece no caminho de volta da recursão. Nas linhas 7 a 14 o algoritmo decide se o *falsificador* tem uma estratégia vencedora, enquanto nas linhas 15 a 22 essa decisão é para o *verificador*. Além disso, a linha 12 é executada toda vez que o *falsificador* tem estratégia vencedora, enquanto a linha 20 é executada toda vez que o *verificador* tem estratégia vencedora.

Para tornar clara a construção da estratégia vencedora vamos analisar o jogo da Figura 2.5. Partindo das folhas até a raiz, temos que cada folha é uma fórmula atômica ou negada, logo se tal fórmula pertencer ao *falsificador*, então o *verificador* tem uma estratégia vencedora uma vez que o *falsificador* não possui movimentos, caso contrário, se pertencer ao *verificador*, então o *falsificador* tem uma estratégia vencedora uma vez que o *verificador* não possui movimentos. A partir daí, subindo no nível anterior da árvore podemos avaliar o *nó* pai, que possui uma estratégia vencedora se:

- existe um movimento para uma posição que pertence ao mesmo jogador, na qual possui estratégia vencedora;

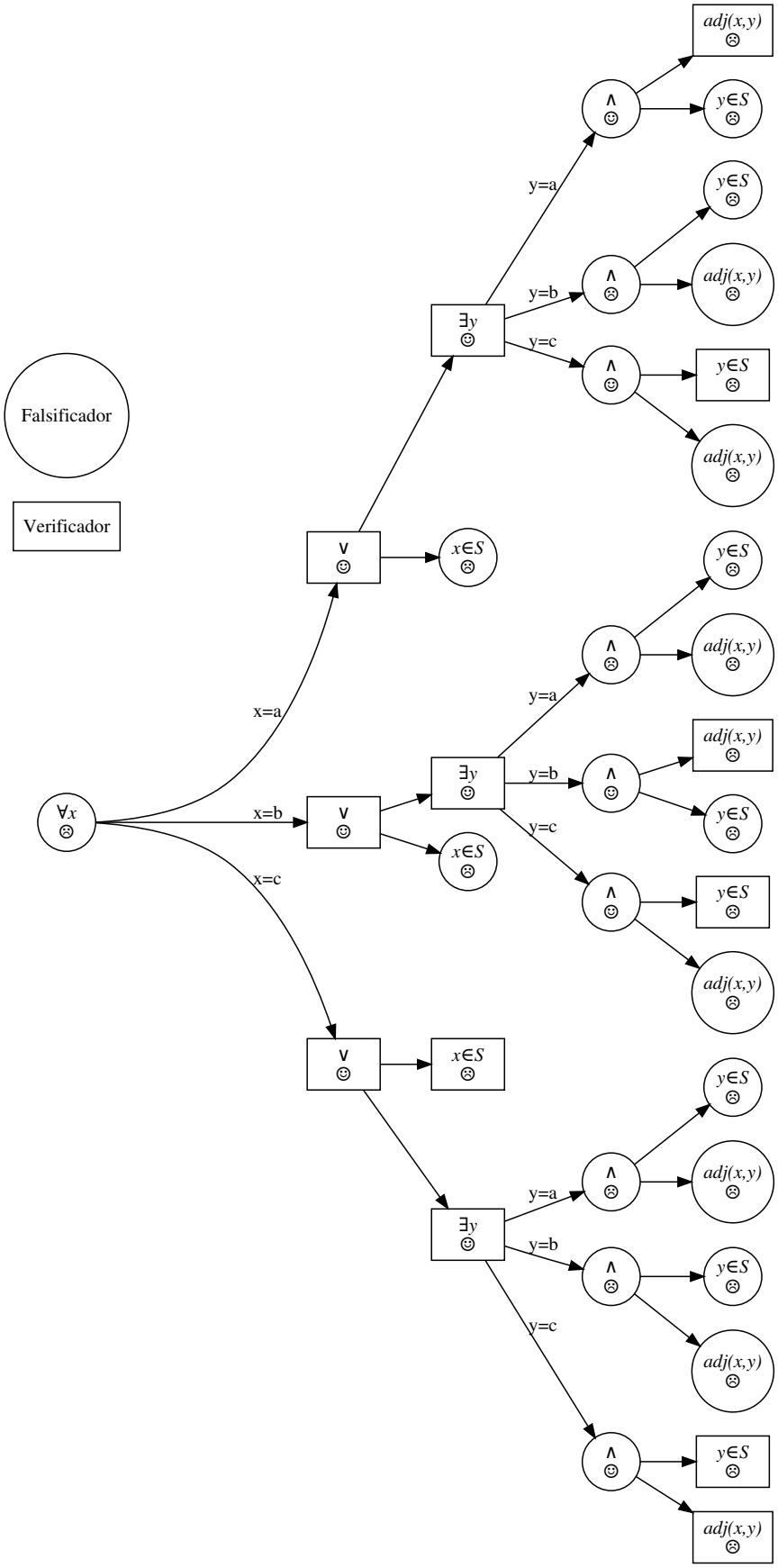


Figura 2.5: Jogo de avaliação da fórmula $cd(S)$.

- existe um movimento para uma posição que pertence ao oponente, o qual

não possui estratégia vencedora.

Note que, determinar se o jogo possui uma estratégia vencedora não depende da compreensão de como o jogo foi gerado, porém depende das posições e dos movimentos do jogo.

Algoritmo 2.1: $avalia(\mathscr{G})$

Entrada: Um jogo $\mathscr{G} = (P, M, P_0, P_1, p_0)$.
Saída: V (Verdadeiro) ou F (Falso).

```
1  início
2      Subjogos ← ∅
3      para cada p' ∈ próximo(p_0) faça
4          Resultado ← avalia(subjogo_𝒢(p'))
5          Subjogos ← Subjogos ∪ {Resultado}
6      fim
7      se p_0 pertence ao falsificador então
8          se todo elemento de Subjogos = V ou Subjogos = ∅ então
9              retorna V
10         fim
11         se F ∈ Subjogos então
12             retorna F
13         fim
14     fim
15     senão se p_0 pertence ao verificador então
16         se todo elemento de Subjogos = F ou Subjogos = ∅ então
17             retorna F
18         fim
19         se V ∈ Subjogos então
20             retorna V
21         fim
22     fim
23 fim
```

CAPÍTULO

3

Extensão da Abordagem para Checar expressão MSO

A solução apresentada no capítulo anterior resolve problemas de decisão em grafos. Como foi visto, de acordo com o tamanho do grafo, construir o jogo para a fórmula *MSO* considerando o grafo dado como entrada custa tempo exponencial. Uma solução menos custosa é apresentada em [12]. Essa solução exige como entrada, a decomposição em árvore do grafo.

O modelo de verificação da fórmula *MSO* é estendido para essa nova solução. A Seção 3.1 trata essa extensão.

Vamos apresentar uma nova versão do Algoritmo 2.1, que avalia um jogo *MC* e define qual jogador possui uma estratégia vencedora. Essa versão poderá apresentar uma terceira avaliação que pode ser chamada de *empate* nos passos intermediários do algoritmo. Quando ocorre um empate, nenhum jogador possui estratégia vencedora e o jogo não pode ser determinado.

Nas seções seguintes, serão apresentados uma série de algoritmos, definições e lemas para a compreensão da nova solução.

3.1 O Modelo Estendido para Verificar a Fórmula **MSO**

A nova solução para verificar se uma fórmula *MSO* é válida para um certo grafo é apresentada como uma extensão do modelo anterior. Agora vamos conhecer o modelo estendido de verificação da fórmula *MSO*. Esse modelo adiciona um novo elemento para o qual um símbolo nulário pode ser mapeado. Esse elemento é um valor *nulo*. Isso, na verdade, significa que o símbolo nulário não foi interpretado ainda, mas poderá ser interpretado no futuro. Em [12]

foi definido um algoritmo que computa a solução utilizando o modelo estendido, através da programação dinâmica. A entrada para o algoritmo é o grafo, a fórmula *MSO* e uma decomposição em árvore do grafo e essa decomposição deve ser do tipo *nice* (veja a Definição 9, a seguir). O algoritmo percorre a decomposição em árvore, a partir das folhas até a raiz, através da programação dinâmica. Os jogos estendidos são construídos sobre o grafo induzido em cada *bag* da decomposição. Como as *bags* têm tamanho pequeno, o universo da estrutura dos jogos construídos são menores do que o universo do grafo original. Sendo assim, a solução é computada parcialmente utilizando a técnica da programação dinâmica. Mais adiante mostraremos que se a solução for encontrada em uma parte do grafo, então essa solução vale para o grafo inteiro. A razão de estender o modelo é possibilitar que o vocabulário da estrutura possa ser interpretado parcialmente nas *bags* da decomposição, antes de chegar na raíz.

Note que essa abordagem constrói jogos menores que o jogo *MC* sempre que as *bags* forem menores que o número de vértices do grafo. E os jogos podem ser determinados sobre uma parte do grafo, antes mesmo de considerar o grafo inteiro.

Na decomposição *nice*, as *bags* mãe e filha diferem em um elemento. Isso é uma vantagem, pois os jogos da *bag* filha, já computados, podem ser úteis ao computar o jogos da *bag* mãe. Basta "remover" ou "adicionar" um elemento, ou combinar jogos da mesma *bag*.

Agora podemos definir formalmente as duas propriedades principais do modelo estendido de verificação:

1. Este modelo está definido para estruturas que interpretam somente uma parte do vocabulário; e

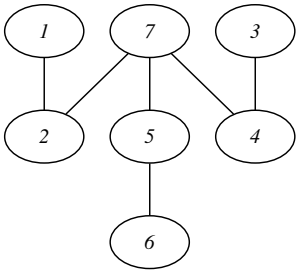
2. Está definido também para a união de estruturas, ou seja, se um jogador tem uma estratégia vencedora no jogo sobre a estrutura $\mathscr{A}$ e φ, então o mesmo jogador tem uma estratégia vencedora no jogo sobre $\mathscr{A} \cup \mathscr{B}$ e φ, para toda estrutura $\mathscr{B}$ compatível com $\mathscr{A}$ (veja a Definição 6);

Definição 9. *(Decomposição em árvore nice).* Uma decomposição em árvore *nice* deve satisfazer as seguintes propriedades:

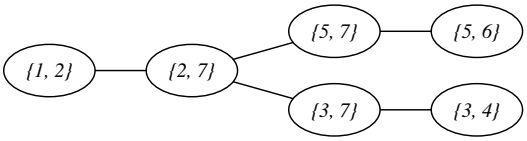
- A árvore é dirigida a partir da raiz;
- Todo nó da árvore tem no máximo dois filhos;
- Cada nó folha é uma *bag* que contém um único elemento;
- Se um nó i tem um único filho j então a diferença entre as *bags* X_i e X_j é de um único elemento. Quando a *bag* X_i for maior do que X_j, dizemos

que i é um nó do tipo *introduce*, caso contrário, se a *bag* X_i for menor do que X_j, dizemos que i é um nó do tipo *forget*.

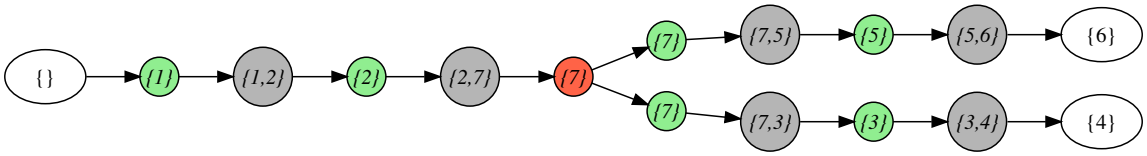
- Se um nó i tem exatamente dois filhos, j_1 e j_2, é necessário que as *bags* X_i, X_{j_1} e X_{j_2} sejam iguais, e dizemos que o nó i é do tipo *join*.



(a) Grafo G



(b) Decomposição em árvore de G.



(c) Decomposição *Nice* de G.

Figura 3.1: Exemplo de Decomposição em Árvore *Nice*.

A Figura 3.1(c) é uma decomposição em árvore *nice* do grafo G da Figura 3.1(a). As *bags* de cor cinza são do tipo *introduce*, as de cor verde são do tipo *forget* e as de cor vermelha são do tipo *join*. A decomposição em árvore da Figura 3.1(b) possui *treewidth*$= 1$. Note que as propriedades de uma decomposição em árvore tradicional são mantidas na decomposição *nice*. Nas próximas seções, veremos que as propriedades exigidas pela decomposição *nice* são fundamentais para a computação do algoritmo da Seção 3.2.

Definimos agora o modelo de verificação estendido. Cada jogo deste modelo está relacionado a uma *bag* X_i da decomposição. Portanto o jogo depende do conjunto $X = X_i \subseteq A$ que é mantido como uma informação adicional nas posições do jogo estendido $EMC(\mathscr{A}, X, \varphi)$.

Definição 10. (Jogo de Avaliação Estendido [12]). O *Jogo de Avaliação Estendido* $EMC(\mathscr{A}, X, \varphi) = (P, M, P_0, P_1, p_0)$ sobre uma τ-estrutura $\mathscr{A}$, um conjunto $X \subseteq A$ e uma fórmula $\varphi \in MSO(\tau)$ é definido por indução sobre a estrutura da fórmula φ como segue. Seja $p_0 = (\mathscr{A}[X \cup \bar{c}^{\mathscr{A}}], X, \varphi)$, onde $\bar{c} = nulário(\tau)$.

1. Se φ é uma fórmula atômica ou atômica negada, então $EMC(\mathscr{A}, X, \varphi) = (\{p_0\}, \emptyset, P_0, P_1, p_0)$, onde:

   - A posição inicial p_0 pertence ao *falsificador* se, e somente se:

– a fórmula φ é atômica do tipo $(c_1,...,c_p) \in R$, tal que $\{c_1,...,c_p\}$ são interpretados em $\mathscr{A}$ e as interpretações de $(c_1,...,c_p)$ estão contidas na interpretação de R; ou

– a fórmula φ é negada do tipo $(c_1,...,c_p) \notin R$, tal que $\{c_1,...,c_p\}$ são interpretados em $\mathscr{A}$ e as interpretações de $(c_1,...,c_p)$ não estão contidas na interpretação de R. Nesses casos $P_1 = \emptyset$.

- A posição inicial p_0 pertence ao *verificador* se, e somente se:

  – a fórmula φ é atômica do tipo $(c_1,...,c_p) \in R$, tal que $\{c_1,...,c_p\}$ são interpretados em $\mathscr{A}$ e as interpretações de $(c_1,...,c_p)$ não estão contidas na interpretação de R; ou

  – a fórmula φ é negada do tipo $(c_1,...,c_p) \notin R$, tal que $\{c_1,...,c_p\}$ são interpretados em $\mathscr{A}$ e as interpretações de $(c_1,...,c_p)$ estão contidas na interpretação de R.
  Nesses casos $P_0 = \emptyset$.

2. Se $\varphi \in \{\forall R\psi, \exists R\psi\}$, para algum símbolo relacional R, ou se $\varphi \in \{\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2\}$, então $EMC(\mathscr{A}, X, \varphi)$ é definido analogamente à $MC(\mathscr{A}, \varphi)$.

3. Se $\varphi \in \{\forall c\psi, \exists c\psi\}$, para algum símbolo nulário c, seja $\mathscr{A}_u = (\mathscr{A}, u)$ a (τ, c)-expansão de $\mathscr{A}$ com $c^{\mathscr{A}_u} = u \in A$ ou $c^{\mathscr{A}_u} = nulo$, e $EMC(\mathscr{A}_u, X, \psi) = (P_u,\ M_u,\ P_{0,u},\ P_{1,u},\ p_u)$ o jogo de avaliação estendido sobre $\mathscr{A}_u$ e ψ, tal que $\mathscr{A}_u$ é uma estrutura expandida de $\mathscr{A}$, isto é, o símbolo c foi adicionado ao vocabulário de $\mathscr{A}$ em $\mathscr{A}_u$, tal que a interpretação de c na estrutura $\mathscr{A}_u$ é igual a u, onde $u \in A$. Então $EMC(\mathscr{A}, X, \varphi) = (P, M, P_0, P_1, p_0)$, sendo que:

   - P é a união de p_0 com P_u para todo $u \in A \cup \{nulo\}$;
   - M é a união de M_u com $\{(p_0, p_u)\}$ para todo $u \in A \cup \{nulo\}$;
   - P_0 é união de p_0 com $P_{0,u}$ para todo $u \in A \cup \{nulo\}$ se, e somente se $\varphi = \forall c\psi$ ou P_0 é a união de $P_{0,u}$ para todo $u \in A \cup \{nulo\}$ se $\varphi = \exists c\psi$;
   - P_1 é união de p_0 com $P_{1,u}$ para todo $u \in A \cup \{nulo\}$ se, e somente se $\varphi = \exists c\psi$ ou P_1 é a união de $P_{1,u}$ para todo $u \in A \cup \{nulo\}$ se $\varphi = \forall c\psi$.

Se o jogo termina em uma posição $p_j \notin P_0 \cup P_1$, então houve um empate entre os jogadores e ninguém ganha o jogo.

Dizemos que o jogo é *determinado* se um dos jogadores possui uma estratégia vencedora, caso contrário o jogo é *não determinado*. Um jogo estendido é *determinado* e o *verificador* tem uma estratégia vencedora se, e somente se, $\mathscr{A} \models \varphi$.

3.2 Visão Simplificada do Algoritmo

Apresentamos aqui uma versão do algoritmo que utiliza a decomposição em árvore *nice* do grafo, para decidir se uma fórmula $\varphi \in MSO$ é válida em um dado grafo. As rotinas, *reduza*(), *combine*() e *remove*() utilizadas no algoritmo serão explicadas nas próximas seções, a princípio devemos saber que cada rotina retorna um jogo. Esta seção tem o objetivo de motivar o uso de tais rotinas combinadas com a decomposição em árvore *nice*.

Seja $(\mathscr{A},U)$ uma $(\tau,U)-$expansão de $\mathscr{A}$, para U um símbolo relacional livre e $U^{\mathscr{A}} \in A$. O algoritmo verifica se $(\mathscr{A},U) \models \varphi$, com $\varphi \in MSO(\tau)$.

Assumimos que $X_{raiz} = \emptyset$, para cada nó i da decomposição, $\mathscr{A}_i$ é uma subestrutura de $\mathscr{A}$ induzida pelos elementos da *bag* X_i e que, $\mathscr{A}_i(U) = (\mathscr{A}_i,U)[X_i]$.

O algoritmo constrói jogos $\mathscr{R}(U)$, que são jogos considerando o subconjunto U, sobre o subgrafo (ou $\tau-$estrutura) induzido pela *bag* X_i.

Em cada *bag* X_i é armazenado um conjunto $S_i(U)$ que contém o jogo sobre o conjunto U, se esse jogo não for $\bot$.

Os símbolos $\top$ ou $\bot$ representam um jogo verdadeiro ou falso, respectivamente. Essa representação é a mesma adotada em [7].

1ª fase. O algoritmo atravessa a decomposição *nice* a partir das folhas até a raiz. Cada nó i da decomposição é uma folha ou um dos tipos *introduce*, *forget* ou *join*.

folha: Sejam $X_i = \{x\}$ e $U \leftarrow U \cap X_i$.

$$\mathscr{R}(U) = reduza(\mathscr{A}_i(U), X_i, \varphi).$$

Se $\mathscr{R}(U) \neq \bot$, então $S_i(U) \leftarrow \{\mathscr{R}(U)\}$.

introduce: Seja j o filho único de i e $X_i = X_j \cup \{x\}$, para $x \notin A_j$. Se existir um jogo $\mathscr{R}_j \in S_j(U_j)$, então para $U_i = U \cap X_i$, tal que $U_j = U_i \cap X_j$ faça,

$$\mathscr{R}_i = \begin{cases} \top & \text{se } \mathscr{R}_j = \top \text{ , e} \\ combine(\mathscr{R}_j, \mathscr{R}(U_i)) & \text{senão.} \end{cases}$$

Se $\mathscr{R}_i \neq \bot$, então $S_i(U_i) \leftarrow \{\mathscr{R}_i\}$.

forget: Seja j o filho único de i e $X_j = X_i \cup \{x\}$, para $x \notin A_i$. Se existir um jogo $\mathscr{R}_j \in S_j(U_j)$, então seja $U_i = U_j \cap X_i$ faça,

$$\mathscr{R}_i = \begin{cases} \top & \text{se } \mathscr{R}_j = \top \text{ , e} \\ remove(\mathscr{R}_j, x) & \text{senão.} \end{cases}$$

Se $\mathscr{R}_i \neq \bot$, então $S_i(U_i) \leftarrow \{\mathscr{R}_i\}$.

join: Sejam j_1 e j_2 , filhos de i e $X_{j_1} = X_{j_2} = X_i$. Seja $U \leftarrow U \cap Xi$. Se existirem jogos $(\mathscr{R}_{j_1}, \mathscr{R}_{j_2}) \in S_{j_1}(U) \times S_{j_2}(U)$, faça

$$\mathscr{R}_i = \begin{cases} \top & \text{se } \mathscr{R}_{j_1} = \top \text{ ou } \mathscr{R}_{j_2} = \top \text{ , e} \\ combine(\mathscr{R}_{j_1}, \mathscr{R}_{j_2}) & \text{senão.} \end{cases}$$

Se $\mathscr{R}_i \neq \bot$, então $S_i(U) \leftarrow \{\mathscr{R}_i\}$.

2ª fase. Seja r a raíz da decomposição.

Se existir um jogo $\mathscr{R}_r \in S_r(U)$, então

$$S_r(U) = avalia(converte(\mathscr{R}_r)),$$

caso contrário $S_r(U) = \{\bot\}$.

Descrição do Algoritmo. Primeiramente, um jogo $\mathscr{R}(U)$ é construído na folha e esse jogo é guardado para ser usado na próxima *bag* a ser alcançada na decomposição.

Quando o nó é uma *bag* X_i do tipo *introduce* ou *forget* o algoritmo analisa o jogo construído no subgrafo induzido pela *bag* X_j que é filha de X_i. Se o verificador tiver uma estratégia vencedora nesse jogo, então ele também terá uma estratégia vencedora no jogo do subgrafo induzido pela *bag* X_i.

Por outro lado, se o jogo da *bag* X_j não for determinado, devemos analisar cada caso separadamente:

- 1º caso: Se X_i for do tipo *introduce*, um novo jogo é construído sobre o subgrafo induzido na bag X_i e combinado com o jogo de X_j. Combinar jogos significa unir os jogos em um só, representando agora o jogo da *bag* X_i, que a grosso modo, foi expandido ao ganhar um novo elemento x que está contido no subgrafo de X_i, mas não está no subgrafo de X_j.

- 2º caso: Se X_i for do tipo *forget*, então x será removido da *bag*, mas será mantido no jogo em alguns casos específicos (como veremos na Seção 3.7) . As propriedades da decomposição em árvore garantem que o elemento x não existirá nas *bags* dos nós acima de i na decomposição. Remover x do jogo permite a redução do tamanho do jogo, o que melhora o desempenho do algoritmo.

Se o nó for uma *bag* X_i do tipo *join*, o algoritmo analisa os jogos das filhas de X_i. Aqui, a ideia é a mesma utilizada no *introduce*, se uma das *bags* filhas de X_i tiver um jogo, tal que o verificador tenha uma estratégia vencedora, então

esse mesmo jogador terá uma estratégia vencedora no jogo da *bag* X_i. Caso contrário, os jogos devem ser combinados, e nesse caso, o jogo da *bag* X_i é equivalente ao jogo do subgrafo induzido nas *bags* abaixo de i na decomposição. Note que nunca existirá um jogo $\top$ e outro $\bot$ entre os filhos de X_i, pois nesse caso, a fórmula φ seria verdadeira e falsa, o que é uma contradição.

Note que nunca é construído um novo jogo equivalente ao jogo do subgrafo induzido nas *bags* abaixo de X_i na decomposição. Ao invés disso, o jogo da *bag* X_j é sempre reutilizado na *bag* X_i.

Quando o algoritmo alcança a raiz r da decomposição, se o jogo $R_r \notin \{\top, \bot\}$, então ele é convertido para um jogo MC e determinado pelo Algoritmo 2.1. Se $S_r(U) = \top$, então $(\mathscr{A}, U)$ é verdadeira para φ, caso contrário é falsa.

Na Seção 2.1 foi apresentada a fórmula para o problema da $3-$Coloração. Lembramos que um grafo é $3-$colorível se for possível dividir seu conjunto de vértices em 3 partes, tal que não existam arestas entre os vértices da mesma parte. Note que a fórmula desse problema não necessita de uma variável relacional livre. Ainda assim, o algoritmo acima verifica essa fórmula assumindo que $U = \{\}$.

3.3 Converter um Jogo **EMC** em **MC**

Quando o algoritmo processa todas as *bags* da decomposição e atinge a raiz, então se houver um jogo na raiz, esse jogo pode estar determinado ou não e, além disso, é equivalente ao jogo construído para todo o grafo. Se o jogo ainda não estiver determinado, então ele deve ser convertido para o modelo tradicional MC, para que possa ser avaliado como $\top$ ou $\bot$. Nesta seção, vamos entender como um jogo EMC é convertido para um jogo MC.

Se $\mathscr{A}$ é uma τ-estrutura, então o jogo $MC(\mathscr{A}, \varphi)$ está inserido dentro do jogo $EMC(\mathscr{A}, X, \varphi)$. Note que os dois jogos são distintos, pois o jogo $EMC(\mathscr{A}, X, \varphi)$ possui algumas posições em que os símbolos nulários não são interpretados e o jogo $MC(\mathscr{A}, \varphi)$ não as possui. O Algoritmo 3.1 transforma um jogo $EMC(\mathscr{A}, X, \varphi)$ em um jogo $MC(\mathscr{A}, \varphi)$ e o Lema 1 mostra que um jogo $MC(\mathscr{A}, \varphi)$ é igual ao jogo $EMC(\mathscr{A}, X, \varphi)$ convertido, através da execução desse algoritmo.

Lema 1. ([12]) Seja $\mathscr{A}$ uma τ-estrutura que interpreta totamente τ, $X \subseteq A$, e $\varphi \in MSO(\tau)$. Então, usando o Algoritmo 3.1, temos que,

$$MC(\mathscr{A}, \varphi) = converte(EMC(\mathscr{A}, X, \varphi)).$$

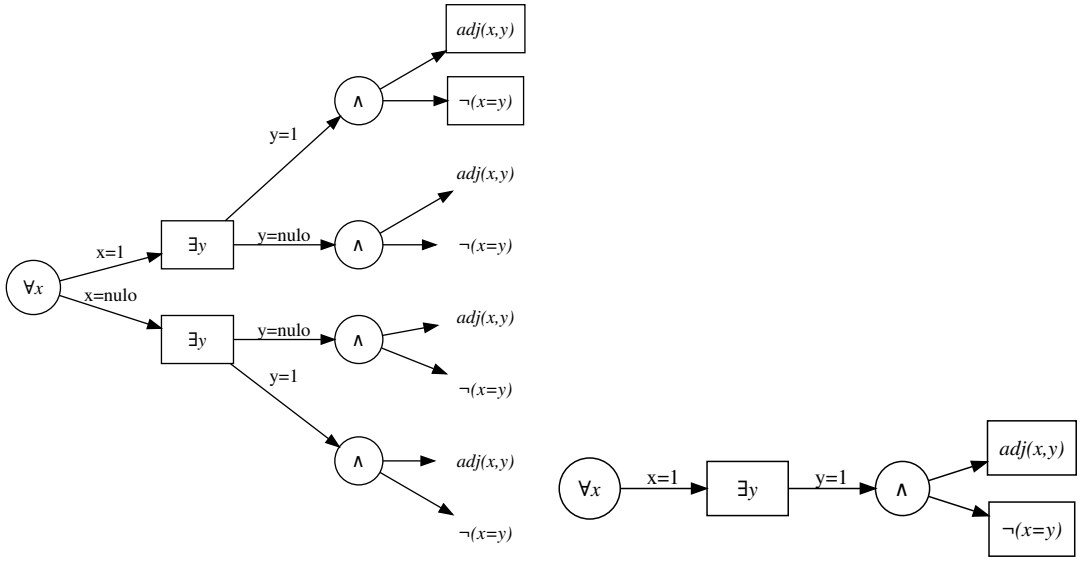
O Algoritmo 3.1, constrói o jogo $\mathscr{G}_1$, isto é, uma cópia do jogo EMC eliminando todas as ramificações que partem da expansão de um símbolo nulário

Algoritmo 3.1: $converte(\mathscr{G})$

Entrada: Um jogo EMC $\mathscr{G} = (P, M, P_0, P_1, p_0)$, com $p_0 = (\mathscr{H}, X, \varphi)$.
Saída: Um jogo $MC(\mathscr{G}_1)$.

1 **início**
2 **se** $\mathscr{G}$ *está determinado* **então**
3 **retorna** $\mathscr{G}$
4 **fim**
5 Seja $\bar{c}$ o conjunto de símbolos nulários em $\mathscr{H}$;
6 Seja $p'_0 = (\mathscr{H}[\bar{c}^{\mathscr{H}}], \varphi)$, ou seja, o universo H de $\mathscr{H}$ são as interpretações dos símbolos nulários;
7 Construa um jogo $\mathscr{G}_1$, tal que $p'_0 \in \mathscr{G}_1$;
8 **para cada** $p_1 = (\mathscr{H}_1, X, \psi) \in$ *próximo*(p_0), *tal que* $\mathscr{H}_1$ *interpreta totalmente seu vocabulário* **faça**
9 $\mathscr{G}' = converte(subjogo_{\mathscr{G}}(p_1))$
10 $\mathscr{G}_1 \leftarrow \mathscr{G}_1 \cup \mathscr{G}'$
11 Adicione em $\mathscr{G}_1$ um movimento de p_0 para $\mathscr{G}_1$
12 **fim**
13 **retorna** $\mathscr{G}_1$
14 **fim**

não interpretado. Dessa forma, o jogo construído a partir da cópia de *EMC* é um jogo *MC*. O laço das linhas $8-12$ atualiza o jogo $\mathscr{G}_1$ com as posições que interpretam totalmente seu vocabulário. As posições que possuem algum símbolo não interpretado não estão contidas em $\mathscr{G}_1$.



(a) Jogo $\mathscr{G}' = EMC(\mathscr{G}^*, \emptyset, \varphi)$ (b) Jogo $EMC(\mathscr{G}^*, \emptyset, \varphi)$ convertido.

Figura 3.2: Um exemplo de entrada e saída do Algoritmo 3.1.

A Figura 3.2(a) ilustra um jogo $EMC(\mathscr{G}^*, \emptyset, \varphi)$ para a estrutura $\mathscr{G}^*$ de um grafo $G = (\{1\}, \emptyset)$ com universo $G^* = \{1\}$ e $\varphi = \forall x(\exists y(adj(x,y) \wedge \neg(x = y)))$. A Figura 3.2(b) ilustra o resultado do algoritmo $converte(\mathscr{G}')$ para $\mathscr{G}' = EMC(\mathscr{G}^*, \emptyset, \varphi)$.

3.4 Função que Avalia um Jogo Estendido

A partir desta seção, para uma posição p de um jogo, serão utilizados os termos *ramo (ou ramo que inicia em p)* e *ramificação (ou ramificação que inicia em p)* como um subjogo que inicia em uma posição p e vai até as suas folhas ou o subjogo considerado a partir da interpretação de um novo símbolo expandido. Por exemplo, na Figura 3.2(a) a ramificação (ou ramo) que parte de $x = 1$ é o subjogo que inicia na posição em que a fórmula é $\exists y$, tal que x é interpretado para 1.

No início do capítulo, mencionamos que uma nova versão do algoritmo que avalia um jogo *MC* seria apresentada para que possa avaliar também jogos *EMC*. O Algoritmo 2.1 considera somente jogos sobre as estruturas em que todos os símbolos nulários são interpretados. Por isso, seu retorno é sempre Verdadeiro ou Falso.

Para avaliar jogos estendidos, em que alguns símbolos do vocabulário não possuem interpretações, é necessário tratar o caso em que o jogo não pode ser definido, por existirem símbolos que ainda podem ser interpretados no futuro. O Algoritmo 3.2 é a nova versão que avalia um jogo *EMC*.

Nesse algoritmo, à medida que o jogo é avaliado, um novo jogo $\mathscr{G}_1$ é construído, conforme a linha 5. Se uma das condições descritas nas linhas 14, 17, 22 e 25 for satisfeita, um jogo determinado será retornado. Caso contrário, o jogo construído que está indeterminado será retornado.

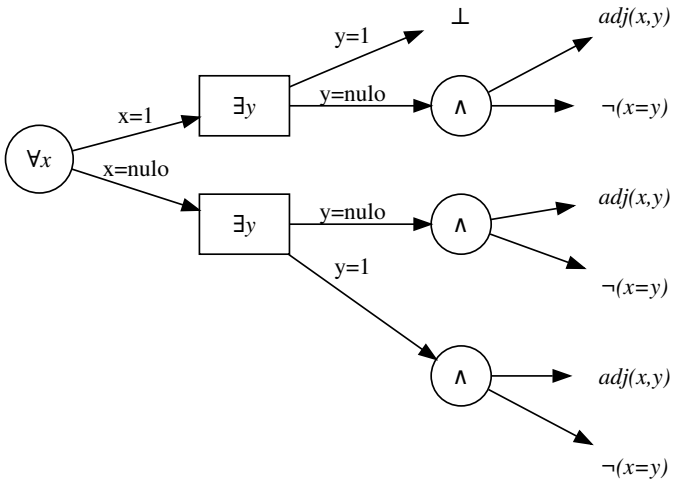


Figura 3.3: Jogo $\mathscr{G}_1 = avaliaEMC(\mathscr{G}')$ para o jogo $\mathscr{G}'$ da Figura 3.2(a).

A Figura 3.3 representa o jogo $\mathscr{G}_1$ retornado pelo Algoritmo 3.2 que recebeu como entrada o jogo $\mathscr{G}' = EMC(\mathscr{G}^*)$ da Figura 3.2(a). Note que o subjogo que está na expansão de $\forall x(x = 1)$ e $\exists y(y = 1)$ é falso, enquanto os outros subjogos não foram determinados, pois em alguns deles, x, y ou ambos não foram interpretados.

Algoritmo 3.2: $avaliaEMC(\mathscr{G})$

Entrada: Um jogo $\mathscr{G} = (P, M, P_0, P_1, p_0)$
Saída: $\top$, $\bot$ ou $EMC(\mathscr{G}_1)$

```
1  início
2      se 𝒢 está determinado então
3          retorna 𝒢
4      fim
5      Construa um jogo 𝒢_1, tal que p_0 ∈ 𝒢_1
6      Subjogos ← ∅
7      para cada p ∈ próximo(p_0) faça
8          𝒢' ← avaliaEMC(subjogo_𝒢(p))
9          𝒢_1 ← 𝒢_1 ∪ 𝒢'
10         Adicione um movimento de p_0 para 𝒢' em 𝒢_1
11         Subjogos ← Subjogos ∪ 𝒢'
12     fim
13     se p_0 pertence ao falsificador então
14         se todo elemento de Subjogos = ⊤ ou Subjogos = ∅ então
15             retorna ⊤
16         fim
17         se ⊥ ∈ Subjogos então
18             retorna ⊥
19         fim
20     fim
21     senão se p_0 pertence ao verificador então
22         se todo elemento de Subjogos = ⊥ ou Subjogos = ∅ então
23             retorna ⊥
24         fim
25         se ⊤ ∈ Subjogos então
26             retorna ⊤
27         fim
28     fim
29     retorna 𝒢_1
30 fim
```

O Lema 2 mostra que se um jogo estendido for determinado, então o jogador vencedor de tal jogo pode vencer o jogo sem utilizar as posições dos ramos construídos por expansões de símbolos nulários não interpretados.

Lema 2. ([12]) Sejam $\mathscr{A}_1$ e $\mathscr{A}_2$ τ-estruturas com $A_1 = A_2$ e c um símbolo nulário, tais que c não é interpretado em $\mathscr{A}_1$, as interpretações dos símbolos relacionais R de τ são iguais em $\mathscr{A}_1$ e $\mathscr{A}_2$ e, para todo símbolo nulário $d \in nulário(\tau)\backslash\{c\}$, as interpretações de d são iguais em $\mathscr{A}_1$ e $\mathscr{A}_2$. Seja $\varphi \in MSO(\tau)$:

$$\text{Se } avalia(EMC(\mathscr{A}_1, X, \varphi)) \in \{\top, \bot\} \text{ então } A_1 \neq \emptyset \text{ e}$$

$$avalia(EMC(\mathscr{A}_1, X, \varphi)) = avalia(EMC(\mathscr{A}_2, X, \varphi)) \ .$$

Ideia da Prova. Se $\mathscr{G} = EMC(\mathscr{A}_1, X, \varphi) \in \{\top, \bot\}$, então existe um $subjogo_{\mathscr{G}}(p_i)$ que determina $\mathscr{G}$. Lembramos que $subjogo_{\mathscr{G}}(p_i)$ termina em subjogos que são folhas. Note que se p_i é determinado, existe pelo menos uma folha que interpreta todos os seus símbolos nulários, nesse caso os símbolos interpretados são diferentes de c. Portanto o vencedor pode ganhar o jogo sem depender dos jogos onde c ocorre. □

O Lema 3, mostra que se um jogador tem uma estratégia vencedora no jogo estendido *EMC*, então esse mesmo jogador tem uma estratégia vencedora no jogo tradicional *MC*.

Lema 3. ([12]) Seja $\mathscr{A}$ uma τ-estrutura que interpreta totamente τ, $X \subseteq A$, e $\varphi \in MSO(\tau)$. Se $avalia(EMC(\mathscr{A}, X, \varphi)) \in \{\top, \bot\}$, então,

$$avalia(MC(\mathscr{A}, \varphi)) = avalia(EMC(\mathscr{A}, X, \varphi)).$$

Ideia da Prova. O Lema 2 afirma que, se um jogo *EMC* é determinado, então sua avaliação não depende das ramificações onde os símbolos nulários não são interpretados. Por isso, podemos aplicar o Algoritmo 3.1 no jogo $EMC(\mathscr{A}, X, \varphi)$ para remover tais ramificações, pois por hipótese $EMC(\mathscr{A}, X, \varphi)$ é determinado. Assim, temos que

$$avalia(EMC(\mathscr{A}, X, \varphi)) = avalia(converte(EMC(\mathscr{A}, X, \varphi))).$$

Pelo Lema 1, $converte(EMC(\mathscr{A}, X, \varphi)) = MC(\mathscr{A}, \varphi)$, portanto, podemos concluir que $avalia(MC(\mathscr{A}, \varphi)) = avalia(EMC(\mathscr{A}, X, \varphi))$. □

3.5 Jogo Reduzido

Nas próximas duas seções, veremos que algumas funções podem construir jogos que contenham elementos no universo da estrutura e interpretações de símbolos nulários que não estão na *bag* X do jogo.

Sejam x e y dois vértices desta forma. Seja $\mathscr{G}$ um jogo e cada vértice x e y pertencentes a $subjogo_{\mathscr{G}}(p_1)$ e $subjogo_{\mathscr{G}}(p_2)$, respectivamente.

Se pudermos mapear cada $subjogo_{\mathscr{G}}(p_{k1})$ no ramo de $subjogo_{\mathscr{G}}(p_1)$ em um $subjogo_{\mathscr{G}}(p_{k2})$ no ramo de $subjogo_{\mathscr{G}}(p_2)$ de forma que $subjogo_{\mathscr{G}}(p_{k1})$ e $subjogo_{\mathscr{G}}(p_{k2})$ tenham a mesma fórmula ψ e que exista um isomorfismo entre os universos das estruturas de $subjogo_{\mathscr{G}}(p_{k1})$ e $subjogo_{\mathscr{G}}(p_{k2})$ de forma que os elementos contidos em X sejam mapeados para eles mesmos, então $subjogo_{\mathscr{G}}(p_1)$ e $subjogo_{\mathscr{G}}(p_2)$ são equivalentes e podemos eliminar um deles.

Vamos observar os subjogos dos ramos $x = 4$ e $x = 5$ nos jogos $\mathscr{G}$ das Figuras 3.4 e 3.5 construídos sobre um caminho P_6 com $V = \{0,1,2,3,4,5\}$ e

$E = \{(i,j) | j = i+1\}$. Sejam $subjogo_{\mathscr{G}}(p_1)$ o subjogo que interpreta $x = 4$ com $\psi = \vee$ e $A_1 = \{2,3,4\}$, e $subjogo_{\mathscr{G}}(p_2)$ o subjogo que interpreta $x = 5$ com $\psi = \vee$ e $A_2 = \{2,3,5\}$. Sabemos que $X = \{2,3\}$. Note que não existe isomorfismo entre os universos de p_1 e p_2, pois teríamos o seguinte mapeamento $a : A_1 \to A_2$,

$$a(2) = 2$$
$$a(3) = 3$$
$$a(4) = 5$$

mas como a estrutura do jogo é um caminho P_6, temos que $subjogo_{\mathscr{G}}(p_1)$ contém a aresta $adj(3,4)$, mas $subjogo_{\mathscr{G}}(p_2)$ não contém a aresta $adj(3,5)$. Logox $subjogo_{\mathscr{G}}(p_1)$ e $subjogo_{\mathscr{G}}(p_2)$ não são equivalentes.

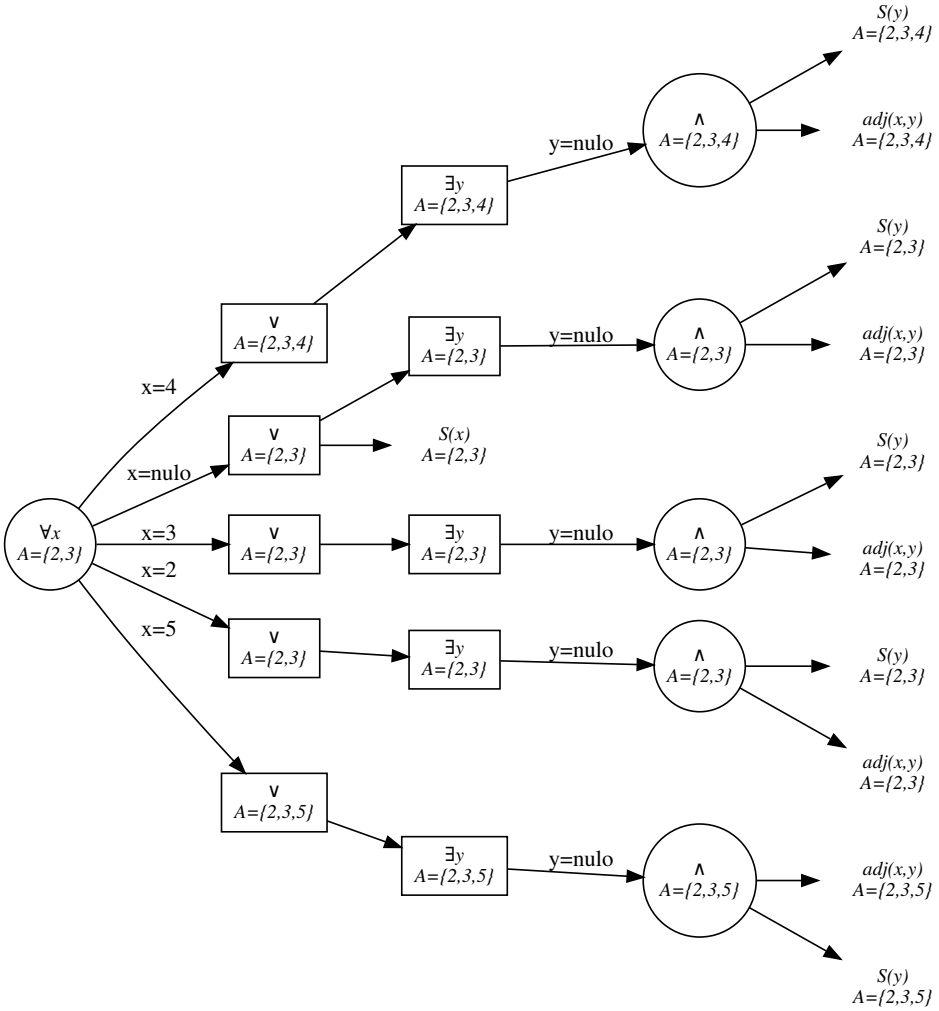


Figura 3.4: Jogo $\mathscr{G}$ cuja estrutura é um caminho com 6 vértices tal que os vértices são rotulados de 0 a 5 de forma que $(i,i+1)$ é uma aresta no caminho. Além disso, $U = \{\}$.

Considere o jogo $\mathscr{G}$ da Figura 3.5. Nesse jogo, o universo de $subjogo_{\mathscr{G}}(p_1)$ é $A_1 = \{2,4\}$ e de $subjogo_{\mathscr{G}}(p_2)$ é $A_2 = \{2,5\}$, para $X = \{2\}$. Note que o mapeamento

$a : A_1 \to A_2$,

$$a(2) = 2$$
$$a(4) = 5$$

é um isomorfismo entre os universos de $subjogo_{\mathscr{G}}(p_1)$ e $subjogo_{\mathscr{G}}(p_2)$. Logo $subjogo_{\mathscr{G}}(p_1)$ e $subjogo_{\mathscr{G}}(p_2)$ são equivalentes e $subjogo_{\mathscr{G}}(p_2)$ foi removido do jogo. Essa remoção é representada pelas arestas pontilhadas.

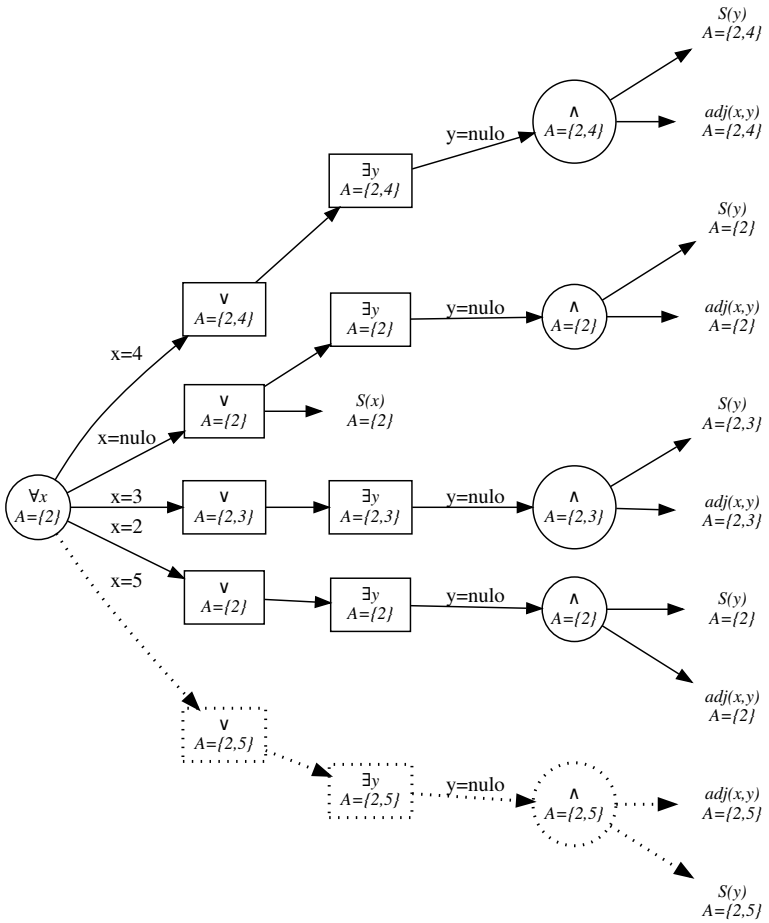


Figura 3.5: Jogo $\mathscr{G}$ cuja estrutura é um caminho com 6 vértices tal que os vértices são rotulados de 0 a 5 de forma que $(i, i+1)$ é uma aresta no caminho. Além disso, $U = \{\}$. Arestas pontilhadas denotam que não pertencem mais ao jogo.

Agora, definiremos a equivalência entre jogos e mostraremos que para todo jogo $\mathscr{G}$ podemos construir um jogo $\mathscr{G}'$ tal que $avalia(\mathscr{G}) = avalia(\mathscr{G}')$ se $avalia(\mathscr{G}) \in \{\top, \bot\}$ e $\mathscr{G}' \subseteq \mathscr{G}$ com $\mathscr{G}'$ potencialmente menor do que $\mathscr{G}$. Ou seja podemos reduzir o tamanho de um jogo estendido eliminando subjogos que são equivalentes a outros jogos. Manter jogos reduzidos é fundamental para o desempenho do algoritmo que valida uma fórmula φ para um determinado grafo.

Definição 11. (Equivalência entre jogos). Duas posições p_1 e p_2 são equivalentes, denotado por $p_1 \cong p_2$ se e somente se,

1. $p_1 = (\mathscr{H}_1, X, \varphi)$ e $p_2 = (\mathscr{H}_2, X, \varphi)$, para alguma fórmula φ e $X \subseteq H_1 \cap H_2$; e
2. existir um isomorfismo $h : H_1 \to H_2$ entre $\mathscr{H}_1$ e $\mathscr{H}_2$, tal que $h(a) = a$ para todo $a \in X$.

Seja $\mathscr{G}$ um jogo arbitrário, chamamos de $subjogos(\mathscr{G})$ o conjunto de todos os subjogos contidos no jogo $\mathscr{G}$.

Dois jogos $\mathscr{G}_1$ e $\mathscr{G}_2$ são equivalentes, denotado por $\mathscr{G}_1 \cong \mathscr{G}_2$ se: $p_1 \cong p_2$, com p_1 é a posição inicial de $\mathscr{G}_1$ e p_2 é a posição inicial de $\mathscr{G}_2$ e existir uma bijeção $\pi : subjogos(\mathscr{G}_1) \to subjogos(\mathscr{G}_2)$ tal que $\mathscr{G}' \cong \pi(\mathscr{G}')$ para todo $\mathscr{G}' \in subjogos(\mathscr{G}_1)$.

Ou seja, os itens 1 e 2 mostram a definição para que duas posições sejam equivalentes. No item 1, p_1 e p_2 tem mesma fórmula φ e o mesmo X, tal que $X \subseteq H_1$ e $X \subseteq H_2$. O item 2 afirma que, deve haver um isomorfismo h entre os universos das estruturas de p_1 e p_2, mas para todo elemento a do universo que estiver contido em X, o mapeamento de a em h deve ser uma identidade. E por fim, para que dois jogos $\mathscr{G}_1$ e $\mathscr{G}_2$ sejam equivalentes, deve haver uma bijeção π entre cada subjogo de $\mathscr{G}_1$ e $\mathscr{G}_2$, tal que para cada $subjogo_{\mathscr{G}_1}(p'_1) \in subjogos(\mathscr{G}_1)$ e $subjogo_{\mathscr{G}_2}(p'_2) \in subjogos(\mathscr{G}_2)$, $p'_1 \cong p'_2$ se π mapear $subjogo_{\mathscr{G}_1}(p'_1)$ em $subjogo_{\mathscr{G}_2}(p'_2)$.

O Algoritmo 3.3 é o procedimento para reduzir um jogo estendido. Ele constrói um jogo $\mathscr{G}_1$ a partir de $\mathscr{G}$, potencialmente menor. Se $\mathscr{G}$ for determinado, $\mathscr{G}_1$ é um jogo com uma única posição $p_0 \in \{\top, \bot\}$, caso contrário, se $\mathscr{G}$ for indeterminado, $\mathscr{G}_1$ é um jogo que contém somente uma parte do jogo, cujas folhas estão indeterminadas. O laço das linhas $9-16$ avalia cada $subjogo_{\mathscr{G}}(p)$, tal que p está contido em $próximo(p_0)$. Na linha 11, se φ for uma fórmula universal e o jogo reduzido do $subjogo_{\mathscr{G}}(p)$ for falso, então o jogo reduzido do $subjogo_{\mathscr{G}}(p_0)$ também é falso. Porque como a fórmula é universal, um movimento do $subjogo_{\mathscr{G}}(p_0)$ para um subjogo falso é suficiente para invalidar a fórmula φ de p_0. Na linha 14, se φ for uma fórmula existencial e o jogo reduzido do $subjogo_{\mathscr{G}}(p)$ for verdadeiro, então o jogo reduzido do $subjogo_{\mathscr{G}}(p_0)$ também é verdadeiro. Porque como a fórmula é existencial, um movimento de $subjogo_{\mathscr{G}}(p_0)$ para um subjogo verdadeiro é suficiente para validar φ de p_0. Na linha 17, se o jogo reduzido do $subjogo_{\mathscr{G}}(p)$ for indeterminado e esse jogo não for equivalente a nenhum $subjogo_{\mathscr{G}_1}(p')$, tal que p' está contido em $próximo(p_0)$ de $\mathscr{G}_1$, então $\mathscr{G}_1$ é atualizado com o jogo reduzido de $subjogo_{\mathscr{G}}(p)$. A condicional da linha 22 é satisfeita toda vez que, ou φ é universal e todo $subjogo_{\mathscr{G}}(p')$, tal que p' está contido em $próximo(p_0)$, é verdadeiro ou φ é existencial e todo $subjogo_{\mathscr{G}}(p')$, tal que p' está contido em $próximo(p_0)$, é falso.

As chamadas ao Algoritmo 3.2 nas linhas 6 e 23, não consomem muito tempo de execução, pois na linha 6 a fórmula de p_0 é atômica ou negada e

Algoritmo 3.3: $reduza(\mathscr{G})$

Entrada: Um jogo EMC $\mathscr{G} = (P, M, P_0, P_1, p_0)$, com $p_0 = (\mathscr{H}, X, \varphi)$
Saída: Um jogo $EMC(\mathscr{G}_1)$

```
1  início
2      se 𝒢 está determinado então
3          retorna 𝒢
4      fim
5      se φ é uma fórmula atômica ou negada então
6          retorna avaliaEMC(𝒢)
7      fim
8      Construa um jogo 𝒢₁, tal que p₀ ∈ 𝒢₁
9      para cada p ∈ próximo(p₀) faça
10         𝒢' ← reduza(subjogo_𝒢(p))
11         se φ é universal e 𝒢' = ⊥ então
12             retorna ⊥
13         fim
14         se φ é existencial e 𝒢' = ⊤ então
15             retorna ⊤
16         fim
17         se 𝒢' é indeterminado e não é equivalente a qualquer subjogo em
           subjogos(𝒢₁) então
18             𝒢₁ ← 𝒢₁ ∪ 𝒢'
19             Adicione o um movimento de p₀ para 𝒢'
20         fim
21     fim
22     se 𝒢₁ = {p₀} então
23         retorna avaliaEMC(𝒢₁)
24     fim
25     retorna 𝒢₁
26 fim
```

na linha 23 p_0 é a única posição do jogo, ou seja, a chamada recursiva do Algoritmo 3.2 nunca é executada.

3.6 A função **combine**

Esta função é usada no processamento do *introduce* e *join* da fase 1 do algoritmo da Seção 3.2.

Seja X_i uma *bag introduce* e X_j a *bag* filha de X_i tal que $X_i = X_j \cup \{x\}$, para um vértice x do grafo.

Note que a *bag* X_i do tipo *introduce* difere da *bag* X_j apenas na adicão de arestas incidentes em x, tal que $\{x\} = X_i \setminus X_j$. Por isso, os jogos considerados em ambas as *bags* X_i e X_j são jogos parecidos. Sejam $\mathscr{R}_i$ e $\mathscr{R}_j$, jogos considerados nas *bags* X_i e X_j, respectivamente. Nessa seção veremos que se $\mathscr{R}_j \in \{\top, \bot\}$, então $\mathscr{R}_i$ tem a mesma estratégia vencedora de $\mathscr{R}_j$ (veja o Lema 4). Isso é

importante, pois se a fórmula φ já foi resolvida no subgrafo induzido por X_j, a adição de arestas não deve mudar a solução. Por outro lado, se $\mathscr{R}_j \notin \{\top, \bot\}$, então basta adicionar x na estrutura de $\mathscr{R}_j$.

Sejam X_i uma *bag join*, X_{j_1} e X_{j_2} as *bags* filhas de X_i e $\mathscr{R}_{j_{k(k\in\{1,2\})}}$ os jogos das *bags* X_{j_1} e X_{j_2}. Se $\mathscr{R}_{j_1} \in \{\top, \bot\}$ ou $\mathscr{R}_{j_2} \in \{\top, \bot\}$ então $\mathscr{R}_i$ tem a mesma estratégia vencedora de $\mathscr{R}_{j_{k(k\in\{1,2\})}}$ (veja o Lema 6). Por outro lado, se $\mathscr{R}_{j_1} \notin \{\top, \bot\}$ e $\mathscr{R}_{j_2} \notin \{\top, \bot\}$, então temos uma possível solução sobre o grafo induzido em X_{j_1} e temos uma possível solução sobre o grafo induzido em X_{j_2}. Então se unirmos tais soluções, temos uma possível solução para X_i.

Sejam $EMC(\mathscr{A}_1, X_1, \varphi) \notin \{\top, \bot\}$ e $EMC(\mathscr{A}_2, X_2, \varphi) \notin \{\top, \bot\}$. Seja $X = X_1 \cap X_2 = A_1 \cap A_2$. Note que estes jogos são do tipo $\mathscr{R}_j$ e $\mathscr{R}(U_i)$, considerados no procedimento *introduce* do algoritmo da Seção 3.2, e são do tipo $\mathscr{R}_{j_1}$ e $\mathscr{R}_{j_2}$ considerados no procedimento *join*. No *introduce*, $A_i \cap A_j = X_i \cap X_j = X_i \setminus \{x\}$. Porém, no *join* $A_{j_1} \cap A_{j_2} = X_{j_1} \cap X_{j_2} = X_i$.

Sabemos que tais jogos podem ter vindo de outras *bags*, portanto ambas as estruturas podem possuir universos maiores do que $(X_i)_{i\in\{1,2\}}$, como veremos na Seção 3.7. Além disso não existem arestas entre $A_1 \setminus X$ e $A_2 \setminus X$. Uma vez que os elementos de $A_1 \setminus X$ e $A_2 \setminus X$ podem ter sido removidos ao processar as *bags* anteriores.

Sejam p_1 e p_2 as posições iniciais de subjogos definidos em $EMC(\mathscr{A}_1, X_1, \varphi)$ e $EMC(\mathscr{A}_2, X_2, \varphi)$, respectivamente, tal que p_1 e p_2 têm a mesma fórmula ψ.

Seja c um símbolo nulário, tal que c está contido nas estruturas de p_1 e p_2. A interpretação de c está em $A_1 \setminus X$, $A_2 \setminus X$ ou X. Se a interpretação de c estiver em X e for igual em ambas as estruturas, então devemos analisar o que há em comum entre essas estruturas. Ou seja, vamos analisar $\mathscr{A}_1[X]$ e $\mathscr{A}_2[X]$. Se houver um isomorfismo entre tais estruturas induzidas em X, então ambas são iguais (veja a Definição 6). Nesse caso, poderíamos unir tais subjogos em um só.

Pelo Lema 5, a partir dos jogos *EMC* definidos anteriormente, podemos constuir um jogo equivalente ao jogo $reduza(EMC(\mathscr{A}_1 \cup \mathscr{A}_2, X_1 \cup X_2, \varphi))$. Dessa forma estamos juntando subjogos de um jogo no outro através da união entre as estruturas (veja Definição 7).

Dois jogos podem ser combinados quando a fórmula φ é a mesma e as estruturas dos jogos são compatíveis. O Lema 4, mostra que quando o jogo $\mathscr{R}_j$ da *bag* X_j é determinado, então o jogo $\mathscr{R}_i$ da *bag* X_i, que é do tipo *introduce*, deve ter a mesma estratégia vencedora.

Lema 4. (Nó do tipo *Introduce* [12]). Sejam $\mathscr{A}$ e $\mathscr{B}$ τ-estruturas compatíveis com $B = A \uplus \{b\}$[1]. Seja $X \subseteq A$ e $\varphi \in \mathrm{MSO}(\tau)$. Seja $\mathscr{G} = EMC(\mathscr{A}, X, \varphi)$ e $\mathscr{G}' = EMC(\mathscr{B}, X \cup \{b\}, \varphi)$,

[1] A união disjunta de dois conjuntos D_1, D_2 é denotada por $D_1 \uplus D_2$

1. Se $avalia(\mathcal{G}) = \top$, então $avalia(\mathcal{G}') = \top$; e

2. Se $avalia(\mathcal{G}) = \bot$, então $avalia(\mathcal{G}') = \bot$.

Ideia da Prova. A prova do Lema 4 é feita por indução sobre a estrutura da fórmula φ. Supomos que que $avalia(\mathcal{G})$ é $\top$, o caso $\bot$ é análogo.

Se φ é uma fórmula atômica, então seus símbolos nulários são interpretados, pois $avalia(\mathcal{G})$ é $\top$. Desde que $\mathscr{A}$ e $\mathscr{B}$ são compatíveis, as interpretações dos símbolos nulários de $\mathscr{A}$ e $\mathscr{B}$ são iguais, portanto temos que $avalia(\mathcal{G}')$ também é $\top$.

Se $\varphi = \psi_1 \vee \psi_2$ ou $\varphi = \psi_1 \wedge \psi_2$, por definição existe um subjogo em $subjogos(\mathcal{G})$ para cada $\psi \in \{\psi_1, \psi_2\}$ e um subjogo em $subjogos(\mathcal{G}')$ para cada $\psi \in \{\psi_1, \psi_2\}$. Por hipótese de indução, se $avalia(\mathcal{G}_\psi) = \top$ então $avalia(\mathcal{G}'_\psi) = \top$, logo se $avalia(\mathcal{G}) = \top$, temos que $avalia(\mathcal{G}') = \top$.

Se $\varphi = \exists c\psi$, existe uma $(\tau, c)-$expansão $\mathscr{A}'$ da estrutura $\mathscr{A}$ e um subjogo $\mathcal{G}_{c^{\mathscr{A}'}} \in subjogos(\mathcal{G})$, tal que $avalia(\mathcal{G}_{c^{\mathscr{A}'}}) = \top$. Seja $\mathscr{B}'$ uma $(\tau, c)-$expansão de $\mathscr{B}$, e um subjogo $\mathcal{G}'_{c^{\mathscr{B}'}} \in subjogos(\mathcal{G}')$, cuja interpretação de c em $\mathscr{A}'$ e $\mathscr{B}'$ são iguais. Nesse caso, $\mathscr{A}'$ e $\mathscr{B}'$ são compatíveis. Portanto $avalia(\mathcal{G}'_{c^{\mathscr{B}'}}) = \top$. Logo, $avalia(\mathcal{G}') = \top$.

Se $\varphi = \forall c\psi$, considere uma $(\tau, c)-$expansão $\mathscr{B}'$ de $\mathscr{B}$ e $\mathscr{A}' = \mathscr{B}[A]$. Note que, se a interpretação de c em $\mathscr{B}'$ for diferente de b, então c é interpretado para o mesmo elemento de A em ambas as estruturas $\mathscr{A}'$ e $\mathscr{B}'$. Mas, se a interpretação de c em $\mathscr{B}'$ for igual a b ou *nulo*, então c é interpretado para *nulo* em $\mathscr{A}'$. Nesse caso, $\mathscr{A}'$ e $\mathscr{B}'$ são compatíveis, portanto $avalia(\mathcal{G}') = \top$.

Para $\varphi = \exists R\psi$ ou $\varphi = \forall R\psi$, existe um subjogo $\mathcal{G}'_{R'} \in subjogos(\mathcal{G}')$, para todo $R' \subseteq B$ e existe um subjogo $\mathcal{G}_R \in subjogos(\mathcal{G})$, para todo $R \subseteq A$, tal que $R = R' \setminus \{b\}$. Quando $\varphi = \exists R\psi$, existe um subjogo $\mathcal{G}_R$, tal que $avalia(\mathcal{G}_R) = \top$. Desde que $(\mathscr{A}, R)$ e $(\mathscr{B}, R)$ são compatíveis, então $avalia(\mathcal{G}'_R) = \top$, logo $avalia(\mathcal{G}') = \top$. Quando $\varphi = \forall R\psi$, sabemos que $avalia(\mathcal{G}_R) = \top$, para todo $R \subseteq A$. Mas $(\mathscr{A}, R)$ e $(\mathscr{B}, R')$ são compatíveis, logo, $avalia(\mathcal{G}'_{R'}) = \top$ para todo $R' \subseteq B$. Portanto, $avalia(\mathcal{G}') = \top$. □

O Algoritmo 3.4 é o procedimento que combina dois jogos, cujas estruturas são compatíveis. Ele computa o produto cartesiano entre os $subjogos(\mathcal{G}_1)$ e $subjogos(\mathcal{G}_2)$ que estão no mesmo nível da árvore do jogo. Quando as estruturas destes subjogos são compatíveis e suas fórmulas φ são iguais, os dois subjogos são combinados através da união de suas estruturas e de suas *bags*. A linha 2, constrói a posição p_0 do novo subjogo combinado $\mathcal{G}$, que por sua vez é construído na linha 3. O laço das linhas $4-12$ e o laço aninhado das linhas $5-11$, são responsáveis por computar o produto cartesiano entre os filhos de dois subjogos e combinar tais filhos, quando possível.

A chamada ao Algoritmo 3.3 na linha 13 garante que o novo subjogo $\mathcal{G}$ não tenha jogos equivalentes entre si e avalia se o subjogo é $\top$ ou $\bot$, quando for

Algoritmo 3.4: $combine(\mathscr{G}_1, \mathscr{G}_2)$

Entrada: Dois jogos $\mathscr{G}_i = (P_i, M_i, P_{0_i}, P_{1_i}, p_{0_i})$, com $p_{0_i} = (\mathscr{H}_i, X_i, \varphi)$, tal que $\mathscr{H}_1$ e $\mathscr{H}_2$ são duas $\tau-$estruturas compatíveis e $X_i \subseteq H_i$.

Saída: Um jogo $\mathscr{G}$

```
1  início
2      Seja p_0 = (H_1 ∪ H_2, H_1 ∪ H_2, φ)
3      Construa um jogo G, tal que p_0 ∈ G
4      para cada p_1 ∈ próximo(p_0_1) faça
5          para cada p_2 ∈ próximo(p_0_2) faça
6              se As fórmulas MSO de p_1 e p_2 forem iguais e a estrutura de p_1
               for compatível com a estrutura de p_2 então
7                  G' ← combine(subjogo_G1(p_1), subjogo_G2(p_2))
8                  G ← G ∪ G'
9                  Adicione o um movimento de p_0 para G' em G.
10             fim
11         fim
12     fim
13     retorna reduza(G)
14 fim
```

determinado.

O Lema 5 afirma que combinar dois jogos é equivalente a construir um jogo EMC, tal que sua estrutura $\mathscr{A}$ é a união das estruturas dos dois jogos, e seu conjunto X é a união dos conjuntos X dos dois jogos.

Lema 5. ([12]) Sejam $\mathscr{A}_1$ e $\mathscr{A}_2$ $\tau-$estruturas compatíveis, $\varphi \in MSO(\tau)$ e $X_1 \subseteq A_1$ e $X_2 \subseteq A_2$ com $A_1 \cap A_2 = X_1 \cap X_2$. Para $i \in \{1,2\}$, $\mathscr{R}_i = reduza(EMC(\mathscr{A}_i, X_i, \varphi)) \notin \{\top, \bot\}$ e $\mathscr{G}_i \cong \mathscr{R}_i$. Então

$$reduza(EMC(\mathscr{A}_1 \cup \mathscr{A}_2, X_1 \cup X_2, \varphi)) \cong combine(\mathscr{G}_1, \mathscr{G}_2).$$

Lema 6. (Nó do tipo *Join* [12]). Sejam $\mathscr{A}, \mathscr{B}$ τ-estruturas compatíveis, $X \subseteq A \cap B$, e $\varphi \in MSO(\tau)$. Sejam $\mathscr{G} = EMC(\mathscr{A}, X, \varphi)$ e $\mathscr{G}' = EMC(\mathscr{A} \cup \mathscr{B}, X, \varphi)$

1. Se $avalia(\mathscr{G}) = \top$, então $avalia(\mathscr{G}') = \top$; e
2. Se $avalia(\mathscr{G}) = \bot$, então $avalia(\mathscr{G}') = \bot$.

O jogo $\mathscr{G}$ da Figura 3.7 é o jogo resultante ao combinar os jogos da Figura 3.6. Note que os subjogos da fórmula $\forall x$ tem as possibilidades de combinação conforme a Tabela 3.1. Mas a chamada ao procedimento $reduza(\mathscr{G}_1)$ na linha 13 do Algoritmo 3.4, elimina os jogos que são equivalentes (veja Seção 3.5). Os pares $(\vee, \vee)$, tal que as interpretações de y são distintas, não são compatíveis, pois a definição de compatibilidade afirma que, os símbolos

nulários que são interpretados nas duas estruturas devem ter a mesma interpretação. Note que a Tabela 3.1 apresenta somente o produto cartesiano entre os filhos de $\forall x$. Os subjogos da fórmula $\vee$ que vieram de $x = nulo$ possuem fórmulas distintas. Portanto, esses subjogos só existirão ao combinar $x = nulo$ de $\mathscr{G}_1$ com $x = nulo$ de $\mathscr{G}_2$.

$(\mathscr{G}_{1_\psi}, \mathscr{G}_{2_\psi})$	$\mathscr{G}_1$	$\mathscr{G}_2$	Compatíveis
$(\vee, \vee)$	$x = nulo$	$x = nulo$	$\surd$
$(\vee, \vee)$	$x = nulo$	$x = 2$	$\surd$
$(\vee, \vee)$	$x = nulo$	$x = 3$	$\surd$
$(\vee, \vee)$	$x = 3$	$x = nulo$	$\surd$
$(\vee, \vee)$	$x = 3$	$x = 2$	
$(\vee, \vee)$	$x = 3$	$x = 3$	$\surd$
$(\vee, \vee)$	$x = 4$	$x = nulo$	$\surd$
$(\vee, \vee)$	$x = 4$	$x = 2$	
$(\vee, \vee)$	$x = 4$	$x = 3$	
$(\vee, \vee)$	$x = 5$	$x = nulo$	$\surd$
$(\vee, \vee)$	$x = 5$	$x = 2$	
$(\vee, \vee)$	$x = 5$	$x = 3$	

Tabela 3.1: Produto cartesiano entre os subjogos de $\psi = \forall y$ dos jogos de $\mathscr{G}_1$ e $\mathscr{G}_2$ da Figura 3.6.

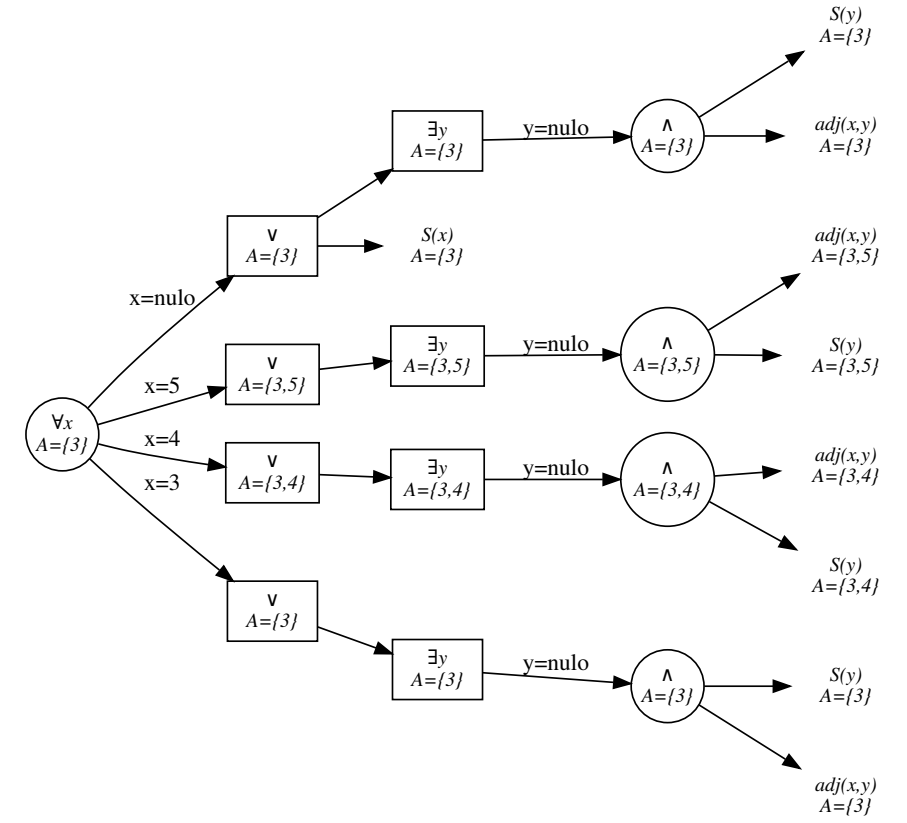
3.7 A função **remove**

Seja X_i uma *bag forget* e X_j a *bag* filha de X_i tal que $X_j = X_i \cup \{x\}$, sendo x um vértice do grafo.

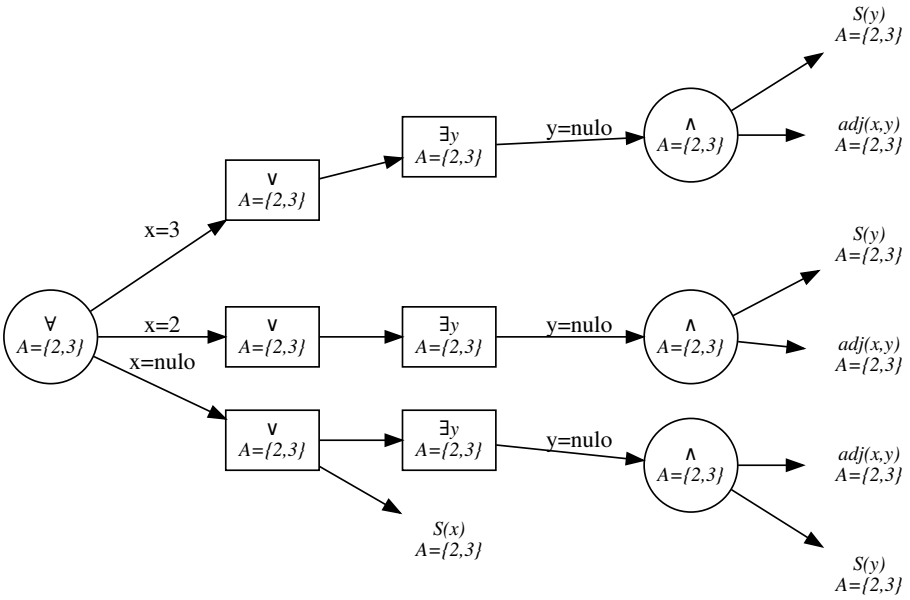
Note que a *bag* X_i difere da *bag* X_j apenas na remoção do vértice x e das arestas incidentes em x. Por isso, os jogos considerados em ambas as *bags* X_i e X_j são jogos parecidos. Sejam $\mathscr{R}_i$ e $\mathscr{R}_j$, jogos considerados nas *bags* X_i e X_j, respectivamente. Nessa seção veremos que se $\mathscr{R}_j \in \{\top, \bot\}$, então $\mathscr{R}_i$ tem a mesma estratégia vencedora de $\mathscr{R}_j$ (veja o Lema 7). Isso é importante, pois se a fórmula φ já foi resolvida no subgrafo induzido por X_j, a remoção de arestas incidentes em x não deve mudar a solução. Por outro lado, se $\mathscr{R}_j \notin \{\top, \bot\}$, então basta desconsiderar x da estrutura de $\mathscr{R}_j$.

O objetivo do Algoritmo 3.5 é manter nos jogos apenas os ramos necessários, ou seja, manter jogos o quanto menores possível.

Seja $G = (V, E)$ um grafo, $X \subseteq V$, $\mathscr{G} = reduza(EMC(\mathscr{A}, X, \varphi)) \notin \{\top, \bot\}$ e $x \in X$. Se todos os vizinhos de x estão em X, então não existem arestas $adj(x, y)$, tal que $y \in V \setminus X$. Assim podemos desconsiderar x de todos os subjogos do jogo $\mathscr{G}$, que não possuem um símbolo nulário interpretado para x. Podemos remover x do universo das estruturas dos subjogos, inclusive removê-lo da interpretação do símbolos unários e remover as arestas que incidem nele.



(a) $\mathscr{G}_1$



(b) $\mathscr{G}_2$

Figura 3.6: O jogo $\mathscr{G}_1$ foi computado na *bag* $X_j = \{3\}$ que é do tipo *forget* com $U = \{\}$. O jogo $\mathscr{G}_2$ foi computado na *bag* $X_i = \{2,3\}$ que é do tipo *introduce* com $U = \{\}$. O grafo considerado aqui é um caminho com 6 vértices.

Essa remoção pode gerar subjogos com estruturas muito parecidas entre si. Tornando assim os subjogos equivalentes. Isso ocorre desde que nos sub-

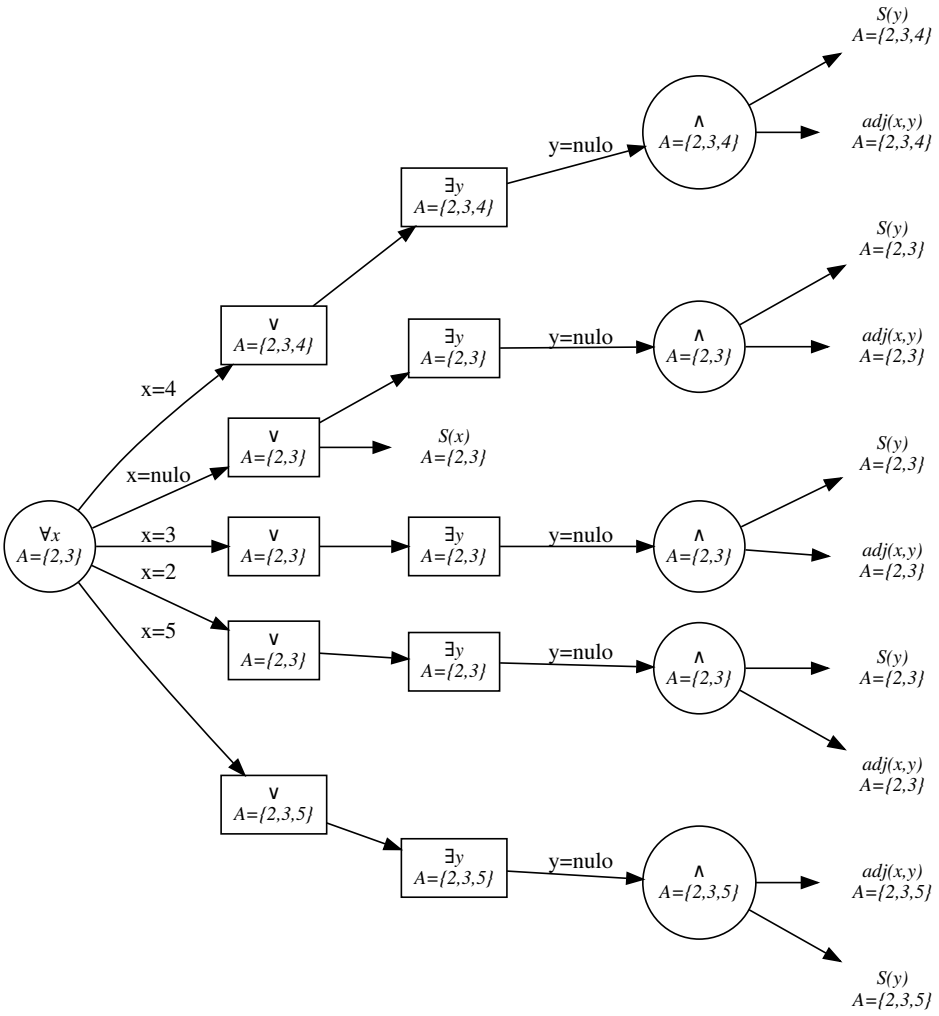


Figura 3.7: O jogo $\mathscr{G}$ retornado da função $combine(\mathscr{G}_1,\mathscr{G}_2)$, tal que $\mathscr{G}_1$ e $\mathscr{G}_2$ são os jogos da Figura 3.6. O grafo considerado nesse jogo é um caminho com 6 vértices e $U = \{\}$.

jogos que não interpretam símbolos nulários para x, a estrutura contenha x e todas as interpretações dos símbolos relacionais e unários que foram desconsiderados anteriormente.

O jogo resultante desse processo é equivalente ao original, ainda que um elemento x tenha sido desconsiderado. Pelo Lema 8 o jogo $\mathscr{G}$ resultante da remoção de x é equivalente ao $reduza(EMC(\mathscr{A}, X \setminus \{x\}, \varphi))$.

O Algoritmo 3.5 é o procedimento *remove*. Ele constrói um novo jogo $\mathscr{G}_1$ com base no jogo $\mathscr{G}$. Para cada novo $subjogo_{\mathscr{G}_1}(p_0)$, o algoritmo verifica, nas linhas $2-7$, se o elemento x é a interpretação de algum símbolo nulário do vocabulário τ. Se for, x é removido da *bag* X de p_0, mas continuará existindo na estrutura. Caso contrário, x é removido da *bag* X e da estrutura de p_0.

$\mathscr{G}$.

Lema 7. (Nó do tipo *Forget* [12]). Sejam $\mathscr{A}$ uma τ-estrutura, $X' \subseteq X \subseteq A$ e $\varphi \in$ MSO(τ). Seja $\mathscr{G} = EMC(\mathscr{A}, X, \varphi)$ e $\mathscr{G}' = EMC(\mathscr{A}, X', \varphi)$,

Algoritmo 3.5: $remove(\mathscr{G},x)$

Entrada: Um jogo $\mathscr{G} = (P,M,P_0,P_1,p_0)$, com $p_0 = (\mathscr{H},X,\varphi)$ e $x \in X$.
Saída: Um jogo $\mathscr{G}_1$

```
1  início
2      se Existe um símbolo nulário cuja interpretação é igual a x em ℋ então
3          Seja p_0 = (ℋ, X \ {x}, φ).
4      fim
5      senão
6          Seja p_0 = (ℋ[H \ {x}], X \ {x}, φ).
7      fim
8      Construa um jogo 𝒢_1, tal que p_0 ∈ 𝒢_1
9      para cada p ∈ próximo(𝒢) faça
10         𝒢' ← remove(subjogo_𝒢(p))
11         𝒢_1 ← 𝒢_1 ∪ 𝒢'
12         Adicione o um movimento de p_0 para 𝒢' em 𝒢_1.
13     fim
14     retorna reduza(𝒢_1)
15 fim
```

1. Se $avalia(\mathscr{G}) = \top$ então $avalia(\mathscr{G}') = \top$
2. Se $avalia(\mathscr{G}) = \bot$ então $avalia(\mathscr{G}') = \bot$

Ideia da Prova. Os jogos $\mathscr{G}$ e $\mathscr{G}'$ do Lema 7 são quase idênticos, com uma única diferença nas posições p_0 de cada um. Sendo que $p_0 = (\mathscr{H},X,\varphi) \in \mathscr{G}$ e $p_0' = (\mathscr{H}',X',\varphi) \in \mathscr{G}'$, onde $\mathscr{H} = \mathscr{A}[X \cup \bar{c}^{\mathscr{A}}]$ e $\mathscr{H}' = \mathscr{A}[X' \cup \bar{c}^{\mathscr{A}}]$. Note que, as estruturas das posições p_0 e p_0' são induzidas na união de sua *bag* com as interpretações da estrutura $\mathscr{A}$. Isso permite que os elementos contidos em $X \setminus X'$ continuem existindo nas estruturas induzidas. Além disso, p_0 e p_0' pertencem ao mesmo jogador. Portanto os itens 1 e 2 do Lema 7, se verificam. □

Lema 8. ([12]) Sejam $\mathscr{A}$ uma τ−estrutura, $X \subseteq A$ e $x \in X$. Seja $\varphi \in MSO(\tau)$ e $\mathscr{G} \cong reduza(EMC(\mathscr{A},X,\varphi)) \notin \{\top,\bot\}$. Então

$$reduza(EMC(\mathscr{A},X \setminus \{x\},\varphi)) \cong remove(\mathscr{G},x).$$

Observe na Figura 3.7 que o jogo $\mathscr{G}$ contém em todos os seus subjogos o elemento 3. Contudo, a Figura 3.8 apresenta o jogo $\mathscr{G}_1$ como resultado do algoritmo $remove(\mathscr{G},3)$. Note que $\mathscr{G}_1$ manteve todo o ramo em que o símbolo nulário x é interpretado para 3, porém removeu 3 de todos os outros subjogos que não pertencem a esse ramo. Perceba também que a ramificação em que o símbolo nulário x era interpretado para 5, foi eliminada. Isso porque o algoritmo $reduza(\mathscr{G}_1)$ detectou equivalência entre esse ramo e outro, após as remoções.

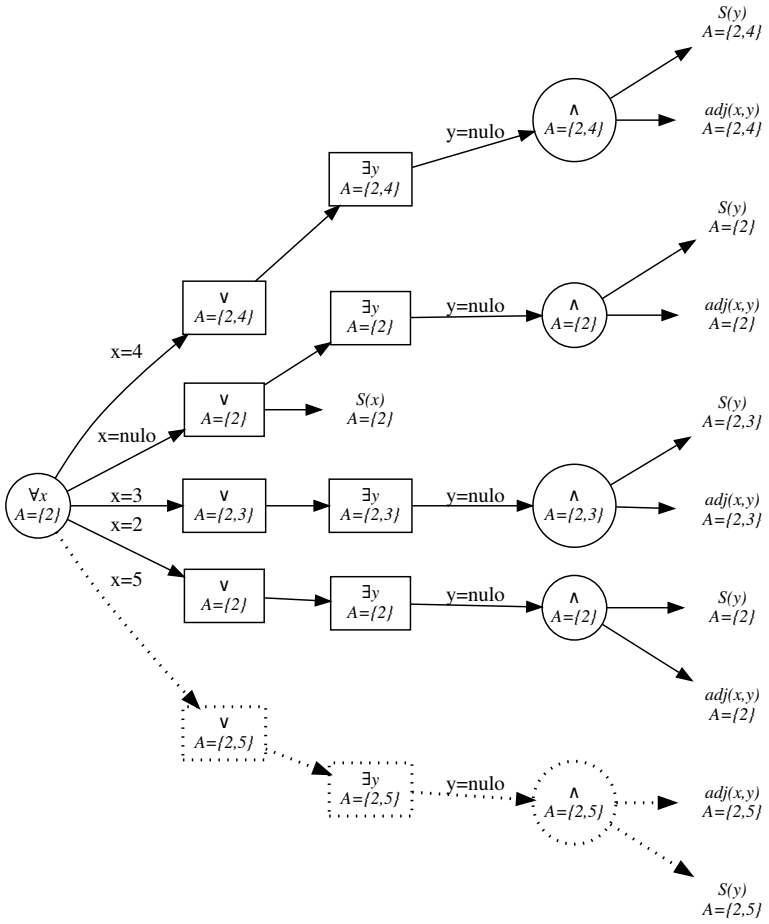


Figura 3.8: Jogo $\mathscr{G}_1$ retornado do Algoritmo 3.5, que teve como entrada o jogo $\mathscr{G}$ da Figura 3.7 com $x = 3$ e $U = \{\}$. O grafo considerado aqui é um caminho com 6 vértices. Arestas pontilhadas denotam que não pertencem mais ao jogo.

3.8 Solução em Problemas de Otimização

Na Seção 3.2 vimos um algoritmo que utiliza a decomposição em árvore *nice* para decidir se uma fórmula $\varphi \in MSO(\tau)$ vale para um dado grafo $G = (V,E)$. Vimos que em alguns problemas a fórmula φ exige como entrada um conjunto U, tal que $U \subseteq V$, além disso, U é a solução para o problema expresso em φ sobre o grafo G.

Vamos considerar agora o problema de encontrar o melhor U de G para φ. Teríamos que testar todos os subconjuntos do conjunto de vértices do grafo G, ou seja, teríamos que executar o algoritmo $2^{|V|}$ vezes, tal que $|V|$ é o tamanho de V. Supomos que $V = \{0,1,2,3\}$. Temos o seguinte conjunto potência de V,

$$\begin{aligned}\mathscr{P}(V) = &\{\{\},\{0\},\{1\},\{2\},\{3\},\{0,1\},\{0,2\},\{0,3\},\{1,2\},\{1,3\},\\ &\{2,3\},\{0,1,2\},\{0,1,3\},\{0,2,3\},\{1,2,3\},\{0,1,2,3\}\}\end{aligned}$$

Imagine que estamos procurando o melhor U no ciclo com 4 vértices (C_4)

para o problema do Conjunto do Dominte

$$cd(S) = \forall x(x \in S \ \vee \ \exists y(y \in S \wedge adj(x,y))).$$

A Figura 3.9 é a decomposição em árvore *nice* do grafo C_4. Observe que ela contém uma *bag* $X_i = \{0,3\}$. Note que, se executarmos o algoritmo da Seção 3.2 para cada $U \in \mathscr{P}(V)$, temos que para cada $U \in \{\{0,3\},\{0,1,3\},\{0,2,3\},\{0,1,2,3\}\}$, o mesmo jogo $\mathscr{R}(U_i)$ seria construído, para $U_i = U \cap X_i$.

O Teorema 3.1, apresenta uma solução para esse problema, que é um problema de otimização. Uma nova versão do algoritmo da Seção 3.2 é descrito aqui, como uma construção prática desse teorema. Essa versão, utiliza a técnica da programação dinâmica, o que evita a construção de um mesmo jogo várias vezes.

Teorema 3.1. (Teorema 2 [12].) Seja τ um vocabulário relacional, um conjunto $\bar{R} = \{R_1, R_2, ..., R_l\}$ que pertence ao conjunto de símbolos unários de τ, e $\tau' = \tau \setminus \bar{R}$. Seja $\varphi \in MSO(\tau)$, e $w, \alpha_1, \alpha_2, ..., \alpha_l \in \mathbb{Z}$ constantes. Dado uma $\tau'-$estrutura $\mathscr{A}$ e uma decomposição em árvore $D = (X,T)$ de $\mathscr{A}$ com largura no máximo w, onde $T = (I,F)$ e $X = (X_i)_{i \in T}$, podemos computar

$$min\left\{\sum_{k=1}^{l} \alpha_k |U_k| \text{ tal que } U_i \subseteq A, 1 \leq i \leq l, \text{ e } (\mathscr{A}, U_1, U_2, ..., U_l) \models \varphi\right\}$$

em tempo $O(|T|)$.

O algoritmo descrito a seguir é dividido em duas fases, como na primeira versão. Cada nó i da decomposição será associado à uma tabela S_i que contém alguns jogos e seus correspondentes valores, que é a solução parcial do problema.

Na primeira fase o algoritmo computa os jogos reduzidos estendidos $\mathscr{G} \cong reduza(\mathscr{A}_i', \emptyset, \varphi)$ e os valores $\sum_{k=1}^{l} \alpha_k |U_k|$ para todas as estruturas $\mathscr{A}_i' = (\mathscr{A}, U_1, ..., U_l)$ onde $U_i \subseteq A$ para $1 \leq i \leq l$.

Na segunda fase, o algoritmo já atingiu todas as *bags* da decomposição inclusive a raiz. Portanto, ele testa se o verificador possui uma estratégia vencedora em cada jogo $\mathscr{G}$, e escolhe o jogo com o melhor valor. Em outras palavras, testa se $(\mathscr{A}, U_1, ..., U_l) \models \varphi$.

Assumimos que $X_{raiz} = \emptyset$, e para cada nó i da decomposição, $\mathscr{A}_i$ é uma subestrutura de $\mathscr{A}$ induzida pelos elementos da *bag* X_i. E $\mathscr{A}_i(U) = (\mathscr{A}_i, U)[X_i]$.

Seja

$$\mathscr{AR}_i = \mathscr{P}(\mathscr{A}_i) \times ... \times \mathscr{P}(\mathscr{A}_i) = \mathscr{P}(\mathscr{A}_i)^l \ ,$$

o conjunto de todas as possíveis interpretações de símbolos relacionais livres $(R_1, ..., R_l)$ em $\mathscr{A}_i$. Por exemplo, seja $l = 2$, $A_i = \{1,2\}$ o universo de $\mathscr{A}_i$ temos que,

$$\mathscr{P}(\mathscr{A}_i) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\} \text{ , então}$$

$$\mathscr{A}\mathscr{R}_i = \{ \begin{array}{llll} (\emptyset,\emptyset), & (\emptyset,\{1\}), & (\emptyset,\{2\}), & (\emptyset,\{1,2\}), \\ (\{1\},\emptyset), & (\{1\},\{1\}), & (\{1\},\{2\}), & (\{1\},\{1,2\}), \\ (\{2\},\emptyset), & (\{2\},\{1\}), & (\{2\},\{2\}), & (\{2\},\{1,2\}), \\ (\{1,2\},\emptyset), & (\{1,2\},\{1\}), & (\{1,2\},\{2\}), & (\{1,2\},\{1,2\}) \quad \}. \end{array}$$

Sejam $(U_1,...,U_l) \cap X_i \leftarrow (U_1 \cap X_i,...,U_l \cap X_i)$ e

$$\mathscr{A}\mathscr{R}_i \cap X_i = \{(U_1,...,U_l) \cap X_i \mid (U_1,...,U_l) \in \mathscr{A}\mathscr{R}_i\}$$

uma restrição de $\mathscr{A}\mathscr{R}_i$ para X_i. Usando o exemplo anterior, seja $X_i = \{2\}$ temos,

$$\mathscr{A}\mathscr{R}_i \cap X_i = \{ \begin{array}{ll} (\emptyset,\emptyset), & (\emptyset,\{2\}), \\ (\{2\},\emptyset), & (\{2\},\{2\}) \quad \}. \end{array}$$

Seja $\bar{U} = (U_1,...,U_l) \in \mathscr{A}\mathscr{R}_i \cap X_i$. A tabela S_i é um mapeamento de tuplas $\bar{U} \in \mathscr{A}\mathscr{R}_i \cap X_i$ para conjuntos de jogos sobre a estrutura $\mathscr{A}_i$. Ou seja, jogos $\mathscr{R}$ com $\mathscr{R} \neq \bot$ e $val_i \in \mathbb{Z} \cup \{\infty\}$ os correspondentes valores de tais jogos.

Primeiramente, inicializamos a tabela. Seja $S_i(\bar{U}) = \emptyset$ para todo $\bar{U} \in \mathscr{A}\mathscr{R}_i \cap X_i$ e $val_i(\mathscr{R}) \leftarrow \infty$ para todo jogo reduzido $\mathscr{R}$.

1ª fase. O algoritmo atravessa a decomposição *nice* a partir das folhas até a raiz. E cada nó i da decomposição é uma folha ou um dos tipos *introduce*, *forget* ou *join*.

folha: Seja $X_i = \{x\}$. Para todo $\bar{U} = (U_1,...,U_l) \in \mathscr{A}\mathscr{R}_i \cap X_i$ o algoritmo constrói $\mathscr{R}(\bar{U}) = reduza(\mathscr{A}_i(\bar{U}), X_i, \varphi)$.

Se $\mathscr{R}(\bar{U}) \neq \bot$, então

$$S_i(\bar{U}) \leftarrow \{\mathscr{R}(\bar{U})\} \text{ e } val_i(\mathscr{R}(\bar{U})) \leftarrow 0.$$

introduce: Seja j o filho único de i e $X_i = X_j \cup \{x\}$ para $x \notin A_j$.

Para cada $\bar{U}_j = (U_{j,1},...,U_{j,l}) \in \mathscr{A}\mathscr{R}_j \cap X_j$ e cada $\bar{U}_i = (U_{i,1},...,U_{i,l}) \in \mathscr{A}\mathscr{R}_i \cap X_i$, tal que $(U_{j,1},...,U_{j,l}) = (U_{i,1} \cap X_j,...,U_{i,l} \cap X_j)$, o algoritmo considera cada jogo $\mathscr{R}_j \in S_j(\bar{U}_j)$.

Seja

$$\mathscr{R}_i = \begin{cases} \top & \text{se } \mathscr{R}_j = \top \text{ , e} \\ combine(\mathscr{R}_j, \mathscr{R}(\bar{U}_i)) & \text{senão.} \end{cases}$$

Se existe um $\mathscr{R}'_i \in S_i(\bar{U}_i)$ com $\mathscr{R}'_i \cong \mathscr{R}_i$, então $\mathscr{R}_i \leftarrow \mathscr{R}'_i$.

Se $\mathscr{R}_i \neq \bot$, então $S_i(\bar{U}_i) \leftarrow S_i(\bar{U}_i) \cup \{\mathscr{R}_i\}$ e $val_i(\mathscr{R}_i) \leftarrow min\{val_i(\mathscr{R}_i), val_j(\mathscr{R}_j)\}$.

forget: Seja j o filho único de i e $X_j = X_i \cup \{x\}$, para $x \notin A_i$.

Para cada $\bar{U}_j = (U_{j,1}, ..., U_{j,l}) \in \mathscr{A}\mathscr{R}_j \cap X_j$ o algoritmo considera cada jogo $\mathscr{R}_j \in S_j(\bar{U}_j)$. Seja $\bar{U}_i = (U_{i,1}, ..., U_{i,l}) = (U_{j,1} \cap X_i, ..., U_{j,l} \cap X_i)$ e

$$\mathscr{R}_i = \begin{cases} \top & \text{se } \mathscr{R}_j = \top \text{ , e} \\ remove(\mathscr{R}_j, x) & \text{senão.} \end{cases}$$

Se existe um $\mathscr{R}'_i \in S_i(\bar{U}_i)$ com $\mathscr{R}'_i \cong \mathscr{R}_i$, então $\mathscr{R}_i \leftarrow \mathscr{R}'_i$.
Se $\mathscr{R}_i \neq \bot$, então $S_i(\bar{U}_i) \leftarrow S_i(\bar{U}_i) \cup \{\mathscr{R}_i\}$ e

$$val_i(\mathscr{R}_i) \leftarrow min \left\{ val_i(\mathscr{R}_i), val_j(\mathscr{R}_j) + \sum_{k=1}^{l} \alpha_k (x \in U_{j,k}) \right\}$$

onde

$$(x \in U_{j,k}) = \begin{cases} 0 & \text{se } x \notin U_{j,k} \\ 1 & \text{se } x \in U_{j,k}. \end{cases}$$

join: Sejam j_1 e j_2 , filhos de i e $X_{j_1} = X_{j_2} = X_i$. Para cada $\bar{U} = (U_1, ..., U_l) \in \mathscr{A}\mathscr{R}_i \cap X_i$ o algoritmo considera cada par $(\mathscr{R}_{j_1}, \mathscr{R}_{j_2}) \in S_{j_1}(\bar{U}) \times S_{j_2}(\bar{U})$.

Seja

$$\mathscr{R}_i = \begin{cases} \top & \text{se } \mathscr{R}_{j_1} = \top \text{ ou } \mathscr{R}_{j_2} = \top \text{ , e} \\ combine(\mathscr{R}_{j_1}, \mathscr{R}_{j_2}) & \text{senão.} \end{cases}$$

Se existe um $\mathscr{R}'_i \in S_i(\bar{U}_i)$ com $\mathscr{R}'_i \cong \mathscr{R}_i$, então $\mathscr{R}_i \leftarrow \mathscr{R}'_i$.
Se $\mathscr{R}_i \neq \bot$, então $S_i(\bar{U}_i) \leftarrow S_i(\bar{U}_i) \cup \{\mathscr{R}_i\}$ e

$$val_i(\mathscr{R}_i) \leftarrow min\{val_i(\mathscr{R}_i), val_{j_1}(\mathscr{R}_{j_1}) + val_{j_2}(\mathscr{R}_{j_2})\}.$$

2ª fase. Seja r a raíz da decomposição e

$$\bar{U}_r = (\emptyset, ..., \emptyset) \in \mathscr{A}\mathscr{R}_r \cap X_r = \mathscr{A}\mathscr{R}_r \cap \emptyset$$

O algoritmo começa com $OPT \leftarrow \infty$ e considera cada $\mathscr{R}_r \in S_r(\bar{U}_r)$.
Se $avalia(converte(\mathscr{R}_r)) = \top$, então o algoritmo atualiza

$$OPT \leftarrow min\{OPT, val_r(\mathscr{R}_r)\}.$$

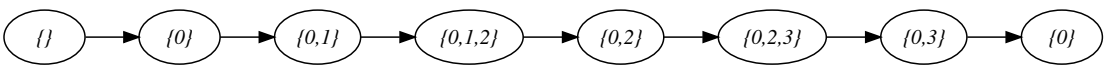


Figura 3.9: Decomposição em Árvore *Nice* do Ciclo com 4 vértices, tal que $V = \{0,1,2,3\}$ é o conjunto de vértices.

Descrição do Algoritmo. O valor val_i de $\mathscr{R}_i$ é calculado pela somatória das contribuições dos vértices contidos em cada $U_k \in \bar{U}$. Essa contribuição é definida por α_k na função *min*.

Na primeira fase ao processar as folhas todos os jogos são inicializados com o valor 0. Esses jogos, são as primeiras soluções encontradas para os subconjuntos dos elementos da *bag* folha.

Cada vez que o algoritmo processa uma *bag* X_i do tipo *introduce*, cada jogo de $\bar{U}_j$ é considerado:

- adicionando o elemento x no jogo para $\bar{U}_j$, e
- adicionando o elemento x no jogo para cada $U \in \bar{U}_j \cup \{x\}$,

assim, temos o valor de cada jogo, considerando todas as possibilidades de U para os vértices considerados até o momento. O valor de cada jogo $\mathscr{R}_i$ é o melhor valor entre:

1. $val_j(\mathscr{R}_j)$; e
2. $val_i(\mathscr{R}'_i)$, se existir um $\mathscr{R}'_i$ tal que $\mathscr{R}'_i \cong \mathscr{R}_i$.

No primeiro caso temos que, φ vale para $\bar{U}_j$, logo φ vale para $\bar{U}_j \cup \{x\}$, portanto, o valor da solução se mantém. No segundo caso, existe uma solução equivalente na tabela $S_i(\bar{U}_i)$, então o valor da solução equivalente se mantém.

Cada vez que o algoritmo processa uma *bag* X_i do tipo *forget*, cada jogo de $\bar{U}_j$ é considerado para cada $\bar{U}_i = \bar{U}_j \cap X_i$:

- "removendo" o elemento x do jogo para $\bar{U}_j$.

Note que na execução do *introduce* o valor do jogo $\mathscr{R}_i$ é apenas atualizado de acordo com o valor de outro jogo já existente. Note também que na *bag* X_i do tipo *join* cada vértice $y \in X_i$ foi introduzido uma vez na subárvore da filha X_{j_1} e outra vez na subárvore da filha X_{j_2}, isto é, y foi introduzido duas vezes. Por outro lado, um vértice x só pode ser removido uma única vez da decomposição, conforme a definição da decomposição em árvore. Portanto o processamento do *forget* requer um pouco mais de atenção. Se o elemento x que está sendo removido estiver contido em $\bar{U}_j$, o cálculo de $val_i(\mathscr{R}_i)$ deve considerar a soma da contribuição do vértice x no valor de $\mathscr{R}_j$. Portanto, o valor de cada jogo $\mathscr{R}_i$ na execução do *forget* é o melhor valor entre:

1. $val_j(\mathscr{R}_j)$ somado a α_k, para cada $U_{j_k} \in \bar{U}_j$ que contém x; e
2. $val_i(\mathscr{R}'_i)$, se existir um $\mathscr{R}'_i$ tal que $\mathscr{R}'_i \cong \mathscr{R}_i$.

Cada vez que o algoritmo processa uma *bag* X_i do tipo *join*, cada par de jogos $(\mathscr{R}_{j_1},\mathscr{R}_{j_2})$ é considerado para cada $\bar{U}$ de $S_{j_1}(\bar{U}) \times S_{j_2}(\bar{U})$. Ou seja, $\mathscr{R}_{j_{k(k\in\{1,2\})}}$ representa uma solução encontrada para o subgrafo induzido em todas as *bags* abaixo de X_{j_k} na decomposição, para $\bar{U}$. Ou seja, as soluções de X_{j_k} ainda não computaram os vértices de X_i que são exatamente os vértices contidos na intersecção entre X_{j_1} e X_{j_2}. Portanto para cada par de jogo $(\mathscr{R}_{j_1},\mathscr{R}_{j_2})$, o valor de $\mathscr{R}_i$ é o melhor valor entre :

1. $val_{j_1}(\mathscr{R}_{j_1}) + val_{j_2}(\mathscr{R}_{j_2})$; e
2. $val_i(\mathscr{R}'_i)$, se existir um $\mathscr{R}'_i$ tal que $\mathscr{R}'_i \cong \mathscr{R}_i$.

No primeiro caso estamos somando as constribuições dos vértices já removidos em cada jogo. No segundo caso, já existe uma solução para outro par de jogos que é equivalente a essa, então o valor da solução equivalente se mantém.

Agora mostramos um exemplo das tuplas que são consideradas no processamento do forget e do introduce, sobre a decomposição da Figura 3.9 com $l = 1$.

Iniciando na folha, o algoritmo construirá um jogo para cada $U_i \in \bar{U}$, tal que $\bar{U} = (((\{\}),(\{0\}))$. Supomos que todos os jogos são $\neq \perp$. Assim, a tabela $S_i(\bar{U}_i)$ conterá dois jogos, um para a tupla $U_1 = (\{\})$ e outro para a tupla $U_2 = (\{0\})$.

Agora a programação dinâmica alcança a *bag* mãe da folha, e esse nó é do tipo *introduce*. Cada tupla U_j de $\bar{U}_j$ tem uma lista de tuplas U_i de $\bar{U}_i$ tal que $U_j = U_i \cap X_j$, de acordo com a Tabela 3.2.

U_i	$U_j = U_i \cap X_j$
$(\{\})$	$(\{\})$
$(\{0\})$	$(\{0\})$
$(\{3\})$	$(\{\})$
$(\{0,3\})$	$(\{0\})$

Tabela 3.2: Relação entre as tuplas U_j e U_i para $X_i = \{0,3\}$.

Note também que o algoritmo desconsidera uma tupla U_j quando encontra um jogo falso. Isso significa que esse U_j não é uma solução para a fórmula φ no grafo considerado.

Seguindo na decomposição, a *bag* $\{0,2,3\}$ é alcançada, e esta também é do tipo *introduce*. Portanto o procedimento é o mesmo da *bag* anterior. E a relação entre as tuplas U_j e U_i é dada na Tabela 3.3. E novamente a tabela $S_i(\bar{U}_i)$ terá um jogo $\mathscr{R}_i$ para cada $U_i \subseteq \bar{U}_i$, se $\mathscr{R}_i \neq \perp$.

A próxima a *bag* a ser atingida é um nó do tipo *forget*. Neste caso U_i tem uma lista de tuplas U_j, tal que $(U_i) = (U_j \cap X_i)$. Isso significa que agora cada tupla U_i contém mais de um jogo. Veja a Tabela 3.4 para $X_i = \{0,2\}$.

U_i	$U_j = U_i \cap X_j$
$(\{\})$	$(\{\})$
$(\{0\})$	$(\{0\})$
$(\{2\})$	$(\{\})$
$(\{3\})$	$(\{3\})$
$(\{0,2\})$	$(\{0\})$
$(\{0,3\})$	$(\{0,3\})$
$(\{2,3\})$	$(\{3\})$
$(\{0,2,3\})$	$(\{0,3\})$

Tabela 3.3: Relação entre as tuplas U_j e U_i para $X_i = \{0,2,3\}$.

$U_i = (U_j \cap X_i)$	U_j
$(\{\})$	$(\{\})$
$(\{\})$	$(\{3\})$
$(\{0\})$	$(\{0\})$
$(\{0\})$	$(\{0,3\})$
$(\{2\})$	$(\{2\})$
$(\{2\})$	$(\{2,3\})$
$(\{0,2\})$	$(\{0,2\})$
$(\{0,2\})$	$(\{0,2,3\})$

Tabela 3.4: Relação entre as tuplas U_j e U_i para $X_i = \{0,2\}$.

Quando o nó é do tipo *join* as tuplas $\bar{U}_{j_1}, \bar{U}_{j_2}$ e $\bar{U}_i$ são iguais, mas a lista de jogos de cada tupla $U_i = U_{j_1} = U_{j_2}$ são distintas, pois cada *bag* $X_{j_{k(k\in\{1,2\})}}$ está em caminhos diferentes na decomposição.

3.9 Complexidade do Algoritmo

Vimos na Seção 2.2 que a abordagem prática para o Teorema 1 de Courcelle que utiliza a construção de um autômato é impraticável, pois as constantes envolvidas no tempo de execução são potencialmente grandes. O algoritmo que foi estudado nessa dissertação é tratável por parâmetro fixo e apresenta um tempo de execução limitado superiormente a $f(w, \| \varphi \|)O(|T|)$, onde T é o conjunto de nós da decomposição em árvore e w é a *treewidth* do grafo. Vamos mostrar aqui um resumo da prova apresentada em [12].

Resumo da Prova. Sejam $\mathscr{A}_1$ e $\mathscr{A}_2$ duas $\tau-$estruturas e X contido na intersecção dos universos A_1 e A_2. Se $reduza(EMC(\mathscr{A}_1, X, \varphi)) \cong reduza(EMC(\mathscr{A}_2, X, \varphi))$, então $\mathscr{A}_1$ é equivalente a $\mathscr{A}_2$ com respeito a X e φ.

Agora suponha $\mathbb{T}_X$ o conjunto de todas as τ-estruturas que contém X, isto é, X está contido no universo de tais estruturas e $\mathbb{T}_X{}^{\cong}$ o conjunto de classes de equivalência de $\mathbb{T}_X$ definida sobre a relação de equivalência acima. Então

$$|\mathbb{T}_X{}^{\cong}| \leq exp^{qr(\varphi)+1}((|X|+1)^{O(\|\varphi\|)}),$$

sendo que $\| \varphi \|$ é o comprimento de uma codificação de φ.

O *quantifier rank* $qr(\varphi)$ de uma fórmula $\varphi \in MSO(\tau)$ denota o número máximo de quantificadores aninhados em φ. Sua contagem é definida por indução em φ da seguinte forma,

$qr(\varphi) = 0$ se φ é uma fórmula atômica;
$qr(\varphi) = qr(\neg\varphi)$;
$qr(\varphi) = max\{qr(\psi_1), qr(\psi_2)\}$ se $\varphi \in \{\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2\}$; e
$qr(\varphi) = qr(\varphi) + 1$ se $\varphi \in \{\forall R\psi, \exists R\psi, \forall c\psi, \exists c\psi\}$.

Podemos considerar que $qr(\varphi)+1$ é uma constante. Observe os exemplos:

- $\forall x \exists y(adj(x,y))$ é uma fórmula de *quantifier rank* 2;
- $\forall x(x \in R) \wedge \exists y(adj(x,y))$ é uma fórmula de *quantifier rank* 1;
- $adj(x,y) \vee x \in R$ é uma fórmula de *quantifier rank* 0.

Para todo $t \in \mathbb{N}$, $exp^t(.)$ é uma exponencial iterada t vezes. Por exemplo, $exp^0(x) = x$ e $exp^t(x) = 2^{exp^{t-1}(x)}$.

Considerando uma fórmula fixa φ e um limite superior w na *treewidth* dos grafos, definimos duas constantes p e q tal que $p = qr(\varphi)+1$ e $q = \| \varphi \|$. Então

$$exp^{qr(\varphi)+1}((|X|+1)^{O(\|\varphi\|)}) \leq exp^p((w+2)^{O(q)}).$$

Esse número é uma constante e o tamanho de um jogo reduzido está limitado à essa constante, uma vez que seus jogos não possuem subjogos equivalentes entre si.

Agora considere cada $i \in T$. Temos que o tamanho de $|\mathscr{A}\mathscr{R}_i \cap X_i| = O(2^{|X_i|l})$, para $l \leq |\tau|$ e $X_i \leq w+1$, isso também é constante. Para cada $\bar{U} \in \mathscr{A}\mathscr{R}_i \cap X_i$, considere $S_i(\bar{U})$. Essa tabela só mantém jogos que não são equivalentes entre si, logo

$$|S_i(\bar{U})| \leq exp^p((w+2)^{O(q)})$$

é uma constante. E cada jogo $\mathscr{R}$ contido em $S_i(\bar{U})$ é tal que,

$$|\mathscr{R}| \leq exp^p((w+2)^{O(q)})$$

novamente uma constante. Portanto todas as operações, *reduza*, *avalia*, *combine*, *remove* e *converte*, levam tempo constante.

Para cada nó $i \in T$ o algoritmo leva tempo $f(w, \| \varphi \|) = O(exp^p((w+2)^{O(q)})$, ou seja, seu tempo de execução para T é

$$f(w, \| \varphi \|)O(|T|)$$

Como $f(w, \| \varphi \|)$ é uma constante, podemos concluir que o algortimo executa em tempo $O(|T|)$.

CAPÍTULO

4

Implementação e Experimentos

Alguns testes foram realizados como prova de conceito. Uma transcrição do algoritmo, sem tentativas de otimizações, foi realizada para validar a técnica proposta em [12] e o comportamento assintótico do tempo de execução do algoritmo.

Existem várias formas de otimizar a implementação do algoritmo. Por exemplo, os autores de [12] destacam que a função $reduza(\mathscr{G})$ foi implementada diretamente onde necessário.

O algoritmo foi implementado em *Python* e os testes foram executados no Debian GNU/Linux 8.6 sobre o processador Intel(R) Core(TM) $i7-4790$ 3.60GHz com 32GB de RAM.

Os problemas explorados nos testes foram Conjunto Dominante, Conjunto Independente Máximo e $3-$Coloração, sobre os grafos caminho, ciclo, *grid* e tripla asteroidal. Consideramos as triplas asteroidais de acordo com a Figura 4.1 e à medida que aumentamos o tamanho da tripla, o número de vértices cresce em múltiplo de 3, pois cada vértice extremo recebe um novo adjacente. Portanto o número de vértice das triplas consideradas neste trabalho é sempre um múltiplo de 3 somado com 1. Não foi utilizada uma heurística para a decomposição em árvore. A decomposição foi gerada especificamente para cada família de grafos, por isso os experimentos foram limitados apenas a algumas classes de grafos.

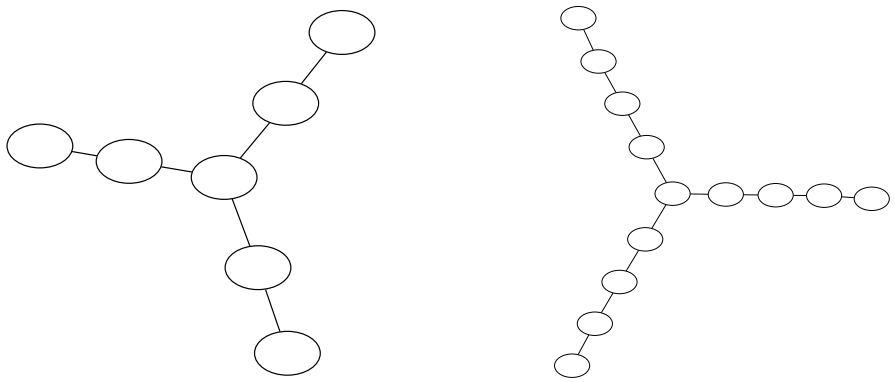
Para o problema do Conjunto Dominante utilizamos $\alpha = (1)$ e a fórmula

$$cd(S) \equiv \forall x(x \in S \ \vee \ \exists y(y \in S \wedge adj(x,y))).$$

Para o problema do Conjunto Independente Máximo utilizamos $\alpha = (-1)$ e

a fórmula

$$ci(S) \equiv \forall x \forall y(\neg(x \in S) \vee \neg(y \in S) \vee \neg adj(x,y)).$$



(a) Tripla Asteroidal com 7 vértices.

(b) Tripla Asteroidal com 13 vértices.

Figura 4.1: Exemplos de Triplas Asteroidais.

Para o problema da $3-$Coloração utilizamos $\alpha = (0)$ e a fórmula

$$\begin{aligned} 3col \equiv \exists X \exists Y \exists Z\ \forall x((x \in X \vee x \in Y \vee x \in Z) \wedge \\ (\neg x \in X \vee x \in Y) \wedge (\neg x \in X \vee x \in Z) \wedge (\neg x \in Y \vee x \in Z) \wedge \\ \forall y((\neg x \in X \vee y \in X) \wedge (\neg x \in Y \vee y \in Y) \wedge (\neg x \in Z \vee y \in Z) \vee \neg adj(x,y))). \end{aligned}$$

As fórmulas $cd(S)$ e $ci(S)$ são expressões de primeira ordem, uma vez que não apresentam quantificadores $\forall$ e $\exists$ sobre variáveis de conjunto. Porém a fórmula da $3-$coloração é uma fórmula cujos jogos têm tamanho exponencial no universo da estrutura. Sejam A e l, o universo da estrutura e o número de variáveis relacionais quantificadas, respectivamente. O número de ramificações que partem de X, Y e Z é $2^{|A|l}$. Porém nesse algoritmo, esse número depende da *treewidth* do grafo que é potencialmente menor do que A.

A Tabela 4.1 apresenta os testes para grafos caminhos, ciclos e triplas asteroidais. A coluna n indica o número de vértices do grafo e a coluna *valor* indica o valor da solução ótima encontrada pelo algoritmo. Um gráfico de n para *Tempo em Segundos* foi gerado para mostrar que a implementação se comporta de forma linear para as entradas consideradas, comprovando o tempo de execução esperado (veja a Figura 4.2).

Note que a coluna *valor* para o problema da $3-$Coloração é $\top$, ou seja, todos os caminhos com n vértices podem ter seus vértices coloridos com 3 cores, tal que cada vértice adjacente recebe cores distintas. Na verdade nos caminhos, 2 cores são suficientes, portanto isso também é verdade para 3 cores.

A Tabela 4.2 apresenta os testes para *grids*. Consideramos decomposições com largura p para as *grids* de dimensão $p \times q$. Os gráficos da Figura 4.3 foram

gerados a partir dos resultados da Tabela 4.2, porém, cada tabela apresenta resultados para *grids*, cuja largura da decomposição variam de acordo com p, sobre um mesmo problema. Por isso, cada gráfico da Figura 4.3 contém uma reta para cada p distinto, logo, cada reta representa o comportamento do algoritmo para cada largura de decomposição considerada.

Observe que o tempo de execução da nossa implementação é maior do que o apresentado no artigo [12]. Dentro do prazo para a finalização deste trabalho não foi possível nos dedicarmos à tentativa de otimização do código e consequentemente à realização de testes com grafos aleatórios, o que também exige a utilização de uma heurística para a construção da decomposição em àrvore *Nice* de tais grafos, como já foi mencionado no início deste capítulo.

Conjunto Independente Máximo em Ciclo

n	valor	execuções	Tempo em Segundos		
			min	max	médio
10	-5	3	0.84	0.90	0.87
20	-10	3	2.39	2.40	2.39
30	-15	3	4.05	4.08	4.06
40	-20	3	5.78	5.82	5.80
60	-30	3	9.19	9.28	9.22
90	-45	3	14.31	14.34	14.33
100	-50	3	15.92	15.96	15.94

Conjunto Dominante em Tripla Asteroidal

n	valor	execuções	Tempo em Segundos		
			min	max	médio
31	10	3	2.76	2.77	2.77
61	21	3	5.79	6.25	5.98
91	31	3	8.98	9.02	9.00
121	40	3	12.41	12.48	12.44
181	61	3	18.73	18.94	18.84
271	91	3	27.48	27.94	27.75
301	100	3	30.93	31.01	30.98

3-Coloração em Caminho

n	valor	execuções	Tempo em Segundos		
			min	max	médio
5	$\top$	3	3.5	3.51	3.46
6	$\top$	3	5.98	6.07	6.01
7	$\top$	3	8.87	8.99	8.93
8	$\top$	3	11.83	12.39	12.02
9	$\top$	3	14.92	15.17	15.06
10	$\top$	3	17.62	18.03	17.84
11	$\top$	3	20.9	21.07	20.91
12	$\top$	3	23.88	24.23	24.02
13	$\top$	3	26.64	27.62	26.98
14	$\top$	3	29.91	30.14	30.00
15	$\top$	3	32.99	33.39	33.25
16	$\top$	3	35.85	38.11	36.77
17	$\top$	3	38.91	39.31	39.08
18	$\top$	3	42.26	42.80	42.45
19	$\top$	3	45.08	45.25	45.17
20	$\top$	3	47.26	48.75	48.11

Tabela 4.1: Tempo de execução para grafos com n vértices.

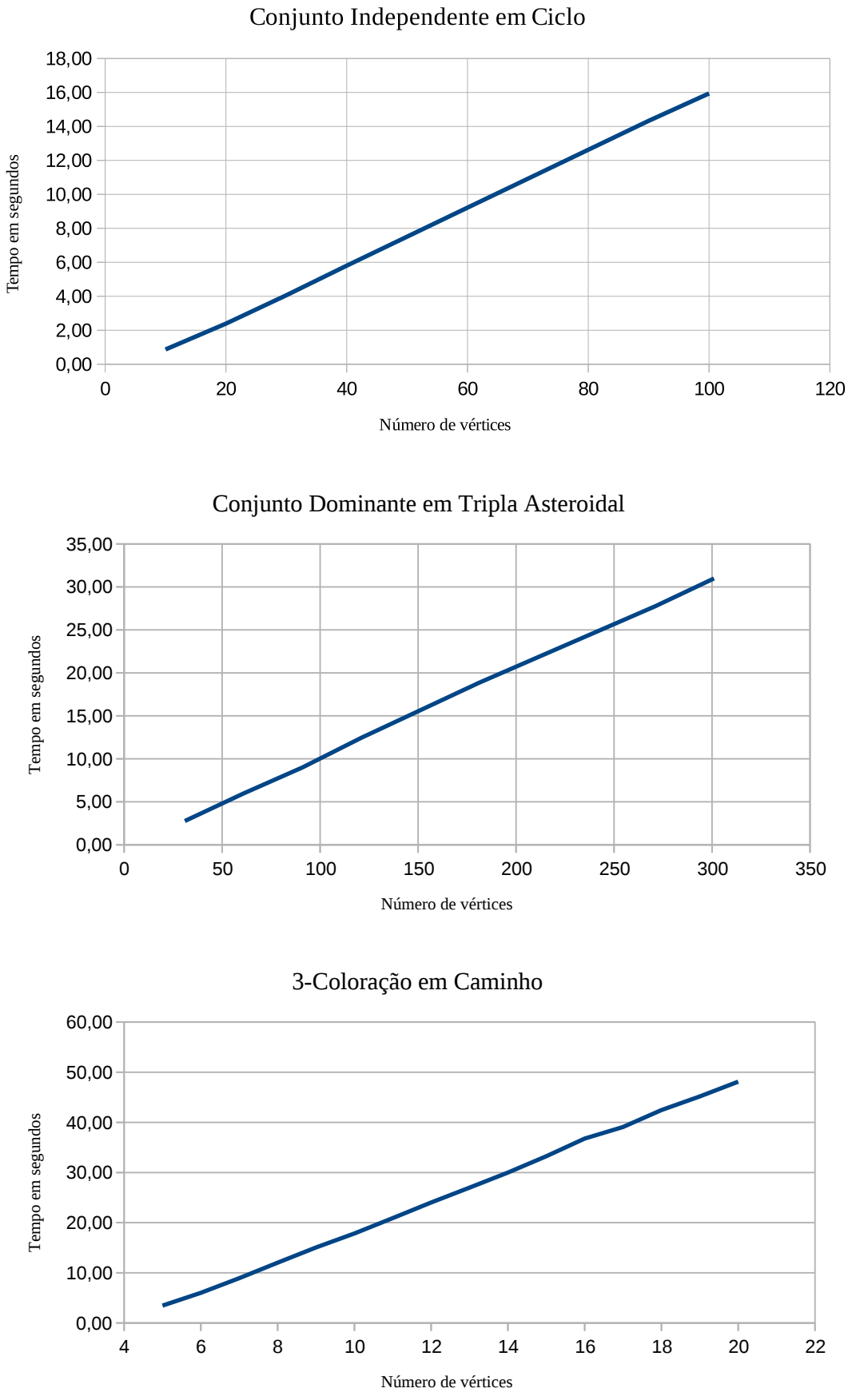


Figura 4.2: Gráfico de n para *Tempo em Segundos*.

3-Coloração

valor	n	execuções	dimensão	Tempo em Segundos		
				min	max	médio
$\top$	6	3	2×3	19.70	19.79	19.73
$\top$	8	3	2×4	39.76	40.14	39.99
$\top$	10	3	2×5	67.20	67.37	67.28
$\top$	12	3	2×6	94.65	97.54	95.90
$\top$	14	3	2×7	123.10	124.94	124.14
$\top$	16	3	2×8	151.14	151.70	151.35
$\top$	12	3	3×4	553.95	556.49	554.85
$\top$	15	3	3×5	811.15	812.87	811.90
$\top$	18	3	3×6	1065.43	1072.19	1069.31
$\top$	21	3	3×7	1334.45	1427.23	1371.26
$\top$	24	3	3×8	1601.58	1612.91	1605.51
$\top$	27	3	3×9	1869.86	1882.66	1875.90

Conjunto Dominante

valor	n	execuções	dimensão	Tempo em Segundos		
				min	max	médio
2	6	3	2×3	0,46	0,53	0,48
3	8	3	2×4	1,18	1,19	1,19
3	10	3	2×5	2,22	2,24	2,23
4	12	3	2×6	3,47	3,62	3,53
4	14	3	2×7	4,58	4,70	4,64
5	16	3	2×8	5,91	5,97	5,94
4	12	3	3×4	14,51	15,66	14,91
4	15	3	3×5	26,90	27,57	27,16
5	18	3	3×6	37,99	38,88	38,38
6	21	3	3×7	50,72	50,96	50,82
7	24	3	3×8	58,81	60,32	59,73
7	27	3	3×9	71,20	71,82	71,46
6	20	3	4×5	228,67	242,64	236,25
7	24	3	4×6	336,25	348,37	341,28
7	28	3	4×7	440,14	448,17	445,35
8	32	3	4×8	530,90	547,96	540,12
10	36	3	4×9	643,13	651,62	648,15

Conjunto Independente Máximo

valor	n	execuções	dimensão	Tempo em Segundos		
				min	max	médio
-3	6	3	2×3	0,22	0,23	0,22
-4	8	3	2×4	0,45	0,46	0,46
-5	10	3	2×5	0,70	0,71	0,71
-6	12	3	2×6	0,93	0,94	0,94
-7	14	3	2×7	1,18	1,19	1,18
-8	16	3	2×8	1,41	1,42	1,41
-6	12	3	3×4	2,24	2,36	2,28
-8	15	3	3×5	3,44	3,50	3,46
-9	18	3	3×6	4,76	5,09	4,93
-11	21	3	3×7	6,01	6,07	6,04
-12	24	3	3×8	7,03	7,08	7,05
-14	27	3	3×9	8,40	8,45	8,43
-10	20	3	4×5	14,82	15,30	15,02
-12	24	3	4×6	19,68	19,81	19,77
-14	28	3	4×7	24,87	24,99	24,91
-16	32	3	4×8	30,10	30,30	30,17
-18	36	3	4×9	35,15	35,56	35,29

Tabela 4.2: Tempo de execução para grafos *grids*.

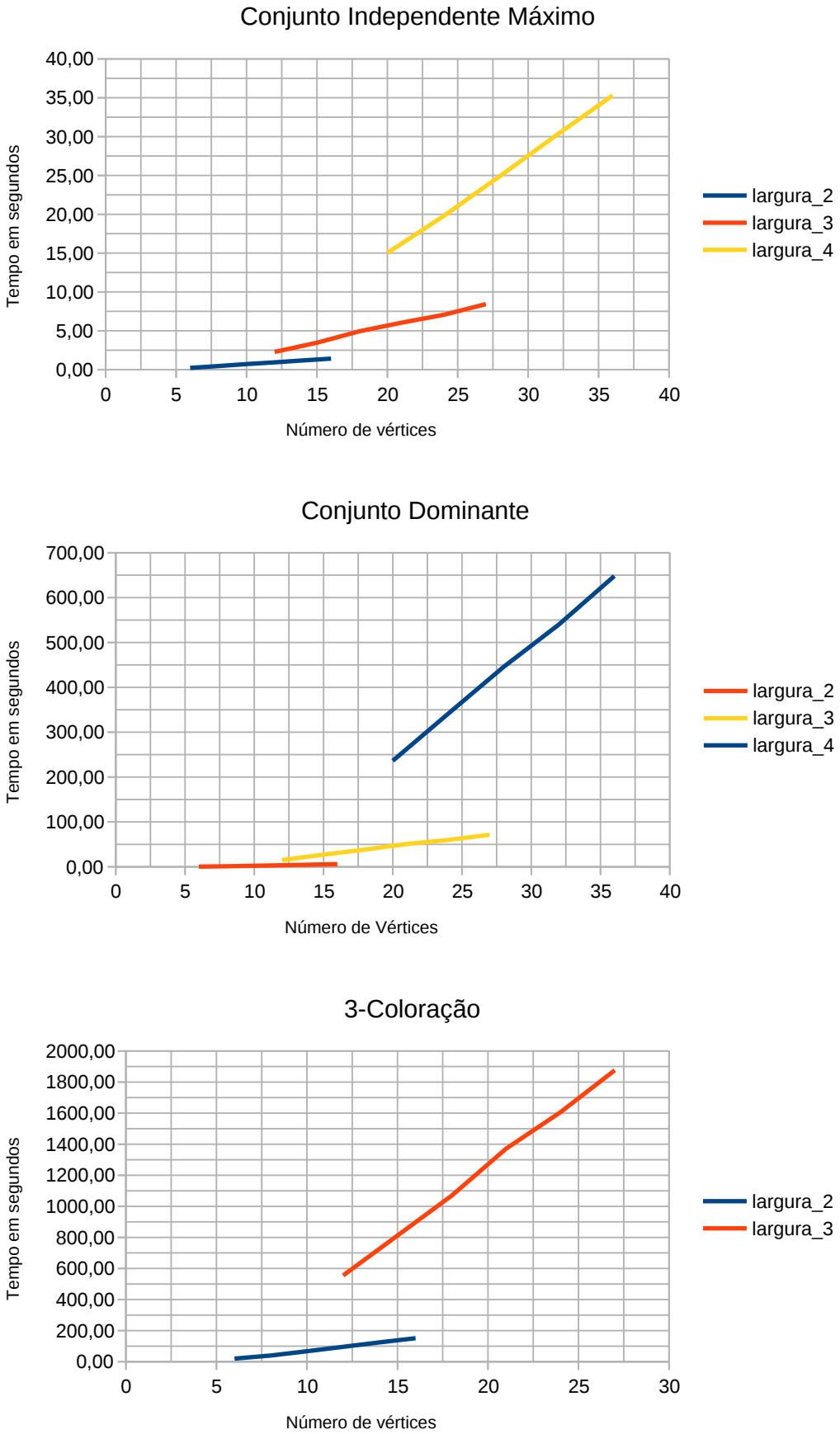


Figura 4.3: Gráfico de n para *Tempo em Segundos* utilizando *grids*. Cada reta é a representação do intervalo de valores para *grids* com a mesma largura de decomposição.

CAPÍTULO

5

Conclusões e Trabalhos Futuros

No Capítulo 2 foram estudados os conceitos sobre Decomposição em Árvore de um Grafo, Lógica Monádica de Segunda Ordem e Tratabilidade por Parâmetro Fixo, exigidos para a compreensão do algoritmo proposto em [12] que é uma construção do Teorema 1 estendido para problemas de otimização. Baseada no Modelo de Verificação com Jogos, a primeira versão da solução constrói um jogo a partir de uma expressão *MSO* e um grafo, sendo que o tamanho do jogo construído é exponencial no número de vértices do grafo, o que é impraticável considerando grafos suficientemente grandes.

No Capítulo 3 estudamos uma extensão do Modelo de Verificação que utiliza uma decomposição em árvore *Nice*. Nessa versão o algoritmo apresenta um tempo de execução linear no número de nós da árvore, quando alguns parâmetros baseados na entrada do algoritmo são fixados e limitados por um valor constante. Sua construção utiliza a técnica da Programação Dinâmica sobre a decomposição *Nice* para evitar várias computações de uma solução parcial. Pudemos verificar que a complexidade de um problema em grafos diminui para grafos com *treewidth* limitada.

No Capítulo 4 foram mostrados os resultados da implementação e pudemos confirmar as validações propostas neste trabalho.

Listamos agora algumas otimizações que podem ser feitas futuramente no algoritmo estudado;

- heurística para otimizar a poda de ramos na função $reduza(\mathscr{G})$, isto é, avaliar primeiramente os subjogos que apresentam maior probabilidade de terem uma estratégia vencedora, permitindo que os subjogos irmãos possam ser desconsiderados;

- heurística para escolher o melhor nó da Decomposição em Árvore para iniciar a transformação da decomposição em uma Decomposição *Nice* baseado no objetivo da função de otimização, isto é, se a otimização maximiza ou minimiza o problema. Um exemplo de trabalho que otimiza decomposições *Nice* pode ser visto em [15]; e

- heurística para escolher a ordem dos nós da Decomposição *Nice* a serem visitados pelo algoritmo na tentativa de determinar precocemente os $\bar{U} \in \mathscr{A}\mathscr{R}_i \cap X_i$ que invalidam a expressão. Por exemplo, quando o nó da decomposição é um *join* e $\bar{U}$ invalida a fórmula *MSO*, então em cada par de jogos de $\bar{U}$ pelo menos um jogo é falso. Se pudermos computar esse jogo antes do outro, então $\bar{U}$ já pode ser desconsiderado.

Referências Bibliográficas

[1] Arnborg, S., Corneil, D. G., e Proskurowski, A. (1987). Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284.

[2] Arnborg, S. e Proskurowski, A. (1989). Linear time algorithms for np-hard problems restricted to partial k-trees. *Elsevier Science*, 23(1):11–24.

[3] Bodlaender, H. L. (1996). A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317.

[4] Bodlaender, H. L., Fomin, F. V., Koster, A. M. C. A., Kratsch, D., e Thilikos, D. M. (2012). On exact algorithms for treewidth. *ACM Transactions on Algorithms (TALG)*, 9(1).

[5] Bodlaender, H. L. e Koster, A. M. C. A. (2007). Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269.

[6] Bollobas, B. (1979). *Graph Theory, An Introductory Course*. Springer-Verlag.

[7] Courcelle, B. (1990). The monadic second–order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation 85*.

[8] Courcelle, B. e Engelfriet, J. (2001). *Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach*. Cambridge University Press.

[9] Courcelle, B., Makowsky, J. A., e Rotics, U. (2001). On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Journal Discrete Applied Mathematics*, 108(1):23–52.

[10] Downey, R. G. e Fellows, M. R. (1995). Fixed parameter tractability and completeness i: Basic results. *SIAM Journal on Computing*, 24(4):873–921.

[11] Frick, M. e Grohe, M. (2004). The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130:3–31.

[12] Kneis, J., Langer, A., e Rossmanith, P. (2011). Courcelle's theorem - a game theoretic approach. *Discrete Opritmization 8*.

[13] Langer, A., Reidl, F., Rossmanith, P., e Sikdar, S. (2014). Practical algorithms for mso model–checking on tree–decomposable graphs. *Computer Science Review*.

[14] Robertson, N. e Seymour, P. D. (1983). Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322.

[15] van der Graaff, L. (2015). Dynamic programming on nice tree decompositions. Master's thesis, University Utrecht.