Rafael Araujo Serpa

Projeto de implementação e prototipagem de regras de manipulação de provas em dedução natural

RELATÓRIO DE PROJETO FINAL

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DEPARTAMENTO DE INFORMÁTICA

Programa de graduação em Engenharia de Computação

Rio de Janeiro
julho de 2023

Pontifícia Universidade Católica
do Rio de Janeiro

Rafael Araujo Serpa

Projeto de implementação e prototipagem de regras de manipulação de provas em dedução natural

Relatório de Projeto Final

Relatório de Projeto Final, apresentado ao programa de Engenharia de Computação da PUC-Rio como requisito parcial para a obtenção do titulo de Engenheiro de Computação.

Orientador: Prof. Edward Hermann Haeusler

Rio de Janeiro
julho de 2023

Resumo

Serpa, Rafael Araujo; Haeusler, Edward Hermann. **Projeto de implementação e prototipagem de regras de manipulação de provas em dedução natural**. Rio de Janeiro, 2023. 35p. Projeto de Graduação – Departamento de Engenharia Elétrica e Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Esse documento serve como um estudo sobre redução de provas através de algoritmos relacionados à Compressão Horizontal, contento implementações das regras de colisões entre vértices redundantes em uma demonstração proveniente de Dedução Natural

Palavras-chave

Teoria da prova, Lógica minimal, Dedução natural, Compressão Horizontal, Compressão de provas, projeto graduação

Sumário

1
Introdução

1.1
Motivação

A logica formal é um dos temas de estudo mais antigos da humanidade. A origem organizada tem raízes na Grécia antiga, com Aristóteles e Euclides, de onde surgiram s primeiras sistematizações do processo de pensamento e os primeiros conceitos basilares para o raciocínio lógico, tais como as leis da contradição e do terceiro excluído.

No final do século XIX e começo do século XX, surgiu um interesse grande em estudar de forma mais rigorosa as estruturas e propriedades que compõe a lógica formal. Uma teoria criada acerca desses conceitos é chamada de teoria da prova, que estuda sistemas dedutivos e ferramentas para provar tautologias de forma consistente. Dentre elas, podemos destacar a Dedução Natural, um sistema intuitivo para provar uma tautologia e de forma flexível à logica empregada, podendo ser descritiva, intuicionista, clássica, entre outras.

Desde então, a logica formal e teoria da prova continuaram a se expandir, principalmente em áreas como ciência da computação, inteligência artificial, linguística, filosofia e matemática. Em particular, validar uma demonstração tem sido um tema bastante desafiador devido à necessidade de diminuir o custo computacional em caso de provas extensas.

Embora esses pontos já sejam suficientes para estudarmos lógica formal e como diminuir o tamanho de uma prova, existe outro motivo ainda mais interessante que motivou este projeto. Em (Gordeev-Hermann-2019) e (Gordeev-Hermann-II-2019) é mostrado que existe um paralelo entre essas demonstrações e complexidade computacional.

A determinação de uma tautologia em um sistema de Dedução Natural é considerado um problema PSPACE-Completo, pois o tamanho da árvore gerada cresce bastante em relação ao tamanho da fórmula a ser verificada. A demonstração em (Hermann-JoseFlavio-Robinson-2023) afirma que é possível comprimir uma prova para que ela seja no máximo polinomialmente maior que a tautologia a ser demonstrada. Nesse caso, teríamos o resultado que PSPACE = NP, um dos problemas em aberto muito relevantes da computação.

1.2 Guia de leitura

Esse trabalho pode ser divido em uma parte teórica e uma parte de implementação. Foi necessário um estudo básico de teoria da prova para entender o contexto do problema e um estudo detalhado do algoritmo descrito em (Hermann-JoseFlavio-Robinson-2023) para seguir com a implementação flexível das regras de colisão entre vértices.

Na seção 2 fizemos um resumo dos principais conceitos de teoria da prova necessários para o decorrer do projeto. Mais especificamente, é feito um estudo sobre as bases de uma demonstração em dedução natural referentes à logica minimal $M \supset$. Essa fundamentação foi feita seguindo os livros de referência (Paulo-Hermann 2006) e (Pawitz 1965)

Na sequência, chegamos na seção 3, que descreve detalhadamente todo o processo de Compressão Horizontal de provas em Dedução Natural. Desde o aspecto básico de representação de uma demonstração como um grafo, até as regras de colisões e como aplicá-las em casos práticos. Foram usados de base para a construção dessa parte (José Flávio-2019) e (Hermann-JoseFlavio-Robinson-2023)

Por fim, a seção 4 faz referência à parte prática do trabalho. A partir de um código-base de implementação da Compressão Horizontal, foi possível colocar todos os aspectos estudados em prática para implementar um algoritmo que receba uma regra de colisão e seja capaz de fazer as modificações necessárias em um grafo de dedução.

2
Dedução Natural

A Dedução Natural (D.N) é um processo utilizado na lógica matemática e na teoria dos conjuntos para estabelecer a validade de argumentos. É um sistema dedutivo, ou seja, nos permite tirar conclusões lógicas a partir de um grupo de premissas e um grupo de regras.

No contexto histórico, a dedução natural pode ser vista como uma evolução do sistema dedutivo de Hilbert, que apresentava o problema de provas muito complexas e de difícil leitura. Desenvolvida por Gerhard Gentzen na década de 30 e depois aperfeiçoada por Dag Prawitz em 1965, surgiu do interesse em estabelecer um formalismo concreto em relação a conceitos já implicitamente utilizados na matemática.

Como o nome já pode sugerir, a ideia é formalizar a naturalidade com que podemos tomar conclusões logicamente. Podemos tomar o seguinte exemplo com duas premissas:

1. Se um aluno vai a escola, então ele aprende.

2. O aluno João foi a escola.

Por mais que seja simples inferir a partir dessas duas premissas que João aprendeu, o que é importante atentar nesse caso é o processo que implicitamente fizemos para chegar a essa conclusão. A D.N tem como objetivo justamente oferecer mecanismos sistemáticos que possam chegar às mesmas conclusões da lógica informal de uma maneira parecida, porém estruturada.

Com o passar dos anos, a D.N atravessou os campos da lógica e da matemática e começou a ser amplamente utilizada em outras áreas do conhecimento. Campos da ciência da computação e da filosofia, que tem um inerente interesse em formalização de processos e raciocínios, têm feito constante uso desses conceitos. A grande vantagem deste sistema dedutivo em relação a outros, é justamente o aspecto intuitivo das provas. Sua estrutura é fácil de entender, as regras são fáceis de aplicar e muitas vezes é possível conferir a corretude de uma prova de forma muito simples.

2.1 Lógica minimal

O sistema de D.N é agnóstico à lógica utilizada para fazer as inferências. Essa propriedade, faz com que seja ainda maior o seu uso, pois comporta desde deduções em lógicas simples, como a lógica minimal e algumas variações da lógica proposicional comum, até em lógicas complexas, como lógicas descritivas e intuicionistas.

Neste trabalho, estamos interessados mais em analisar a prova de uma forma estrutural do que analisar semanticamente as interpretações das premissas e das conclusões. Assim, não é relevante a lógica empregada nos processos de dedução, nos permitindo escolher uma lógica mais simples, que facilitará a leitura e o desenrolar do projeto. Nesse sentido, seguimos com a lógica minimal, mais especificamente apenas com a sua parte que contém a implicação, que podemos identificar pelo símbolo $M_{\supset}$..

Para definir sintaticamente de forma consistente a lógica minimal, precisamos definir dois componentes. O primeiro deles, o **alfabeto**, é o conjunto de todos os símbolos atômicos válidos nessa linguagem. O segundo deles, chamado **regras de formação**, são justamente as formas com que esses símbolos podem se relacionar.

Sobre o alfabeto, podemos definir em 3 tipos de símbolos:

- **Símbolos proposicionais**: [$\perp, A, B, ...,\mathrm{A}_1,A_2,...,B_1,...$]
- **Símbolos auxiliares**: [(,)]
- **Conectivos**: [$\supset, \neg$]

Chamamos uma combinação desses símbolos de **fórmula**, que representaremos com letras gregas do alfabeto. Uma fórmula é dita bem formada se foi criada a partir de uma dessas seguintes regras de formação:

1. Um símbolo proposicional é uma fórmula bem formada. Também chamamos esse caso de fórmula atômica.

2. Se α é uma fórmula bem formada, então $\neg\alpha$ é uma fórmula bem formada.

3. se α e β são fórmulas bem formadas, então $\alpha \supset \beta$ é uma fórmula bem formada

Por simplicidade, de agora em diante, estaremos dizendo fórmulas no lugar de fórmulas bem formadas.

Ao definir sintaticamente uma fórmula válida, queremos também poder dizer que uma fórmula é válida semanticamente. Para tal, precisamos de regras

básicas, chamadas axiomas, que nos permitam avaliar uma fórmula. Existem vários conjuntos de axiomas possíveis e equivalentes para a lógica minimal, mas não estamos interessados em definir axiomas, visto que estamos interessados nas regras atreladas ao processo de dedução natural. A título de exemplo, segue uma possível definição de axiomas para a lógica minimal.

1. Para qualquer fórmula α, temos que $\alpha \supset \alpha$
2. Se $\alpha \supset \beta$ e $\beta \supset \gamma$ então $\alpha \supset \gamma$
3. Se $\alpha \supset \beta$ e $\neg\beta \supset \gamma$ então $\alpha \supset \gamma$

2.2 Descarte de hipótese

Nessa seção, vamos mostrar como funciona um processo importante no sistema de D.N, principalmente no contexto apresentado da lógica $M_{\supset}$. Consideremos o seguinte exemplo:

– Se x é um número par, então x^2 também será um número par.

Queremos verificar a validade dessa afirmativa. Podemos perceber que estamos em uma proposição da forma $A \supset B$, onde A corresponde a "x é um número par"e B corresponde a "x^2 é um número par". O primeiro passo, naturalmente, seria assumir que x é um número par, ou seja, vamos começar com uma hipótese. Sendo assim, sabemos que existe um k tal que $x = 2k$. Consequentemente, $x^2 = (2k)^2 = 4k^2 = 2(2k^2)$ e podemos ver que x^2 também é par. Dessa forma, para demonstrar uma expressão da forma $A \supset B$, o primeiro passo foi supor A verdadeiro por hipótese e usar essa informação para obter B. Depois desse passo, não é mais importante qualquer declaração feita sobre A, ou seja, a hipótese não é mais relevante para conclusões futuras. Esse processo é conhecido como **descarte de hipótese**

2.3 Regras em Dedução Natural

Como já foi mencionado anteriormente, o processo de D.N procura funcionar de forma parecida com a nossa intuição. Sendo assim, para chegar a uma demonstração, vamos seguindo em etapas, de forma muito parecida com o que foi feito no exemplo anterior. Para seguirmos essas etapas, precisamos definir regras, que são os mecanismos para evoluir o raciocínio. Vamos ver que essas regras estão intimamente ligadas aos conectivos lógicos, onde procuramos incluí-los ou excluí-los de acordo com o interesse da prova.

Voltando ao exemplo anterior: *se x é par, então x^2 é par*. Seguindo a mesma lógica que foi apresentada anteriormente, nós temos premissas A e B e queremos obter a conclusão $A \supset B$. Em outras palavras, o nosso interesse é introduzir o conectivo $\supset$, e para isso fizemos o processo de descarte de hipótese. Em termos esquemáticos, no contexto da D.N, podemos definir essa regra da forma:

$$\frac{\begin{array}{c}[A] \\ \vdots \\ \mathrm{B}\end{array}}{A \supset B} \ (\supset \text{-Introdução})$$

Esse diagrama, que deve ser lido de cima para baixo, representa o processo em etapas que foi feito para demonstrar o exemplo em questão. O primeiro passo foi assumir A por hipótese, assim colocamos o símbolo $[A]$ no topo e entre colchetes, pois se trata de uma hipótese. A partir disso, desenvolvemos nosso raciocínio aritmético, que estamos simbolizando pelos pontos, e chegamos em B. Nesse momento, pudemos finalmente tirar a conclusão $A \supset B$, que por ser uma conclusão, colocamos abaixo de um traço horizontal. Esse processo de começar com premissas e concluir com o conectivo $\supset$ é usado repetidamente ao longo das provas, por isso chamamos de regra $\supset -I$.

Por outro lado, podemos pensar na possibilidade de querermos excluir o conectivo $\supset$. Isto posto, podemos pensar na frase "Se o aluno vai a escola, então ele aprende"e podemos tentar chegar à alguma conclusão a partir dela. Assim como notamos na seção 2, quando este exemplo foi utilizado, só conseguimos tirar uma conclusão caso o aluno tenha ido à escola. Por outro lado, se ele tiver apenas aprendido, não conseguimos descobrir se ele foi à escola ou se aprendeu por outros meios, mas no caso em que ele foi à escola, certamente podemos concluir que ele aprendeu. Chamando "o aluno vai à escola"de A e "o aluno aprende"de B, podemos traduzir toda a sentença na fórmula $A \supset B$. Assim como fizemos na regra $\supset I$, podemos traduzir essa sequência lógica no seguinte diagrama:

$$\frac{A \qquad A \supset B}{B} \ (\supset \text{-Eliminação})$$

Nesse caso, chamamos "A"de premissa menor e "$A \supset B$"de premissa maior. Dessa forma forma, analogamente ao processo intuitivo, conseguimos eliminar o conectivo $\supset$ utilizando duas premissas e concluindo uma nova fórmula, gerando assim a regra $\supset -E$. Essas duas regras vão constituir os mecanismos que serão encontrados nas provas ao longo do trabalho,

2.4 Exemplo de prova

Em posse das regras básicas da lógica minimal, podemos agora deduzir qualquer fórmula válida através dos processos de D.N, basta combiná-las de forma conveniente.

Exemplo 2.1 Uma prova de $(A \supset (B \supset C)) \supset (B \supset (A \supset C))$

Para resolver esse problema, precisamos pensar de uma forma parecida com a forma que foram deduzidas as regras de introdução e eliminação. Podemos prosseguir sequencialmente nos seguintes passos:

1. Para mostrar que $(A \supset (B \supset C)) \supset (B \supset (A \supset C))$, vamos primeiro assumir $(A \supset (B \supset C))$ por hipótese.
2. Em seguida, tendo $(A \supset (B \supset C))$, vamos assumir A por hipótese.
3. Com as duas fórmulas do item anterior, podemos aplicar uma regra de eliminação, ficando com $(B \supset C)$
4. De posse da fórmula $(B \supset C)$, podemos assumir B por hipótese.
5. Com as duas fórmulas do item anterior, podemos aplicar mais uma regra de eliminação, ficando com C
6. Descarregando a hipótese A, ficamos com a fórmula $A \supset C$
7. Descarregando agora a hipótese B, chegamos à fórmula $B \supset (A \supset C)$
8. Por fim, descarregamos a primeira hipótese $(A \supset (B \supset C))$ chegando no resultado final $(A \supset (B \supset C)) \supset (B \supset (A \supset C))$

Como descarregamos todas as hipóteses corretamente e chegamos no resultado desejado, podemos concluir que a fórmula é válida. Podemos representar esses passos da demonstração no seguinte diagrama:

$$\dfrac{\dfrac{\dfrac{[B]^3 \qquad \dfrac{[A]^2 \qquad [A \supset (B \supset C)]^1}{B \supset C}\,(\supset\text{-E})}{\mathrm{C}}\,(\supset\text{-E})}{A \supset C}\,(\supset -I^2)}{B \supset (A \supset B)}\,(\supset -I^3)}{(A \supset (B \supset C)) \supset (B \supset (A \supset C))}\,(\supset -I^1)$$

Exemplo 2.2 Demonstre $A \supset C$ a partir de $A \supset B$ e de $B \supset C$. Podemos demonstrar diretamente no seguinte diagrama:

Nesse caso temos duas fórmulas verdadeiras já dadas, que seriam $A \supset B$ e $B \supset C$

$$\dfrac{\dfrac{B \supset C \qquad \dfrac{[A]^1 \qquad A \supset B}{B}\,(\supset\text{-E})}{\mathrm{C}}\,(\supset -I^1)}{A \supset C}$$

3
Compressão Horizontal

Vimos anteriormente que a D.N foi um sistema criado para refletir o processo de pensamento intuitivo, utilizado para validar argumentos. Embora ela seja capaz de cumprir esse papel com sucesso, com o avanço do estudo sobre provas formais, começaram a surgir outras necessidades para a manipulação desses resultados.

Embora para exemplos simples seja bem agradável ler a prova através de um diagrama obtido pelo processo de D.N, ao aumentarmos a fórmula a ser validada, o tamanho da prova tende a crescer de forma exponencial. Uma prova tão complexa acaba perdendo o aspecto intuitivo, não permitindo a mesma fluides de leitura que poderia ser percebida anteriormente.

Com o passar dos anos, o interesse em lógica formal se atrelou bastante à computação. Dessa forma, a necessidade de trabalhar com provas reduzidas aumentou consideravelmente, com objetivo de reduzir a complexidade computacional, tanto no aspecto de memória quanto no aspecto dos algoritmos.

Nesse contexto que a técnica de Compressão Horizontal(C.N) emergiu como uma solução. Esse processo utiliza diversas técnicas bem definidas para reduzir o tamanho das demonstrações, eliminando redundâncias e simplificando passos. Essencialmente, a ideia é juntar etapas feitas paralelamente em uma etapa só.

É importante dizer que a C.N vem sido constantemente desenvolvida ao dos últimos anos. Diversas outras abordagens podem ser sugeridas ou até combinadas conforme seja o interesse da redução. Nesse capítulo, apresentaremos o modelo de compressão que foi estudado para interesse deste trabalho.

3.1
Conceitos de grafos

Para conseguirmos manipular uma demonstração de forma apropriada, precisamos de uma estrutura de dados que seja compatível com as informações ali existentes. Uma forma natural de fazer esse mapeamento é pensar no modelo de diagrama de D.N como um grafo, em que os vértices seriam equivalentes aos passos e as arestas equivalentes às relações entre eles. A seguir algumas definições básicas sobre grafos:

Definição 3.1 *Um grafo é um par $G = (V, A)$ de conjuntos em que $V = \{v_1, \ldots, v_n \mid \forall i, j v_i \neq v_j\}$ é chamado de **vértices** do grafo e A é um subconjunto de $V \times V$, chamado de **arestas** do grafo. Seus elementos são chamados de vértices e de arestas, respectivamente.*

Definição 3.2 *Um grafo $G = (V, A)$ é **direcionado** quando para todo $v_1, v_2 \in V$ e $(v_1, v_2) \in A$ temos que $(v_1, v_2) \neq (v_2, v_1)$.Em outras palavras, nesse caso o par de arestas é ordenado.*

Definição 3.3 *Um grafo $G = (V, A)$ é **rotulado por vértices** se existe um conjunto de rótulos R e uma função $r : V \to R$. Analogamente, chamamos de **rotulado por arestas** se existe um conjunto de rótulos R e uma função $r : A \to R$.*

Definição 3.4 *Um grafo $G = (V, A)$ tem **arestas coloridas** se existe um conjunto de cores $C = \{c_1, \ldots, c_n\}$ de cores e uma função $c : A \to C$.*

Definição 3.5 *Um **caminho** em um grafo $G = (V, A)$ é um conjunto $U = \{u_1, \ldots, u_n\} \subset V$ tal que $\forall i < n$ temos que $(u_i, u_{i+1}) \in A$.*

Definição 3.6 *Um grafo $G = (V, A)$ é chamado de **árvore** se $\forall v_1, v_2 \in V$ só existe um único caminho entre v_1 e v_2.*

Definição 3.7 *Em um grafo $G = (V, A)$, um vértice $v \in V$ é chamado de **folha** se não existe aresta incidente em v*

Definição 3.8 *Seja um grafo $G = (V, A)$ e um vértice $v \in V$, chamamos de **grau de entrada** de v ou $ge(v)$ a quantidade de arestas $\in A$ que possuem v na segunda coordenada do par ordenado. Analogamente, chamamos de **grau de saída** de v ou $gs(v)$ a quantidade de arestas $\in A$ que possuem v na primeira coordenada do par ordenado*

3.2 Provas como grafos

A figura 3.1 mostra uma representação da demonstração do segundo exemplo da seção 2.4 na forma de grafo. Esquematicamente, é perceptível a semelhança, fazendo com que a conversão seja intuitivamente simples. Podemos destacar alguns pontos a respeito dessa representação em grafo.

- Uma prova em D.N tem um fluxo definido de leitura, que é de cima para baixo. Na forma de grafos, queremos também contemplar esse fluxo já definido, então utilizaremos o grafo direcionado.
- Temos dois tipos distintos de arestas. A linha pontilhada representa a aresta de descarte, que chamaremos de A_d. As linhas contínuas representam as arestas de dedução A_D. Enquanto as arestas de descarte se referem às descartes de hipóteses, as arestas de dedução indicam os novos passos do fluxo da prova.
- Se considerarmos apenas as arestas de dedução, o grafo formará uma árvore.
- O grafo é rotulado nos vértices. Para representar corretamente os passos da demonstração, usaremos exatamente a fórmula correspondente. Dessa forma, teremos uma bijeção entre as fórmulas do diagrama em D.N e o número de vértices da representação em grafo.
- Para cálculo dos graus de entrada e de saída, pensaremos apenas nas arestas de dedução

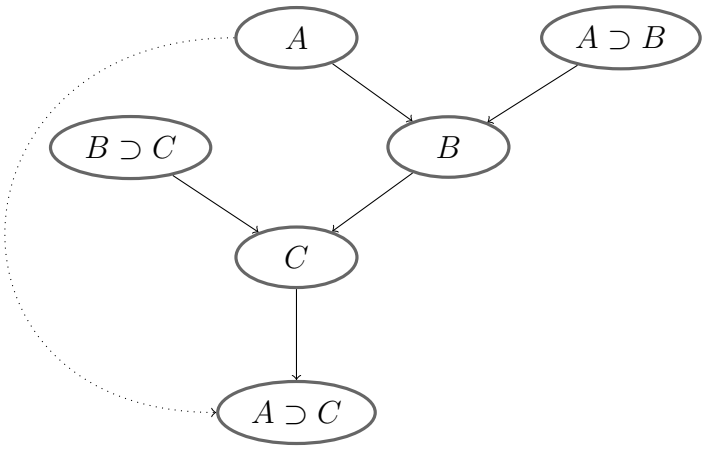
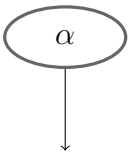


Figura 3.1: Um exemplo de grafo de prova

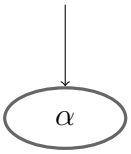
3.2.1 Classificação dos vértices

Como vimos anteriormente, o processo de D.N conta com sucessivas aplicações das regras de introdução e exclusão de implicações. Dessa maneira, existe um número limitado de formas que um vértice pode aparecer no nosso grafo. Podemos listar os seguintes casos:

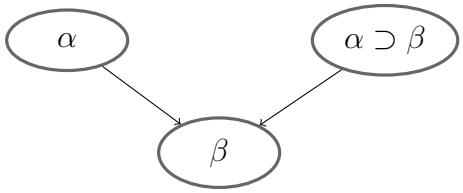
1. **Hipótese**. Nesse caso teremos uma folha no grafo, ou seja, sendo v um vértice que representa uma hipótese, teremos que $ge(v) = 0$ e $gs(v) = 1$



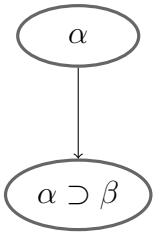
2. **Raiz**.Chamamos de raiz o vértice v que representa a conclusão da prova. Dessa forma teremos $gs(v) = 0$



3. $\supset$ **Eliminação**. Um vértice v pode ser originário de uma regra de eliminação. Assim, teremos $ge(v) = 2$ e $gs(v) = 1$



4. $\supset$ **Introdução**. Outra possibilidade é o vértice v provir de uma regra de inclusão. O vértice v teria $ge(v) = 1$ e $gs(v) = 1$



3.2.2 Conjunto de dependência

Para simplificar o nosso grafo, seria conveniente se não precisássemos trabalhar com esses dois conjuntos de arestas diferentes. Assim, poderíamos pensar em tirar as arestas de descarte, já que as arestas de dedução são imprescindíveis para organizar o fluxo de leitura. Entretanto, para conseguir verificar a validade da demonstração, precisamos identificar se todas as hipóteses foram corretamente descarregadas, que é justamente a informação proveniente das arestas de descarte.

Podemos pensar que as arestas de descarte representam uma dependência que a demonstração tem daquela hipótese até o momento que ela é descarregada. Embora estejamos colocando essa informação diretamente no ponto em que existe a regra de introdução, assim como é feito no diagrama de D.N, é possível perceber que todo o caminho entre esses dois vértices possui essa mesma dependência. A ideia, portanto, é conseguir colocar esse conteúdo diretamente nas arestas de dedução.

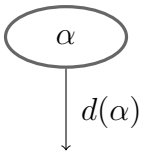
Um conceito bastante utilizado em matemática e em computação é que existe uma bijeção entre um subconjunto dos números naturais e uma sequência binária. A bijeção se dá colocando 0 nos elementos que não estão presentes no conjunto e 1 nos elementos que estão presentes no conjunto. Imagine o subconjunto dos números pares, podemos representá-lo em uma sequência binária como 1010101...

Definição 3.9 *Seja uma ordenação qualquer de todas as fórmulas presentes em um grafo chamamos de* ***conjunto de dependência*** *a sequência binária que representa todas as hipóteses ainda não descarregadas.*

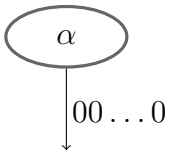
Exemplo 3.1 Seja a fórmula $f = (A \supset B) \supset ((B \supset C) \supset (A \supset C))$. Podemos propor como conjunto das subfórmulas de f sendo $F = \{A, B, C, A \supset B, A \supset C, B \supset C, (B \supset C) \supset (A \supset C)), (A \supset B) \supset ((B \supset C) \supset (A \supset C))\}$. Assim, o conjunto de dependência que representa as hipóteses $G = \{A, B, B \supset C\}$ é $d(G) = 11000100$

A seguir, mostraremos como podemos determinar o conjunto de dependência em um grafo de dedução. Colocaremos essa propriedade como rótulo das arestas e exploraremos todos os casos como na seção anterior.

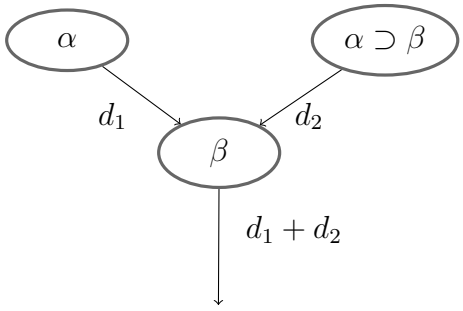
1. **Hipótese**. Aqui estamos inserindo uma dependência nova na aresta, como estamos em uma folha e, portanto, não estamos carregando dependências passadas, a aresta terá valor $d(\alpha)$



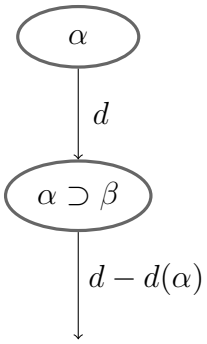
2. **Raiz**.Aqui todas as hipóteses já foram descarregadas, não há mais dependências na prova



3. $\supset$ **Eliminação**. Estaremos adicionando tanto as dependências da esquerda quanto da direita. A esquerda sendo $d(\alpha)$ e a direita sendo $d(\alpha \supset \beta)$, usaremos a notação $d(\alpha) + d(\alpha \supset \beta)$, pois a operação de junção seria análoga à soma de vetores em $\mathbb{R}^n$



4. $\supset$ **Introdução**. Nesse caso, estaremos fazendo uma descarte de hipótese. Dessa forma, estaremos retirando uma dependência da demonstração. Usaremos o sinal de $-$, pois a operação de retirada é análoga a subtração de vetores em $\mathbb{R}^n$



Exemplo 3.2 Seja a fórmula $f = (A \supset B) \supset ((B \supset C) \supset (A \supset C))$ e a ordenação de subfórmulas de f sendo $F = \{A, B, C, A \supset B, A \supset C, B \supset C, (B \supset C) \supset (A \supset C)), (A \supset B) \supset ((B \supset C) \supset (A \supset C))\}$. O grafo de dedução ficaria como segue:

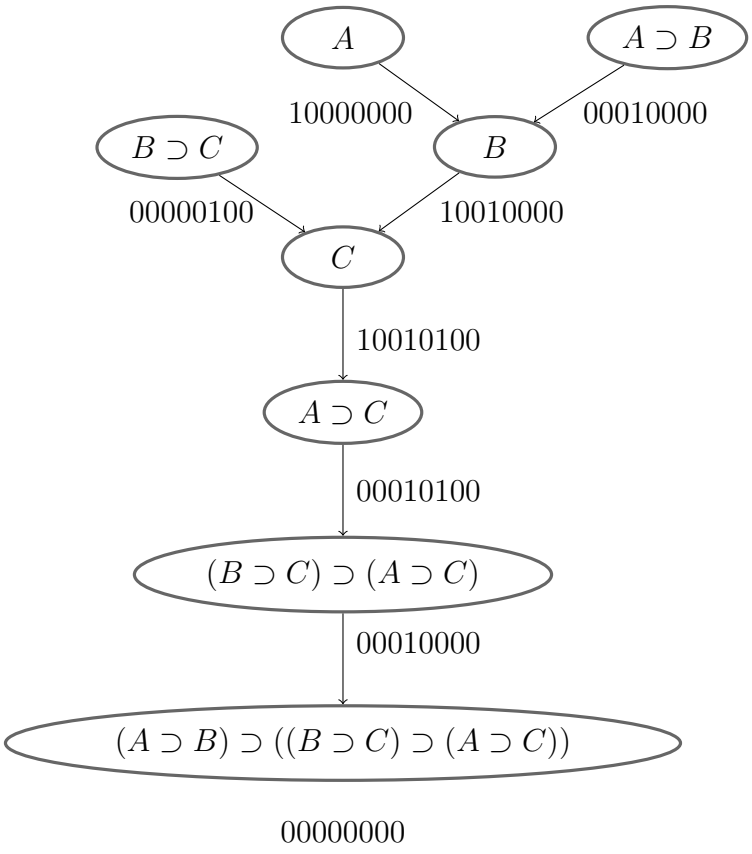


Figura 3.2: Grafo de prova com conjunto de dependência

3.3 O algoritmo de Compressão Horizontal

Após definir os conceitos do grafo de dedução, podemos começar a discussão em como podemos reduzi-lo. A estratégia que usaremos aqui consiste em diminuir o número de vértices, juntando alguns passos que possam ser redundantes. É possível perceber que, embora o processo de D.N siga um fluxo muito bem definido de cima para baixo, os passos dentro de uma mesma altura são independentes. Nesse sentido, gostaríamos de aglutinar vértices de mesmo nível, desde que tenham exatamente o mesmo rótulo.

A ideia do algoritmo é percorrer o grafo de baixo para cima, procurando vértices de rótulos idênticos para cada nível. Ao acharmos tal configuração, vamos colapsar dois a dois todos os vértices semelhantes nessa mesma altura. Um pseudocódigo que ilustra o mecanismo do algoritmo pode ser visto na figura abaixo.

Algorithm 1 Pseudocódigo Compressão Horizontal

```
Nivel ← 1
Altura ← n
while Nivel < Altura do
    ListaIdênticos(Nivel)                ▷ Lista de vértices semelhantes
    for u, v em ListaIdênticos do
        Collapse(u, v)
    end for
    Nivel ← Nivel + 1
end while
```

3.3.1 Arestas de ancestralidade

Pensando mais atentamente sobre o processo de junção de vértices, podemos encontrar algumas dificuldades. A figura abaixo mostra uma sugestão de como poderia funcionar intuitivamente esse processo. No lado esquerdo, temos vértices U e V numa mesma altura, que estamos supondo por conveniência que tem rótulos idênticos.No lado direito, temos esses mesmos vértices colapsados, preservando as suas conexões de entrada e saída.

O problema é que tendo posse apenas do lado direito, não conseguimos fazer o processo reverso. Ou seja, não conseguimos distinguir de quem eram as premissas e as conclusões nem perceber se os vértices eram regras de introdução ou de inclusão. Voltar à situação original é importante para conferirmos se a redução foi bem feita e para conseguirmos validar a demonstração.

Nesse sentido, podemos adotar uma técnica de reconstrução. No grafo de dedução inicial, iremos colorir todas as arestas dedutivas com a cor 0. Ao colapsarmos dois vértices, podemos colorir um dos fluxos de saída de uma cor diferente, de forma com que consigamos identificar de forma única as duas conclusões. Resta então definir as premissas, e assim nós usaremos as chamadas **arestas de ancestralidade**. Essas arestas terão uma cor única e vão indicar o caminho que foi feito. A figura abaixo mostra um exemplo do uso de arestas de ancestralidade.

Com essas novas arestas, fica bem clara a visualização do fluxo que foi comprimido, podendo assim reconstruir o grafo original. Nota-se também que as arestas de ancestralidade possuem rótulos que correspondem à sequência de cores do caminho original.

Muitas questionamentos podem surgir a partir deste novo conceito. Pode-se imaginar o caso em que a aresta de ancestralidade está chegando no vértice a ser colapsado, a possibilidade do vértice já ter sido colapsado e precisar ser

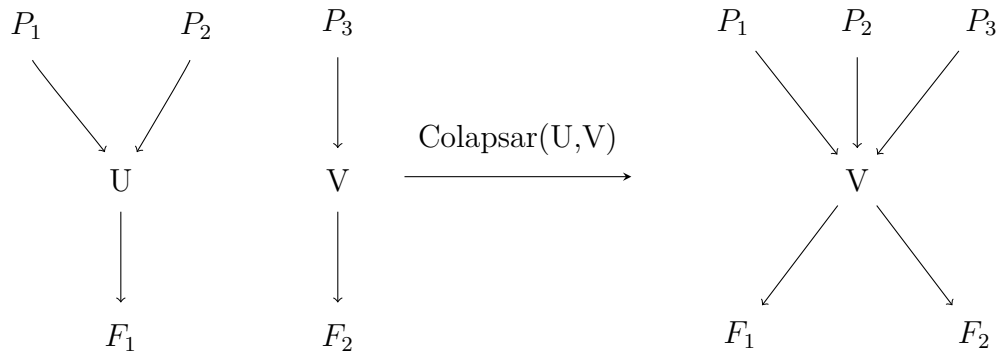


Figura 3.3: Colapso errado entre u e v

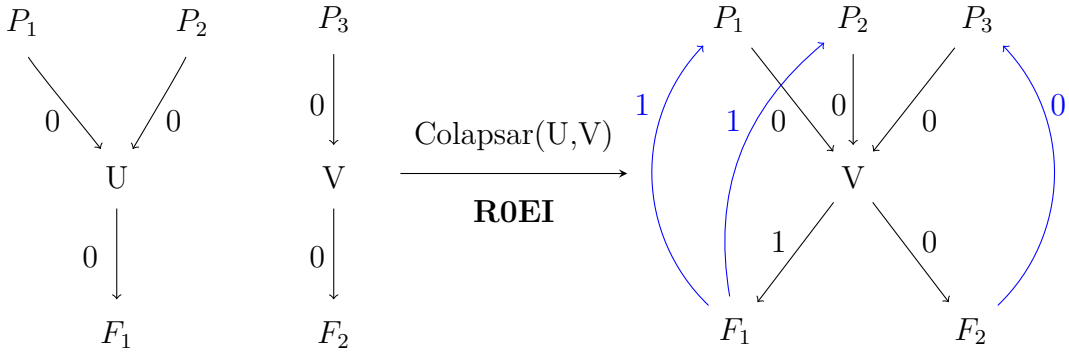


Figura 3.4: Vértices colapsados com arestas de ancestralidade

novamente, ou até mesmo a situação em que os vértices possam ter a mesma aresta.

Por fim, as arestas de ancestralidade são uma adição vital para a representação do grafo comprimido, mas aumentam muito o grau de complexidade do ponto de vista teórico e de implementação. Precisamos definir uma série de regras para comportar bem todas as colisões possíveis.

3.4 Classificação das regras

Nesta seção, iremos definir todas as regras que serão usadas para colapsar dois vértices de rótulos idênticos numa mesma altura de um grafo de dedução. Usaremos os conceitos apresentados na seção sobre provas em grafos pra reduzir o número de possibilidades apenas nos casos de interesse.

Ao todo, temos 36 regras mapeadas, distribuídas em 4 grupos principais de acordo com características fundamentais que se repetem em alguns casos. A proposta é classificar as regras da forma Rim_em_r, em que $i \in \{0, 1, 2, 3\}$ faz menção ao tipo da regra. Os parâmetros $m \in \{H, E, I, X\}$, cujos índices

se referem à esquerda e direita, correspondem ao modo como aquele vértice aparece no grafo de dedução. Como vimos, um vértice pode surgir como hipótese, regra de eliminação e regra de introdução, que podemos mapear como H,E e I respectivamente. A novidade surge com o parâmetro X, que seria quando esse vértice já foi alvo de algum colapso, caso que será frequente no processo de compressão.

Sem perda de generalidade, iremos omitir casos simétricos e casos análogos, não tendo necessidade de escrever explicitamente as 36 regras.

3.4.1 Regras do tipo 0

Aqui definiremos os casos ainda intocáveis, ou seja, os dois vértices ainda não tiveram influência alguma do processo de compressão. Teremos 9 casos neste tipo que são:

R0HH,**R0IH**,**R0HI**,**R0EH**,**R0HE**,**R0II**,**R0EI**,**R0IE**,**R0EE**

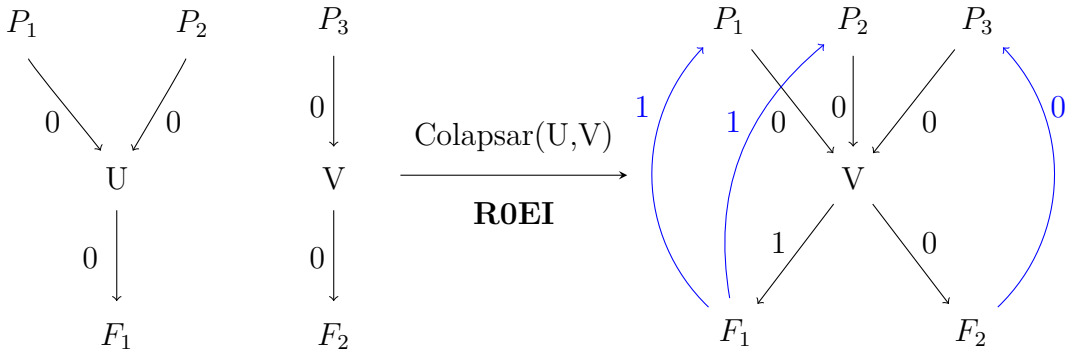


Figura 3.5: Aplicação da regra R0EI

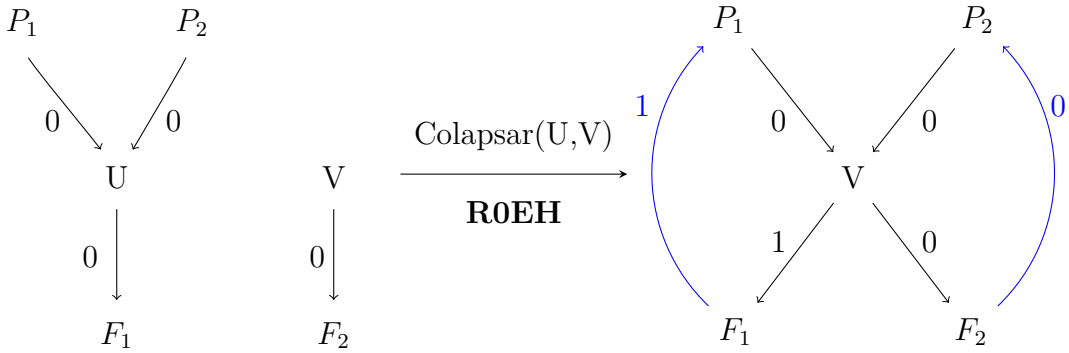


Figura 3.6: Aplicação da regra R0EH

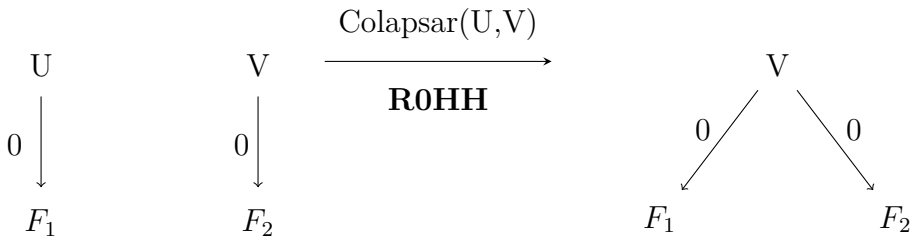


Figura 3.7: Aplicação da regra R0HH

3.4.2 Regras do tipo 1

As regras do tipo 1 focam nos cenários em que um vértice já é fruto de alguma junção passada. É importante notar que esse vértice sempre será o da esquerda, devido à natureza do algoritmo de D.N. Também podemos destacar que ele pode ter mais de um filho, com arestas de valores de cores variando de 0 até n. Para exemplificar, denotaremos j como a aresta de cor com valor mais alto e i como uma aresta de cor genérica. Temos três casos de regras do tipo 1:

R1XH,**R1XI**,**R1HE**

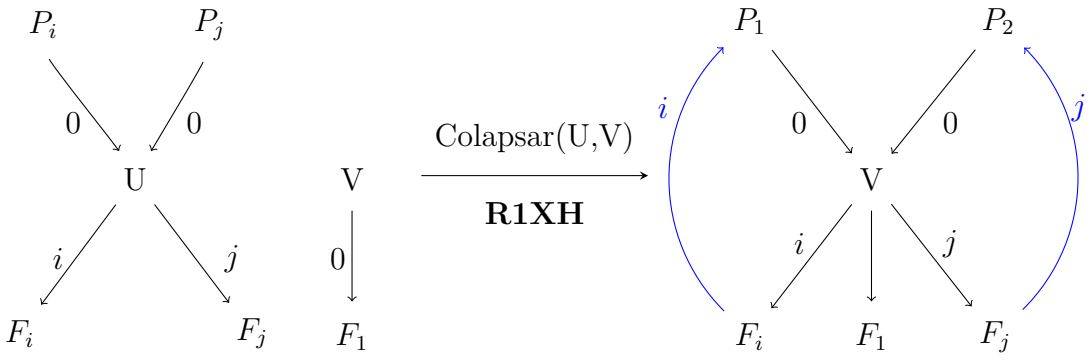


Figura 3.8: Aplicação da regra R1XH

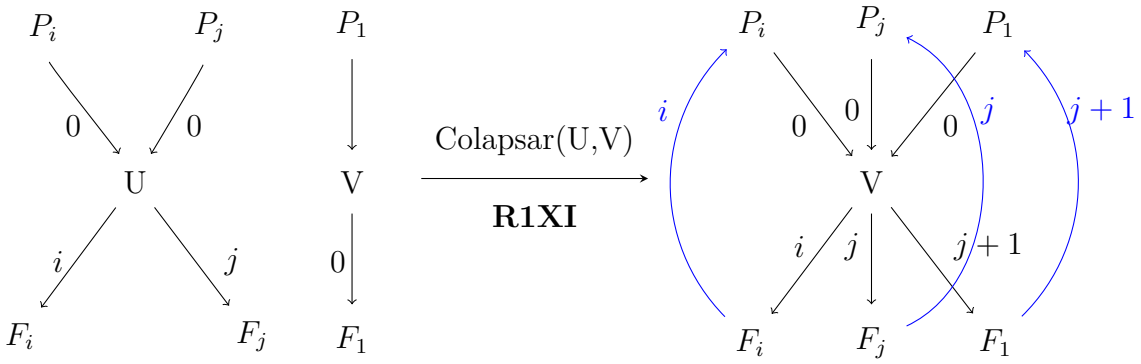


Figura 3.9: Aplicação da regra R1XI

3.4.3 Regras do tipo 2

Agora estamos interessados nos cenários em que pelo menos um dos vértices tem arestas de ancestralidade chegando nele, mas não são resultados de colapsos anteriores. Nesta situação há uma possibilidade dos vértices a serem colapsados compartilharem a mesma aresta, no caso de terem sido pais de vértices que se aglutinaram. Assim, separaremos os casos desse grupo em dois, denotando o índice e para aqueles que tem aresta comum e o índice v caso contrário. No primeiro caso teremos que colapsar também essa aresta, denotando a cor especial λ. São mapeados os seguintes casos:

R_e2HH,**R_e2IH**,**R_e2EH**,**R_e2II**,**R_e2IE**,**R_e2EI**,**R_e2HI**,**R_e2HE**, **R_e2EE**,**R_v2HH**,**R_v2IH**,**R_v2EH**,**R_v2II**,**R_v2IE**,**R_v2EI**,**R_v2HI**,**R_v2HE**, **R_v2EE**

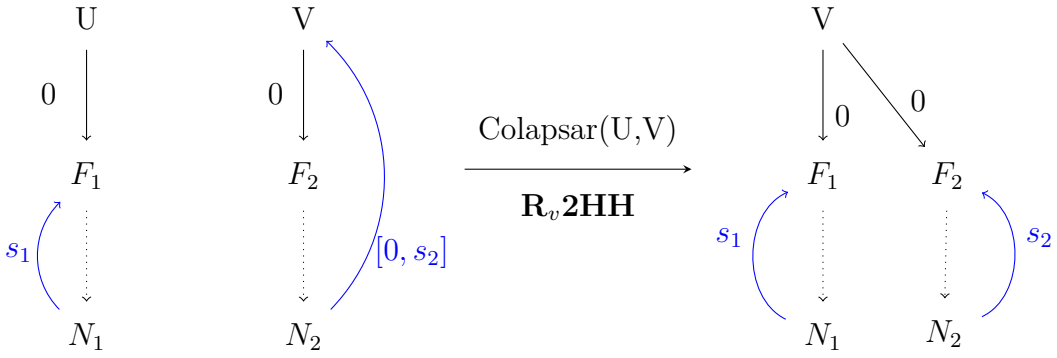


Figura 3.10: Aplicação da regra R_v2HH

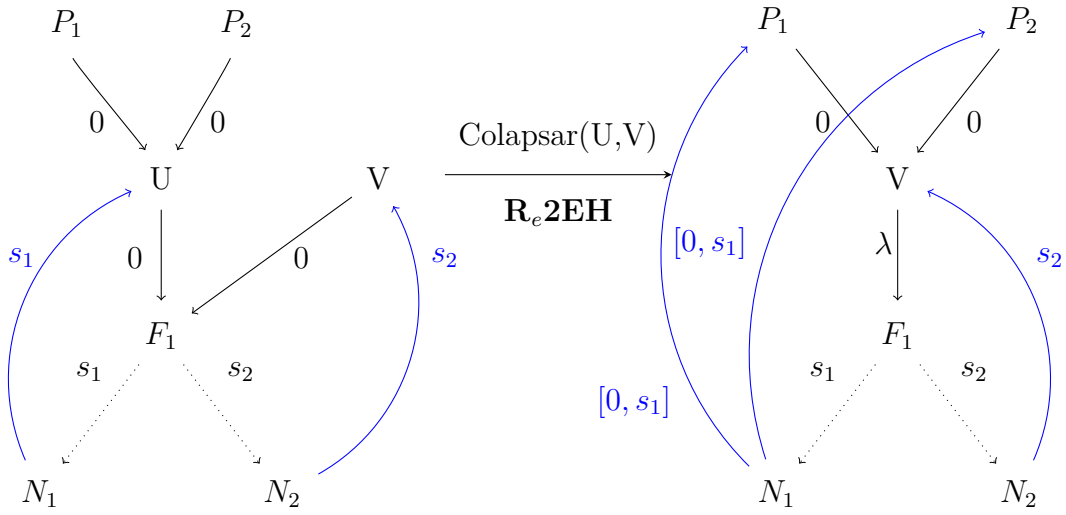


Figura 3.11: Aplicação da regra R$_e$2EH

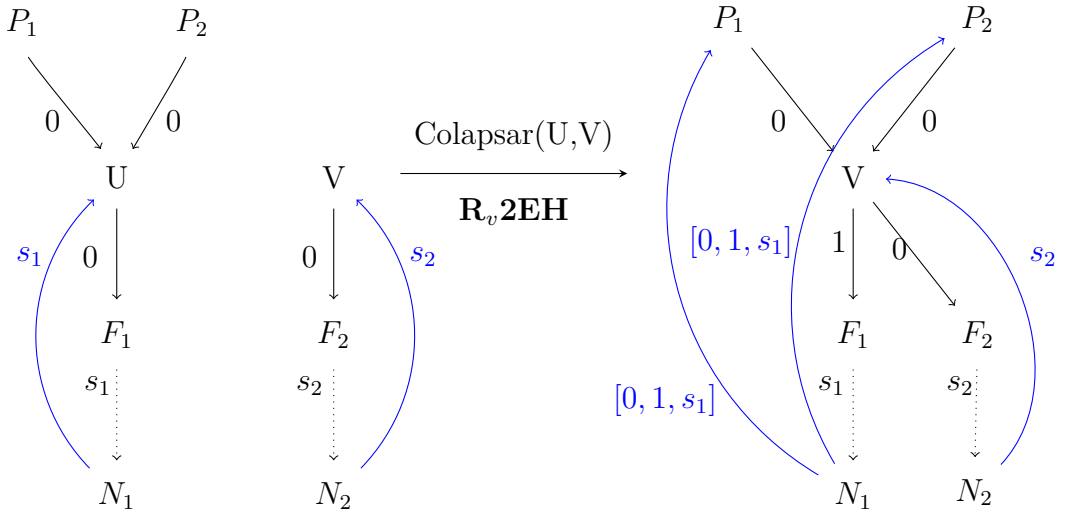


Figura 3.12: Aplicação da regra R$_v$2EH

3.4.4 Regras do tipo 3

Por fim, as regras do tipo 3 comportam as situações em que existe pelo menos uma aresta de ancestralidade chegando nos vértices a serem colapsados, mas o vértice da esquerda já foi alvo de alguma junção anterior. As regras dentro desse contexto são:

R$_e$3XH,**R$_e$3XI**,**R$_e$3XE**,**R$_v$3XH**,**R$_v$3XE**,**R$_v$3XI**

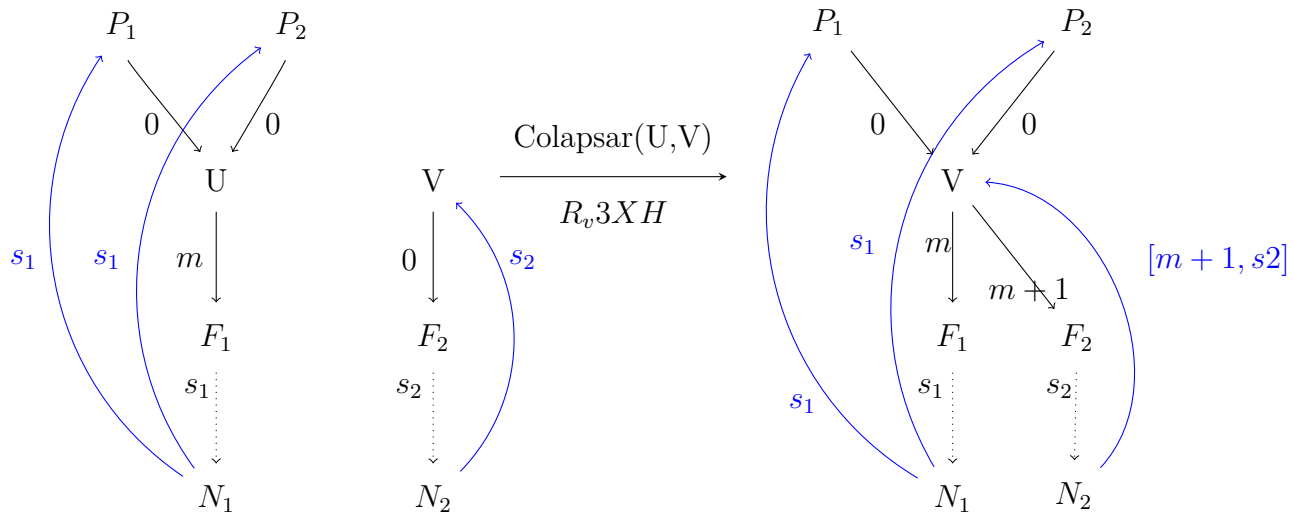


Figura 3.13: Aplicação da fórmula R$_v$3XH

3.5 Exemplo de Compressão Horizontal

Para facilitar a compreensão das regras apresentadas, seguiremos com um exemplo de execução do algoritmo de C.H em um grafo de dedução(retirado de (José Flávio-2019)). Queremos demonstrar a validade da fórmula: $((A_1 \supset A_2) \supset ((A_1 \supset (A_2 \supset A_3)) \supset ((A_2 \supset (A_3 \supset A_4)) \supset (A_1 \supset A_3))))$

O grafo que contempla a demonstração pode ser visto na figura 3.14. Os passos de descarte de hipóteses foram omitidos por conveniência e os vértices que serão alvo de junções estão coloridos com a cor verde. As hipóteses aparecem com retângulos, enquanto fórmulas que surgiram a partir de derivações aparecem no formato de elipse.

Primeiramente, seguindo o processo de baixo para cima e da esquerda para a direita, o primeiro passo seria colapsar os dois vértices correspondentes a A_2 marcados no grafo. Nos dois casos, os vértices vêm de $\supset E$ e não possuem arestas de ancestralidade ou colapsos anteriores. Assim, aplicaremos a regra **R0EE**, adicionando arestas de ancestralidade e labels. O grafo após o colapso pode ser encontrado na figura 3.15, onde o vértice colapsado se encontra na cor roxa.

Passando para o nível superior do grafo, nossos primeiros alvos serão os dois vértices A_1. Como o segundo já possui uma aresta de ancestralidade incidente, nos deparamos com o caso da regra **R$_v$2HH**, onde juntamos os vértices e rebaixamos a aresta de ancestralidade. Ao colapsarmos o vértice resultante com o outro vértice "A_1", temos uma regra da forma **R$_e$3HH**,

cujo procedimento será similar, mas precisamos adicionar o label λ na aresta comum. O resultado dessas duas operações pode ser visto na figura 3.16.

Por fim, a última etapa seria colapsar os dois vértices correspondentes a $A_1 \supset A_2$. Como já vimos, esse caso seria mais uma aplicação da regra $\mathbf{R}_e\mathbf{2HH}$ porém, nesse caso não podemos rebaixar as arestas, visto que já existem arestas de ancestralidade no vértice de destino e, portanto, precisamos excluir essas arestas. O grafo final pode ser visto na figura 3.17

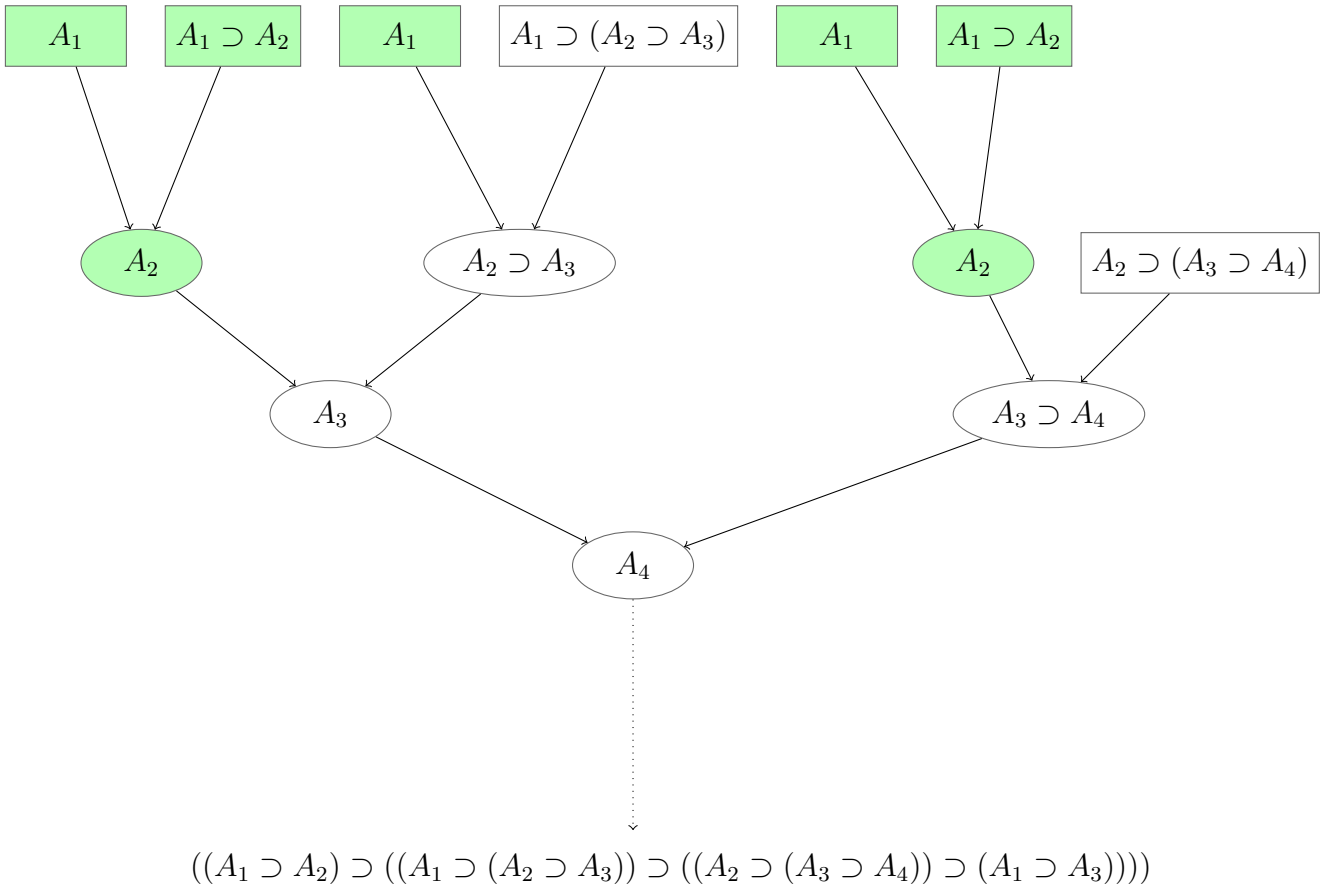


Figura 3.14: Exemplo de Grafo de prova

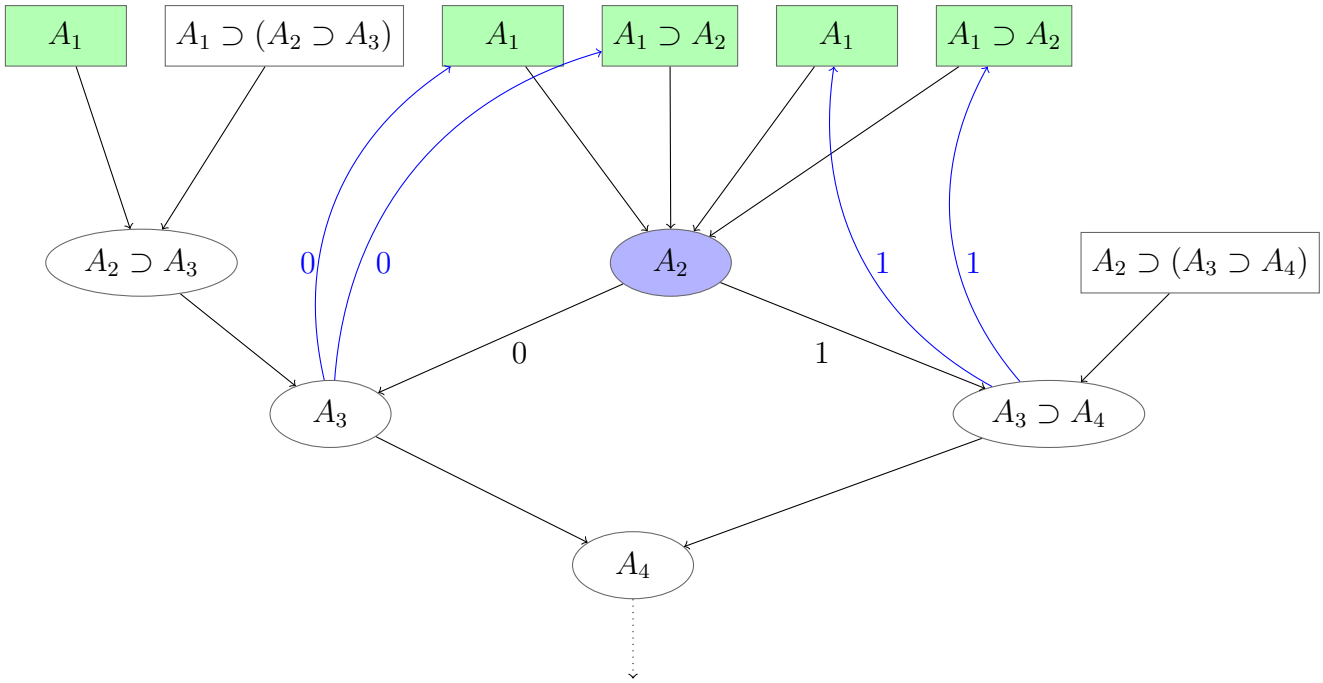


Figura 3.15: Grafo após colapsar vértices A_2

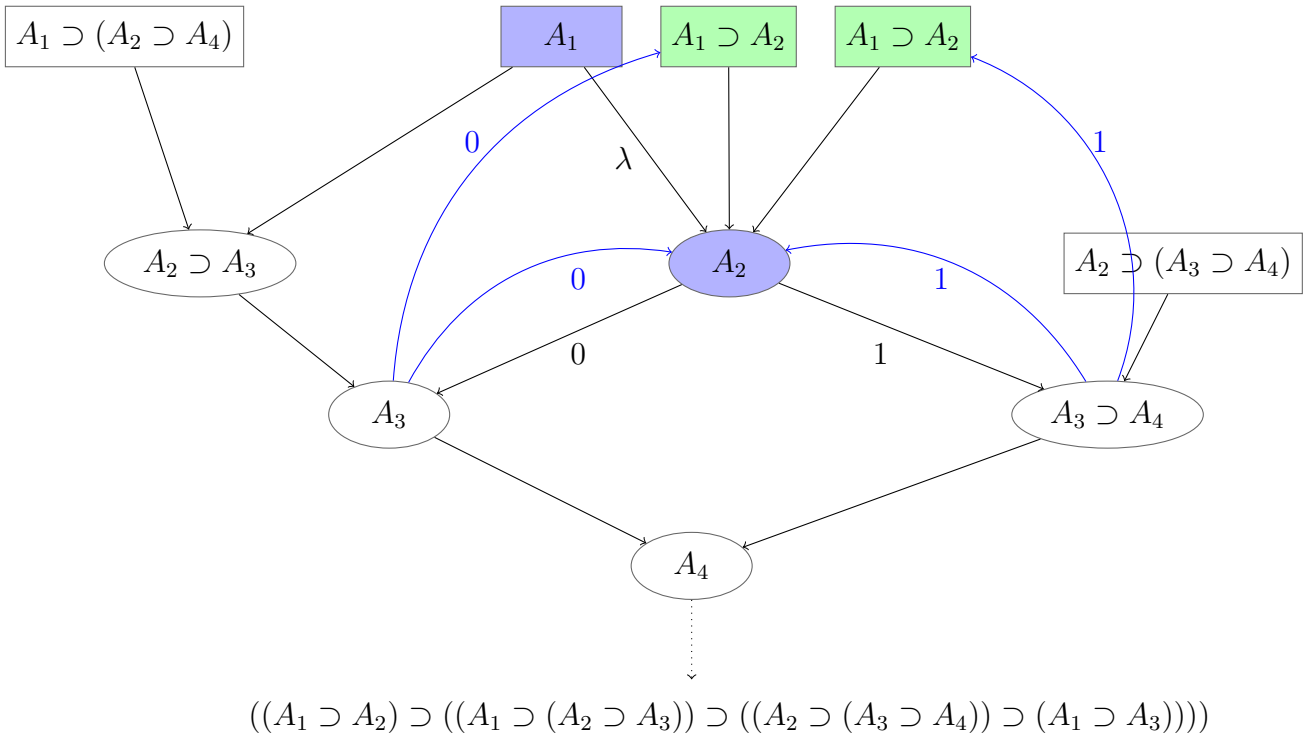


Figura 3.16: Grafo após colapso dos vértices A_1

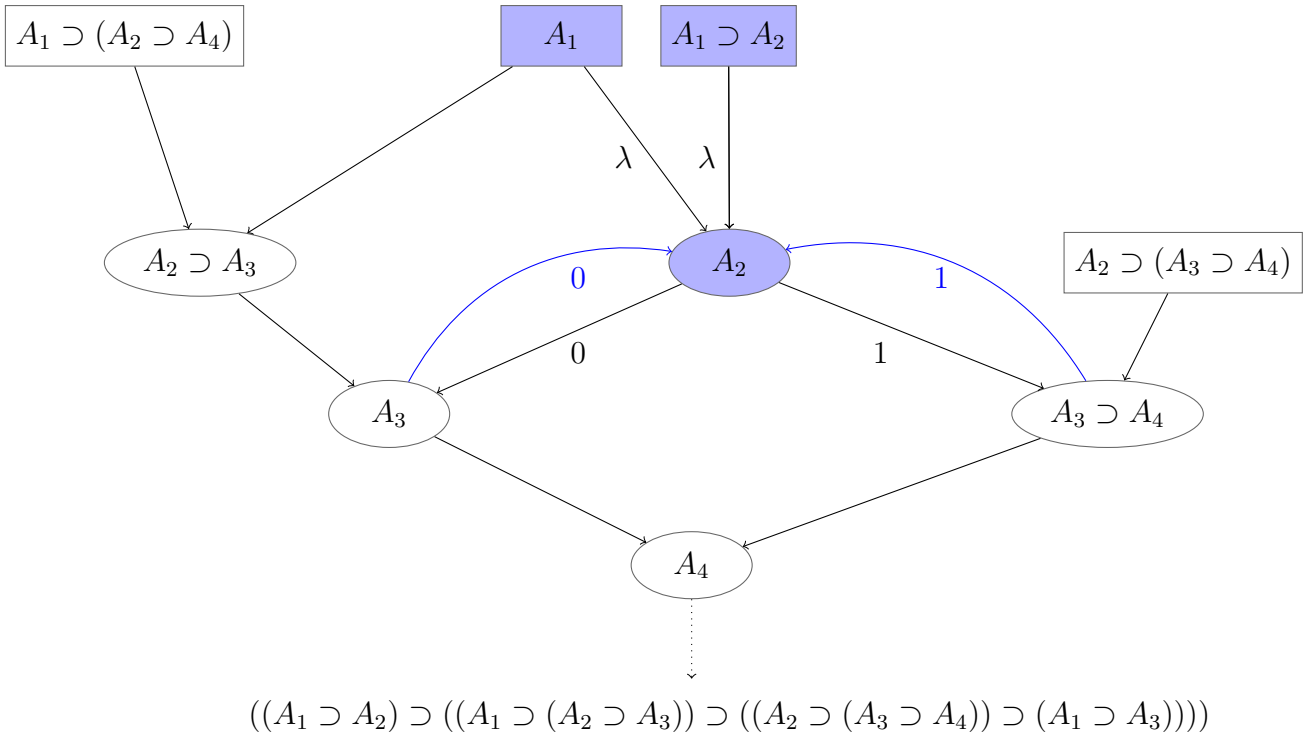


Figura 3.17: Grafo final após o processo de compressão

4
Implementação das regras

Nesta seção, abordaremos a parte prática do projeto, que consiste em implementar uma parte da Compressão Horizontal seguindo um conjunto específico de regras e estruturas de dados. Nosso objetivo é desenvolver um código capaz de interpretar essas regras e aplicá-las a um grafo de dedução.

Durante tentativas anteriores de implementação das regras em outros projetos, identificamos um problema recorrente de inflexibilidade. Para que nosso código seja robusto, ele não só deve abranger os 36 casos já considerados, mas também ser facilmente adaptável a possíveis mudanças e adições de regras no futuro, caso as condições do problema se alterem.

Para alcançar essa flexibilidade, adotaremos uma abordagem que reflita os diagramas que ilustram as regras em formato de entrada para o programa. Para facilitar a inserção e escrita dessas regras, optaremos pelo uso do formato de texto em JSON, que permitirá a definição de campos necessários para abranger todos os tipos de regras. Com essa estratégia, esperamos tornar o processo de descrição das regras mais intuitivo e dinâmico.

O esperado será entregar um módulo, que poderá ser incorporado no restante do algoritmo de compressão. Esse módulo receberá o grafo, os dois vértices a serem colapsados, a regra a ser aplicada e algumas estruturas de dados típicas. Ao final do programa, deverá retornar o grafo devidamente comprimido em relação aos dois vértices.

4.1
Decisões de projeto

Para seguir a compatibilidade com os projetos passados de implementação da C.H, seguimos com o mesmo modelo de representação que estava sendo usado anteriormente. A linguagem de programação escolhida foi o Python 3.0, principalmente pela variedade de bibliotecas relacionadas a grafos.

Entre as bibliotecas principais, podemos destacar a **graphviz** e a **pydotplus**. Através da graphviz, podemos representar facilmente grafos e visualizá-los em arquivos DOT, que é o formato mais popular para representação em grafos. A biblioteca pydotplus, é uma extensão capaz de manipular os grafos através de uma interface mais amigável.

A partir dessas ferramentas, é importante ressaltar as seguintes características de representação:

- Os vértices possuem um identificador único, chamado de name. Eles também possuem a propriedade label, que será destinada para escrever a fórmula relativa àquele vértice.
- As arestas de dedução terão terão como label um inteiro. Inicialmente, todas serão inicializadas com label 0, mas com o decorrer da compressão, esses valores irão mudar conforme descrito nas regras.
- As arestas de ancestralidade serão identificadas pela cor azul. O label, diferentemente das arestas de dedução, será uma lista de inteiros, relativos aos labels das arestas de dedução no fluxo de leitura da prova.
- A cor λ, que pode ser encontrada nas regras, será representada pela cor verde.
- O código que foi usado como base e que irá interagir com o código usa a estrutura e_in, que é um dicionário onde a chave é identificador do vértice e o segredo é uma lista de todas as arestas de dedução que incidem nesse vértice. Analogamente, usaremos também as estruturas e_out e_inA e_outA para arestas de saída, ancestralidade e entrada e ancestralidade e saída, respectivamente. O motivo é para manter a compatibilidade e para economizar na complexidade computacional das buscas.

4.2 Noções básicas sobre implementação

O código deste projeto receberá como entrada os dois vértices a serem colapsados, as estruturas e_in,e_out e_inA e_outA e o grafo de dedução. Como saída, deverá devolver o grafo após a aplicação da correta da regra.

Como já explicamos, o objetivo é descrever perfeitamente os diagramas que foram apresentados para ilustrar as regras. Entretanto, os identificadores usualmente usados nos diagramas, como p1,p2,u,v..., não correspondem aos mesmos identificadores das entradas dos vértices dos grafos.

Para iniciar o mapeamento, adotaremos a convenção de que o vértice "u"no diagrama corresponderá ao vértice da esquerda a ser colapsado, e analogamente, o vértice "v"no diagrama corresponderá ao outro vértice a ser colapsado. Utilizando esses dois vértices como referência, buscaremos as correspondências dos demais vértices através das arestas.

Por exemplo, suponha que no diagrama exista a aresta "p1-u". Com base na posse do vértice "u"no grafo, poderemos procurar nas arestas incidentes do

grafo o vértice que corresponde ao vértice "p1". A partir dessa informação, determinamos tanto o objeto que corresponde a "p1"quanto o objeto que corresponde à aresta "p1-u".

Esse processo de mapeamento nos permitirá identificar as relações entre os vértices do diagrama e do grafo, tornando possível o colapso de vértices e a representação simplificada da estrutura, conforme desejado. Colocaremos essa correspondência em um dicionário, cuja chave será a escrita no diagrama e o segredo será o objeto na implementação do grafo. Para ilustrar, segue um pseudocódigo:

Algorithm 2 Pseudocódigo aplicador de regra

$Data : G, Json, e\_in, e\_out, e\_inA, e\_outA$
$Result : G$

$jsonData \leftarrow ReadJson(Json)$
$DictV, DictE \leftarrow MapVertexEdge(G, JsonData, e\_in, e\_out, e\_inA, e\_outA)$
$G \leftarrow AddEdges(JsonData, DictEdges, G)$
$G \leftarrow DeleteEdges(JsonData, DictEdges, G)$
$G \leftarrow G.Remove(V)$

4.3 Representação em JSON

Como escolhemos JSON como forma de definir o diagrama de uma regra, precisamos escolher quais serão os campos necessários para fazer esse mapeamento. Temos a obrigatoriedade de descrever os vértices u e v a serem colapsados exatamente com esses labels, mas temos liberdade para nomear os outros como quisermos. Os vértices serão representados por strings e as arestas direcionais no formato "A-B", que quer dizer a existência de uma aresta de A para B. Iremos construir e incrementar esses campos aos poucos, passando por todos os 4 tipos de regras diferentes para avaliar as necessidades.

4.3.1 Regras do tipo 0

Nesse conjunto de regras, na parte da esquerda do diagrama precisamos apenas definir o conjunto de arestas. Na parte da direita, precisamos dizer quais são as novas arestas de dedução, quais as arestas de dedução devem ser retiradas e quais arestas de ancestralidade devem surgir. Dessa forma, podemos propor o seguinte JSON para a regra:

```
1 {
2   "rule_name": "name",
```

```
3   "edges": ["A-B"],
4   "new_edges":[{"edge_name": "A-B","edge_label": 0}],
5   "new_ancestor_edges":[{"edge_name": "A-B","edge_label":
        ["0"]}],
6   "delete_edges":["A-B"]
7 }
```

4.3.2 Regras do tipo 1

O método anterior não funciona para as regras do tipo 1. Ao colapsarmos dois vértices, precisamos atribuir um novo label para as arestas do vértice a direita. No caso das regras do tipo 0, como todos as arestas são inicializadas com label 0 e ainda não houve colapso, basta definir o valor 1 para as novas arestas. Por outro lado, no caso do vértice u já ter sido colapsado anteriormente, precisamos descobrir o maior label para continuarmos a sequência.

Assim, usando a estrutura e_out, podemos procurar o maior label de arestas de saída de u, que chamaremos de m. Para buscar uma representação no Json, aproveitaremos o fato de que até então só estávamos usando inteiros positivos. Sendo o label novo negativo, podemos associar da forma $label\_novo = -1 * (label + 1) + m$. Dessa forma, -1 corresponderia ao próprio m, -2 a m+1 e assim sucessivamente.

4.3.3 Regras dos tipos 2 e 3

A dificuldade em representar as regras dos tipos 2 e 3 está relacionada com as arestas de ancestralidade. Nessas situações, precisamos combinar os labels das arestas antigas para construir as novas, mas sem uma regra tão específica quanto nas regras do tipo 1, que precisávamos apenas do label de valor mais alto.

A solução foi criar mais um campo no JSON. Precisamos saber quais são os labels específicos que serão guardados para agregar na nova aresta de ancestralidade. Para isso, o campo SpecialLabel será adicionado. Nele, teremos a aresta que é de interesse o label e uma chave para identificá-lo.

Assim, passaremos a sequência correta no campo de criação da aresta de ancestralidade. no exemplo da regra R_v2EH, partimos de s1 e fomos para [0,1,s1]. Entregaríamos no campo a sequência ["0","1","s1"]. No campo de special label, passaríamos a aresta e o identificador "s1"correspondente.

Outra convenção que tivemos que adotar é no caso do label λ. Nesse caso, assim como fizemos no caso do maior label, usaremos -100 para representar.

Nesse caso, especialmente para as regras do tipo 3, ainda vale a convenção que foi adotada para valores negativos, em que -1 corresponderia ao maior label das arestas de saída de u.

4.4 Testes

Os testes foram feitos inserindo grafos com as configurações das regras e avaliando se os grafos de saída eram compatíveis com o esperado pela teoria.

Importante ressaltar que como entrada do programa, embora não acarrete em erros, não é necessário incluir arestas e vértices que não mudam no processo de junção.

Como exemplo de regra a ser testada, podemos apresentar um JSON que é correspondente à regra R0IE

```
{
  "name": "R0IE",
  "edges": ["p1-u","p2-v","p3-v","u-f1","v-f2"],
  "new_edges":[
        {edge_name": "p2-u", "edge_label": 0},
        {"edge_name": "u-f2","edge_label": 1},
        {edge_name": "p3-u","edge_label": 0}],
  "new_ancestor_edges":[
        {edge_name": "f1-p1","edge_label": [0]},
        {"edge_name": "f2-p2","edge_label": [1]},
        {"edge_name": "f2-p3","edge_label": [1]}],
  "delete_edges":["p2-v","p3-v","v-f2"]
}
```

```
{
  "name": "R1XI",
  "edges": ["p2-v","v-f2"],
  "new_edges":[
        {edge_name": "u-f2", "edge_label": -2},
        {"edge_name": "p2-u","edge_label": 0}],
  "new_ancestor_edges":[
        {edge_name": "f2-p2","edge_label": ["-2"]}]
  "delete_edges":["p2-v","v-f2"]
}
```

```
{
  "name": "Re2EH",
  "edges": ["p1-u","p2-u","p3-v","u-f1","v-f1"],
```

```
4   "new_edges":[{edge_name": "u-f1", "edge_label": -100}],
5   "ancestor_edges":[
6         {edge_name": "n1-u", "edge_label": ["s1"]},
7         {edge_name": "n2-v", "edge_label": ["s2"]}]
8   "new_ancestor_edges":[
9         {edge_name": "n1-p1","edge_label": ["0","s1"]}
10        {edge_name": "n1-p2","edge_label": ["0","s1"]}
11        {edge_name": "n2-u","edge_label": ["s2"]}]
12  "delete_edges":["v-f1","u-f1"],
13  "delete_ancestor_edges":["n2-v","n1-u"]
14 }
```

4.5 Conclusão e Próximos passos

Nesse trabalho, além da descrição teórica sobre o tema e sobre trabalhos anteriores, foi deixado como contribuição um código que pode ser aplicado em um algoritmo de Compressão Horizontal. Nele, é feita a implementação das regras de manipulação dos grafos, onde é retornado o grafo com os dois vértices devidamente colapsados.

Essa implementação foi testada de forma satisfatória, atingindo o seu objetivo com relação à implementação. Além disso, o código é compatível com as implementações de C.H que tem sido elaboradas em python3 com a biblioteca graphviz, podendo ser importado sem grandes atualizações. Por fim, também foram atualizadas as estruturas de dados que estavam sendo usadas no código-base, para não prejudicar o resto do algoritmo de C.H.

Um próximo passo para seguir trabalhando na linha desse projeto, seria a integração completa da implementação descrita com o esboço de Compressão Horizontal que serviu de código-base para o projeto. Embora o algoritmo seja capaz de aplicar perfeitamente as regras de colisão, ele não faz a distinção de quando é necessário aplicar cada tipo de regra, que poderia ser um novo projeto a ser implementado.Também é possível trabalhar no próprio código-base em diversos pontos possíveis de melhora.

Alternativamente, um universo de possibilidades dentro da teoria da prova se abre a partir da conclusão que $NP = PSPACE$ poderiam ser explorados, tanto em projetos práticos quanto teóricos.

Referências bibliográficas

[Gordeev-Hermann-2019] HAEUSLERS, L. G. E. H.. **Proof compression and np versus pspace**. Springer Science+Business Media B.V, 2019.

[Gordeev-Hermann-II-2019] HAEUSLERS, L. G. E. H.. **Proof compression and np versus pspace ii**. Uniwersytet L´odzki, 2019.

[Hermann-JoseFlavio-Robinson-2023] EDWARD HERMANN, J. F. B. J. E. R. F.. **On the horizontal compression of dag-derivations in minimal purely implicational logic**. 2023.

[José Flávio-2019] JR, J. F. B.. **Uma abordagem experimental sobre a compressão de provas em dedução natural minimal implicacional.** Dissertação de Mestrado, Puc-Rio, Abril 2019.

[Paulo-Hermann 2006] E EDWARD HERMANN JAEIÇER, P. B. M.. **Teoria das Categorias para Ciência da Computação**. Editora Sagra Luzzato, RS, Brasil, 2006.

[Pawitz 1965] PRAWITZ, D.. **Natural Deduction**. Dover Publications, USA, 1965.