Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Prova Automática de Teoremas para Lógicas Confluentes

Boris Marinho Ramos Silva Araujo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.a Dr.a Cláudia Nalon

Brasília
2021

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Prova Automática de Teoremas para Lógicas Confluentes

Boris Marinho Ramos Silva Araujo

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.a Dr.a Cláudia Nalon (Orientadora)
CIC/UnB

Prof. Dr. Thiago de Paulo Faleiros
CIC/UnB

Dr. Bruno César Ribas
FGA/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 4 de novembro de 2021

Dedicatória

Dedico este trabalho aos meus pais Guilherme e Joice e à memória de meu melhor amigo Thales.

Agradecimentos

Agradeço à minha orientadora, Cláudia Nalon. Pela empatia e paciência nos momentos difíceis; pela boa vontade, por acreditar neste trabalho, pela dedicação e por ser alguém com quem contar em qualquer desafio que aparecesse.

Agradeço a meus pais, Guilherme e Joice por estarem sempre presentes e pelo apoio incondicional durante toda a minha formação.

Agradeço aos meus amigos: Daniel, Felipe, Gabriel, Heloísa, Iago, Lúcio, Natasha e Thales que perto ou distante sempre me acompanharam.

Resumo

Este trabalho é um projeto sobre lógicas multimodais. É apresentado o arcabouço teórico básico da sintaxe e semântica da lógica multimodal K_n e também o cálculo de resolução para K_n e suas extensões pelos axiomas confluentes: T_a, Ban_a, B_a, D_a, 4_a, $G_a^{0,1,1,1}$, F_a, 5_a e $G1_a$, onde a refere-se ao índice do agente. A parte de experimentação do trabalho consistiu de dois experimentos com o provador KSP. O primeiro experimento consistiu na verificação da correção da implementação das regras de inferência de confluência, onde foram feitos testes de coerência interna nas regras oriundas de um mesmo axioma. Foi identificada falha para a regra de inferência relacionada a um axioma. O segundo experimento consistiu na refatoração do código tendo como objetivo a melhoria de desempenho por meio da mudança de estruturas de dados na parte do pré-processamento. De maneira geral o novo desempenho do programa é pior, porque soluciona menos fórmulas que a implementação antiga. Porém o resultado foi bastante heterogêneo. Apesar da implementação antiga solucionar mais fórmulas, os tempos de resolução foram muito parecidos. Seis regras de inferência tiveram tempo melhor na nova implementação e sete tiveram um tempo pior.

Palavras-chave: lógica modal, confluência, lógica multimodal, KSP

Abstract

This work is a project on multimodal logics. The basic theory about multimodal logic K_n, its syntax and semantics are presented, as well as the resolution calculus for K_n together with the confluence axioms: T_a, Ban_a, B_a, D_a, 4_a, $G_a^{0,1,1,1}$, F_a, 5_a, and $G1_a$, where a is the agent index. The experimental part of this work consisted of two experiments with the KSP prover. The first experiment consisted of verifying the correctness of the implementation of the confluence inference rules; for this purpose, inference rules of the same axiom were tested against each other. It was identified that the inference rule for one axiom had a flaw. The second experiment consisted of refactoring the code aiming for improving performance by changing data structures in the pre-processing part. The results were mixed, some rules gained performance, others lost. There are some special cases that need further investigation.

Keywords: modal logic, confluence, multimodal logic, KSP

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Lógica modal é uma extensão da lógica proposicional, com o acréscimo de dois operadores $\Box$, o operador de necessidade, e $\Diamond$, o operador de possibilidade. A sentença $\Box p \Rightarrow \Diamond q$ se lê "se p é necessário então q é possível". Nessa lógica, que chamamos de lógica K, é possível representar problemas onde uma mesma proposição assume valores diferentes dependendo do contexto. Relações entre diferentes contextos, chamadas de relações de acessibilidade, são também utilizadas para a determinação de proposições necessárias ou possíveis.

Uma generalização da lógica K é a lógica K_n, que chamamos de lógica multimodal. Ela é bastante semelhante à lógica modal K, mas permite especificar proposições do ponto de vista de diferentes agentes, denotados por $a \in A_n = \{1, \ldots, n\}$. Estes agentes são representados como índices nos operadores modais: $\boxed{a}, \Diamond\!\!\!\!\!\!\boxed{a}$.

A base do cálculo utilizado neste trabalho é proposto em [1] e utiliza o método de resolução. Posteriormente o cálculo foi estendido para abrigar lógicas multimodais confluentes [2]. Estas lógicas, denotadas por $\mathsf{K}_n^{p,q,r,s}$, são o objeto deste trabalho. Lógicas confluentes são lógicas multimodais com o acréscimo de axiomas $G_a^{p,q,r,s}$, onde $p, q, r, s \in \mathbb{N}$. Cada axioma $G_a^{p,q,r,s}$ é da forma $\Diamond\!\!\!\!\!\!\boxed{a}^p \boxed{a}^q \varphi \Rightarrow \boxed{a}^r \Diamond\!\!\!\!\!\!\boxed{a}^s \varphi$ tal que $a \in A_n$, $\boxed{a}^0 \varphi = \Diamond\!\!\!\!\!\!\boxed{a}^0 \varphi = \varphi$, $\boxed{a}^n \varphi = \boxed{a}^{n-1} \boxed{a} \varphi$ e $\Diamond\!\!\!\!\!\!\boxed{a}^n \varphi = \Diamond\!\!\!\!\!\!\boxed{a}^{n-1} \Diamond\!\!\!\!\!\!\boxed{a} \varphi$ e φ é uma fórmula bem-formada.

Além dos próprios problemas de lógica, a lógica modal também é utilizada para especificação de *hardware* [3], problemas de concorrência [4], inteligência artificial [5].

Existem diferentes tipos de sistemas dedutivos para resolver problemas de lógicas modais: cálculo axiomático [6], *tableau* [6], dedução natural [7]. Cada cálculo tem sua particularidade, por exemplo, o conjunto de premissas de sistemas axiomáticos é o conjunto de todas as tautologias. Mesmo que não seja necessário gerar todas as tautologias para solucionar um problema, é necessário selecionar as tautologias mais adequadas. Para o caso da dedução natural ocorre algo semelhante e muitas vezes é necessário fazer suposições a respeito da fórmula para chegar a uma solução. Este tipo de heurística por

vezes é de difícil implementação. Então é necessário escolher um cálculo adequado para implementar um provador automático de teoremas. O cálculo implementado no KSP é o cálculo de resolução, porque é de mais simples implementação. Este cálculo é explicado mais adiante no Capítulo 3.

O provador utilizado na experimentação deste trabalho é o provador automático de teoremas KSP que tem sua implementação descrita em [8]. As etapas experimentais foram feitas no módulo que concerne às regras de resolução de lógicas confluentes. A primeira etapa foi a verificação da correção das regras de inferência confluentes $T_a, Ban_a, B_a, D_a, 4_a, G_a^{0,1,1,1}, F_a, 5_a, G1_a$ que correspondem aos axiomas de confluência em que estamos interessados e são descritas na Tabela 3.3, a fim de verificar se regras de inferência de um mesmo axioma geram resultados iguais. A segunda etapa foi a refatoração do código a fim de utilizar estruturas de dados de tempo assintótico menores na manipulação de cláusulas na fase do pré-processamento. O pré-processamento de cláusulas aplica as regras de inferência de confluência. Durante essa fase de pré-processamento as cláusulas eram armazenadas na forma de listas ligadas. Como o comportamento a ser simulado pelo provador requer a representação de conjuntos é necessário evitar repetições. Para checar duplicatas em uma lista ligada é necessário percorrê-la completamente. Então a ideia do Experimento 2 foi substituir as listas ligadas por tabelas de dispersão, onde se pode conferir duplicatas em tempo constante.

A primeira etapa experimental foi capaz de identificar resultados conflitantes nas regras de inferência do axioma Ban_a. Como esta regra de inferência que apresentou erro não possui prova de completude e a implementação aparenta estar correta, não se viu motivo para continuar a investigação. A segunda etapa teve ganhos de desempenho em algumas regras e perdas em outras, mas a implementação refatorada foi capaz de solucionar menos fórmulas que a implementação anterior.

Este trabalho é estruturado da seguinte forma: as definições formais referentes à linguagem e ao cálculo são apresentadas nos Capítulos 2 e 3, respectivamente; o provador KSP é brevemente descrito no Capítulo 4; no Capítulo 5, são detalhados os experimentos e resultados; por fim, no Capítulo 6, apresentamos conclusões e direções para trabalhos futuros.

Capítulo 2

Fundamentos Teóricos

A lógica proposicional modal, utilizada neste trabalho, é uma extensão da lógica proposicional. É natural que revisemos a lógica proposicional e a partir dela construamos a lógica modal, já que muitos conceitos são parecidos, senão idênticos.

A construção da sintaxe da lógica proposicional pressupõe um conjunto enumerável de símbolos proposicionais $P = \{p, q, r, s, \ldots\}$, operadores proposicionais $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$. Sua semântica é determinada por uma função π que atribua valor de verdade aos símbolos proposicionais, ou seja, com a seguinte assinatura: $\pi : P \rightarrow \{V, F\}$.

A partir disso é possível construir fórmulas proposicionais complexas e, a partir das propriedades dos operadores, atribuir valor de verdade para a fórmula completa.

A lógica modal possui dois operadores unários a mais que a lógica proposicional, sendo eles $\Box$ (necessariamente) e $\Diamond$ (possivelmente). A função π como definida para fórmulas proposicionais não é mais capaz de valorar fórmulas com operadores modais, porém o mecanismo sintático é bastante semelhante, então o exploraremos primeiro.

2.1 Sintaxe

A primeira coisa a se determinar no estudo da sintaxe é o conceito de fórmula bem-formada. Uma fórmula precisa obedecer a algumas regras sintáticas para que possa ser corretamente interpretada. Senão, é impossível interpretar o que determinada sequência de símbolos significa. Por exemplo: a fórmula $p \vee q\wedge$ é uma fórmula sintaticamente correta? E a fórmula $\varphi \wedge \psi$? Para responder isso é necessário o conceito de fórmula bem formada.

Definição 1 Fórmula bem formada

Seja $P = \{p, q, r, s, \ldots\}$ o conjunto de símbolos proposicionais; **true** e **false** constantes; e $A_n = \{1, \ldots, n\}, n \in \mathbb{N}$ o conjunto finito dos agentes modais. O conjunto

das fórmulas bem formadas, WFF, é definido indutivamente:

- $\mathbf{true}, \mathbf{false} \in \mathsf{WFF}$;
- se $p \in P$ então $p \in \mathsf{WFF}$;
- se $\varphi \in \mathsf{WFF}$ então $\neg\varphi, \langle a \rangle \varphi, [a]\varphi \in \mathsf{WFF}$, onde $a \in A_n$;
- se φ e $\psi \in \mathsf{WFF}$ então $(\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \Rightarrow \psi), (\varphi \Leftrightarrow \psi) \in \mathsf{WFF}$.

Quando o conjunto de agentes é vazio, esta definição corresponde à da lógica proposicional. Quando o conjunto de agentes é unitário, escrevemos $\Box$ e $\Diamond$ ao invés de $[1]$ e $\langle 1 \rangle$.

Definição 2 Literais, literais modais e cláusulas proposicionais

- Um literal é uma variável proposicional $p \in P$ ou sua negação $\neg p$, denota-se um literal como l.
- Um literal modal é da forma $[a]l$ ou $\langle a \rangle l$, onde l é um literal e $a \in A_n$.
- Uma cláusula proposicional é uma disjunção finita de literais $\bigvee_{i=1}^{n} l_i$.

Agora que sabemos o conceito de fórmula bem formada, sabemos que a fórmula dita acima $p \vee q\wedge$ não é bem formada porque não pode ser construída a partir da Definição 1 e, portanto, está sintaticamente incorreta e é impossível atribuir significado a ela. Para preservar a simplicidade e manter a clareza, parêntesis serão omitidos quando for conveniente seguindo a ordem de precedência definida a seguir.

Precedência	**Operador**
1.	$\neg, \langle a \rangle, [a]$
2.	$\wedge$
3.	$\vee$
4.	$\Rightarrow$
5.	$\Leftrightarrow$

Tabela 2.1: Ordem de precedência dos operadores

Exemplo 1 Ordem de precedência dos operadores

A fórmula $\neg p \vee q \wedge r \Rightarrow s \Leftrightarrow t$ é interpretada corretamente como $(((\neg p \vee (q \wedge r)) \Rightarrow s) \Leftrightarrow t)$.

Uma estrutura fundamental para o estudo da sintaxe multimodal é a árvore sintática das fórmulas, porque a partir delas formamos o conceito de nível modal, que é primordial para o cálculo apresentado no Capítulo 3. A definição a seguir foi retirada de [9].

Definição 3 Árvore sintática

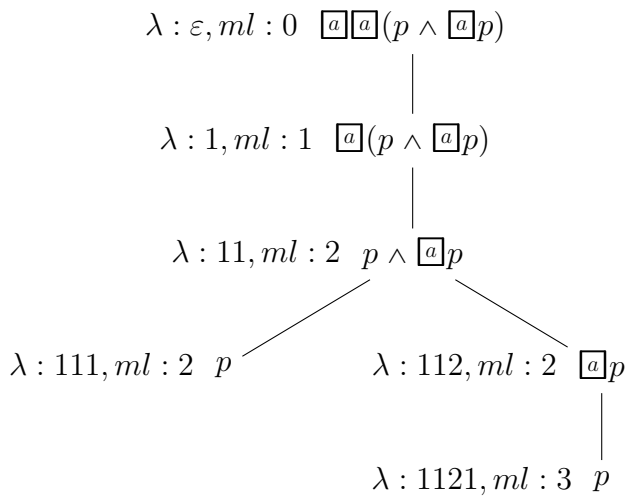
Sejam φ e φ' fórmulas bem-formadas. Seja Σ o alfabeto $\{1, 2\}$ e Σ^* o conjunto de todas as sequências finitas sobre Σ. Denotamos por ε a palavra vazia em Σ^*. Seja $ml \in \mathbb{N}$ o nível modal do nó e $\tau : \mathsf{WFF} \times \Sigma^* \times \mathbb{N} \to \mathcal{P}(\mathsf{WFF} \times \Sigma^* \times \mathbb{N})$ a função que constrói a árvore sintática anotada da fórmula, definida indutivamente a seguir.

- $\tau(\mathbf{true}, \lambda, ml) = \{(\mathbf{true}, \lambda, ml)\}$
- $\tau(\mathbf{false}, \lambda, ml) = \{(\mathbf{false}, \lambda, ml)\}$
- $\tau(p, \lambda, ml) = \{(p, \lambda, ml)\}$, para $p \in P$
- $\tau(\neg\varphi, \lambda, ml) = \{(\neg\varphi, \lambda, ml)\} \cup (\varphi, \lambda.1, ml)$
- $\tau(\varphi \wedge \varphi', \lambda, ml) = \{(\varphi \wedge \varphi', \lambda, ml)\} \cup \tau(\varphi, \lambda.1, ml) \cup \tau(\varphi', \lambda.2, ml)$
- $\tau(\varphi \vee \varphi', \lambda, ml) = \{(\varphi \vee \varphi', \lambda, ml)\} \cup \tau(\varphi, \lambda.1, ml) \cup \tau(\varphi', \lambda.2, ml)$
- $\tau(\varphi \Rightarrow \varphi', \lambda, ml) = \{(\varphi \Rightarrow \varphi', \lambda, ml)\} \cup \tau(\varphi, \lambda.1, ml) \cup \tau(\varphi', \lambda.2, ml)$
- $\tau(\boxed{a}\varphi, \lambda, ml) = \{(\boxed{a}\varphi, \lambda, ml)\} \cup \tau(\varphi, \lambda.1, ml + 1)$
- $\tau(\langle\!\langle a \rangle\!\rangle\varphi, \lambda, ml) = \{(\langle\!\langle a \rangle\!\rangle\varphi, \lambda, ml)\} \cup \tau(\varphi, \lambda.1, ml + 1)$

Dada uma fórmula φ, sua árvore sintática anotada é obtida por $\tau(\varphi, \varepsilon, 0)$.

Exemplo 2 Árvore sintática anotada

A fórmula $\boxed{a}\boxed{a}(p \wedge \boxed{a}p)$ possui a seguinte árvore sintática anotada, representada graficamente. Cada nó é representado por uma (sub)fórmula; posições e níveis modais são anotados à esquerda de cada nó.



No Exemplo 2, p ocorre nos níveis modais 2 e 3, nas posições 111 e 1121, respectivamente. Intuitivamente, o nível modal de uma variável proposicional é equivalente à quantidade de operadores modais no escopo dos quais ocorre.

O conceito de profundidade modal é semelhante ao de nível modal. Porém, enquanto nível modal descreve a quantidade de operadores modais englobando uma subfórmula, o conceito de profundidade modal descreve uma fórmula inteira. Profundidade modal descreve a maior quantidade de operadores modais aninhados em uma fórmula conforme a definição a seguir [9].

Definição 4 Profundidade modal

Sejam $\varphi, \psi \in \mathsf{WFF}$. Define-se a função $\mathsf{md} : \mathsf{WFF} \to \mathbb{N}$ da seguinte forma:

- $\mathsf{md}(p) = \mathsf{md}(\mathbf{true}) = \mathsf{md}(\mathbf{false}) = 0, p \in P$
- $\mathsf{md}(\neg\varphi) = \mathsf{md}(\varphi)$
- $\mathsf{md}(\varphi \wedge \psi) = \mathsf{md}(\varphi \vee \psi) = \mathsf{md}(\varphi \Rightarrow \psi) = \mathsf{max}(\mathsf{md}(\varphi), \mathsf{md}(\psi))$
- $\mathsf{md}(\Box\varphi) = \mathsf{md}(\Diamond\varphi) = 1 + \mathsf{md}(\varphi)$

onde $\mathsf{max} : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ é a função que determina o máximo entre dois números naturais.

Exemplo 3 Profundidade modal

$\mathsf{md}(\Box(p \wedge \Box p)) = 1 + \mathsf{max}(\mathsf{md}(p), \mathsf{md}(\Box p)) = 1 + \mathsf{max}(0, 1 + \mathsf{md}(p)) =$
$1 + \mathsf{max}(0, 1) = 1 + 1 = 2$

2.2 Semântica

Ao investigar a semântica da lógica modal que se descobre sua verdadeira natureza. Construindo uma ideia intuitiva da base por trás da lógica modal, nos concentramos na ideia de mundos. Mundos representam contextos. Enquanto na lógica proposicional existia apenas um contexto para a avaliação de uma fórmula φ, na lógica modal podemos dizer que φ pode assumir um valor de verdade no mundo a e outro no mundo b.

Então, o primeiro objeto matemático necessário para construir a semântica da lógica modal é um conjunto de mundos W, onde fórmulas φ podem possuir valores de verdade distintos em diferentes mundos.

O segundo objeto matemático necessário para construir a semântica da lógica modal é a relação de acessibilidade R, que é uma relação binária que indica a relação entre mundos. Para entender melhor esta relação, é útil finalmente introduzir a semântica dos operadores. Tome como exemplo a Figura 2.1, onde círculos representam mundos, a relação de acessibilidade R é representada pelas arestas e os rótulos de cada mundo apresenta os símbolos proposicionais que são ali verdadeiros. Símbolos que não ocorram como rótulos em mundos são considerados falsos.

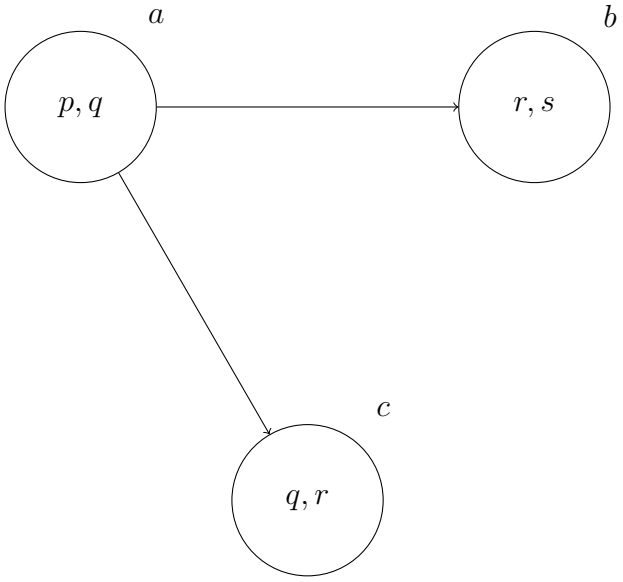


Figura 2.1: Relação de acessibilidade

O mundo a acessa b e c: aRb e aRc. A semântica do operador $\Box$ (necessariamente) é definido a partir dessa relação de acessibilidade. Se em todo mundo acessível a partir de a, φ é verdadeiro, então $\Box\varphi$ é verdadeiro em a. Ou seja, de acordo com a Figura 2.1, r é verdadeiro em b e em c. Como ambos os mundos são acessíveis a partir de a, temos que $\Box r$ é verdadeiro em a.

Já a semântica do operador $\Diamond$ funciona de forma ligeiramente diferente. Se em algum mundo acessível a partir de a φ é verdadeiro, então $\Diamond\varphi$ é verdadeiro em a. Então, na Figura 2.1, a acessa b e c: q, r e s ocorrem em pelo menos um dos mundos acessados por a, então $\Diamond q, \Diamond r, \Diamond s$ são verdadeiros em a. Entretanto, nenhuma dessas fórmulas é verdadeira em b e c, já que estes mundos não conseguem acessar mundos em que as proposições representadas por q, r e s sejam verdadeiras.

A noção intuitiva dada até agora ilustra uma lógica K, mas a noção formal abarca vários agentes. Em uma lógica K_n, com vários agentes, as arestas seriam rotuladas por cada um dos $a \in A_n = \{1, \ldots, n\}, n \in \mathbb{N}$.

Definição 5 *Frame*

Um *Frame* é uma tupla $\langle W, w_0, R_1, R_2, \ldots, R_n \rangle$, onde

- $W \neq \varnothing$ é o conjunto de mundos, com $w_0 \in W$; e
- $R_1, R_2, \ldots, R_n$ são as relações de acessibilidade, tal que $R_a \subseteq W \times W$ para todo $a \in A_n$.

Definição 6 Função de valoração

Seja $\pi : P \times W \longrightarrow \{V, F\}$ a função de valoração. Esta função associa qualquer variável proposicional $p \in P$ e um mundo $w \in W$ a um valor de verdade V ou F.

Exemplo 4 Função de valoração

De acordo com a Figura 2.1, $\pi(p, a) = V$.

Definição 7 Modelo

Seja $F = \langle W, w_0, R_1, R_2, \ldots, R_n \rangle$ um *frame* e $\pi : P \times W \longrightarrow \{V, F\}$ uma função de valoração. Um modelo (ou estrutura) de Kripke é uma tupla $M = \langle W, w_0, R_1, R_2, \ldots, R_n, \pi \rangle$, ou simplesmente $M = \langle F, \pi \rangle$.

Exemplo 5

Utilizando o exemplo da Figura 2.1 temos que:

- $W = \{a, b, c\}$

- $R = \{(a, b), (a, c)\}$
  A função π é definida como:
- $\pi(p, a) = \pi(q, a) = \pi(q, c) = \pi(r, b) = \pi(r, c) = \pi(s, c) = V$
- $\pi(r, a) = \pi(s, a) = \pi(p, c) = \pi(p, b) = \pi(q, b) = \pi(s, b) = F$

Como a função de valoração só associa valores de verdade para variáveis proposicionais, é necessário um mecanismo para atribuir significado semântico a fórmulas complexas. Daí, então, o conceito de satisfatibilidade.

Definição 8 Satisfatibilidade de uma fórmula em um mundo w de um modelo M

Seja $M = \langle W, w_0, R_1, R_2, \ldots, R_n, \pi \rangle$ um modelo. A satisfatibilidade de uma fórmula em um mundo w de um modelo M é definida da seguinte forma:

- $\langle M, w \rangle \models p$ sse $\pi(p, w) = V$
- $\langle M, w \rangle \models \neg\varphi$ sse $\langle M, w \rangle \not\models \varphi$
- $\langle M, w \rangle \models \varphi \vee \psi$ sse $\langle M, w \rangle \models \varphi$ ou $\langle M, w \rangle \models \psi$
- $\langle M, w \rangle \models \varphi \wedge \psi$ sse $\langle M, w \rangle \models \varphi$ e $\langle M, w \rangle \models \psi$
- $\langle M, w \rangle \models \varphi \Rightarrow \psi$ sse $\langle M, w \rangle \not\models \varphi$ ou $\langle M, w \rangle \models \psi$
- $\langle M, w \rangle \models \varphi \Leftrightarrow \psi$ sse $\langle M, w \rangle \models \varphi \Rightarrow \psi$ e $\langle M, w \rangle \models \psi \Rightarrow \varphi$
- $\langle M, w \rangle \models \boxed{a}\varphi$ sse para todo $w' \in W$, se wR_aw' então $\langle M, w' \rangle \models \varphi$
- $\langle M, w \rangle \models \diamondsuit_a\varphi$ sse existe $w' \in W$, tal que wR_aw' e $\langle M, w' \rangle \models \varphi$

Definição 9 Satisfatibilidade local

Seja $M = \langle W, w_0, R_1, R_2, \ldots, R_n, \pi \rangle$ um modelo. Dizemos que uma fórmula é localmente satisfeita se, e somente se, existe um modelo M tal que $w_0 \in W$ satisfaça φ, isto é, $\langle M, w_0 \rangle \models \varphi$. Neste caso, M é um modelo para φ e nós denotamos isso por $M \models_{\mathcal{L}} \varphi$.

Definição 10 Satisfatibilidade global

Seja $M = \langle W, w_0, R_1, R_2, \ldots, R_n, \pi \rangle$ um modelo. Dizemos que uma fórmula é globalmente satisfeita se, e somente se, existe um modelo M em que todos os mundos

$w \in W$ satisfaçam φ neste modelo. Sendo assim, M é um modelo para φ e isso é denotado por $M \models_{\mathcal{G}} \varphi$.

2.3 Extensões

Chamemos a lógica descrita até agora de K_n. Note, da definição de modelo, que não há qualquer restrição sobre as relações de acessibilidade R_a. O acréscimo de axiomas à lógica K_n corresponde à restrição da classe de modelos em que as fórmulas são válidas [2], isto é, são adicionadas restrições às relações de acessibilidade. Por exemplo, a determinação da validade de uma fórmula na lógica K_n com acréscimo do axioma B_a não precisa considerar todos os possíveis modelos, mas tão somente aqueles em que a relação de acessibilidade R_a para o agente $a, a \in A_n$ for simétrica [10]. A Tabela 2.2 lista os axiomas relacionados às regras de inferência implementadas nesse trabalho, as propriedades algébricas advindas deles e a notação em lógica de primeira-ordem da restrição à relação de acessibilidade.

Nome	Axioma	Propriedade	Condição
B_a	$\varphi \Rightarrow \boxed{a}\diamondsuit{a}\varphi$ $\diamondsuit{a}\boxed{a}\varphi \Rightarrow \varphi$	simétrica	$\forall w, w'(wR_aw' \Rightarrow w'R_aw)$
Ban_a	$\varphi \Rightarrow \boxed{a}\varphi$ $\diamondsuit{a}\varphi \Rightarrow \varphi$	modalmente banal	$\forall w, w'(wR_aw' \Rightarrow w = w')$
D_a	$\boxed{a}\varphi \Rightarrow \diamondsuit{a}\varphi$	serial	$\forall w \exists w'(wR_aw')$
F_a	$\diamondsuit{a}\varphi \Rightarrow \boxed{a}\varphi$	funcional	$\forall w, w', w''((wR_aw' \wedge wR_aw'') \Rightarrow w' = w'')$
T_a	$\varphi \Rightarrow \diamondsuit{a}\varphi$ $\boxed{a}\varphi \Rightarrow \varphi$	reflexiva	$\forall w(wR_aw)$
5_a	$\diamondsuit{a}\varphi \Rightarrow \boxed{a}\diamondsuit{a}\varphi$ $\diamondsuit{a}\boxed{a}\varphi \Rightarrow \boxed{a}\varphi$	euclideana	$\forall w, w', w''((wR_aw' \wedge wR_aw'') \Rightarrow w'R_aw'')$
$G1_a$	$\diamondsuit{a}\boxed{a}\varphi \Rightarrow \boxed{a}\diamondsuit{a}\varphi$	convergente	$\forall w, w', w''((wR_aw' \wedge wR_aw'')$ $\Rightarrow \exists w'''(w'R_aw''' \wedge w''R_aw'''))$
$G_a^{0,1,1,1}$	$\boxed{a}\varphi \Rightarrow \boxed{a}\diamondsuit{a}\varphi$ $\diamondsuit{a}\boxed{a}\varphi \Rightarrow \diamondsuit{a}\varphi$	0,1,1,1-convergente	$\forall w, w'(wR_aw' \Rightarrow \exists w''(wR_aw'' \wedge w'R_aw''))$
4_a	$\boxed{a}\varphi \Rightarrow \boxed{a}\boxed{a}\varphi$ $\diamondsuit{a}\diamondsuit{a}\varphi \Rightarrow \diamondsuit{a}\varphi$	transitiva	$\forall w, w', w''(wR_aw' \wedge w'R_aw'' \Rightarrow wR_aw'')$

Tabela 2.2: Lógicas confluentes: axiomas e propriedades

Capítulo 3

Cálculo baseado em resolução para K_n

Existem diferentes tipos de sistemas de dedução que determinam a teoremicidade de uma fórmula: dedução natural [7], tableaux [6], sistemas axiomáticos [6], etc. No caso deste projeto, o cálculo utilizado é o de resolução [1]. Resolução é um cálculo refutacional, isto é, nega-se a fórmula φ a ser provada e tenta-se encontrar um contradição. Diz-se que uma contradição foi encontrada quando um par de cláusulas C e $\neg C$ coexistem. Se uma contradição é encontrada sabemos que $\neg\varphi$ é insatisfatível e, portanto, φ é válida. Em geral, o cálculo é aplicado a fórmulas em forma clausal. Dado um conjunto de cláusulas S e uma cláusula C, uma dedução de C a partir de S no cálculo de resolução é uma sequência de cláusulas terminando em C gerada aplicando-se repetidamente as regras de resolução [11]. A seguir, será apresentado o cálculo baseado em resolução para as lógicas vistas no último capítulo.

3.1 Normalização

Para a construção do nosso cálculo, precisamos primeiramente que as fórmulas estejam em uma forma compatível com as regras de inferência, portanto aplicamos a função de tradução para gerar as mesmas fórmulas mas agora na forma normal separada (SNF).

Antes de se colocar as fórmulas na SNF, as fórmulas são colocadas na forma normal negada. Uma fórmula está na forma normal negada se o operador $\neg$ só está aplicado a variáveis proposicionais e os únicos operadores presentes são $\wedge, \vee, \boxed{a}$ e $\diamondsuit_a$. Uma fórmula pode ser transformada na forma normal negada através da aplicação das regras da Tabela 3.1 [9].

As regras de simplificação dadas na Tabela 3.2 podem ser utilizadas em qualquer etapa da reescrita de modo a simplificar os procedimentos da função de tradução [9].

$\varphi \Rightarrow \varphi' \rightarrow \neg\varphi \vee \varphi'$
$\neg(\varphi \Rightarrow \varphi') \rightarrow \varphi \wedge \neg\varphi'$
$\neg(\varphi \wedge \varphi') \rightarrow \neg\varphi \vee \neg\varphi'$
$\neg(\varphi \vee \varphi') \rightarrow \neg\varphi \wedge \neg\varphi'$
$\neg\neg\varphi \rightarrow \varphi$
$\neg\Box_a\varphi \rightarrow \Diamond_a\neg\varphi$
$\neg\Diamond_a\varphi \rightarrow \Box_a\neg\varphi$

Tabela 3.1: Regras de reescrita

$\varphi \wedge \varphi \rightarrow \varphi$	$\varphi \wedge \neg\varphi \rightarrow \mathbf{false}$	$\Box_a\mathbf{true} \rightarrow \mathbf{true}$
$\varphi \vee \varphi \rightarrow \varphi$	$\varphi \vee \neg\varphi \rightarrow \mathbf{true}$	$\Diamond_a\mathbf{false} \rightarrow \mathbf{false}$
$\varphi \wedge \mathbf{true} \rightarrow \varphi$	$\varphi \wedge \mathbf{false} \rightarrow \mathbf{false}$	$\neg\mathbf{true} \rightarrow \mathbf{false}$
$\varphi \vee \mathbf{false} \rightarrow \varphi$	$\varphi \vee \mathbf{true} \rightarrow \mathbf{true}$	$\neg\mathbf{false} \rightarrow \mathbf{true}$

Tabela 3.2: Regras de simplificação

A SNF_K possui a forma geral $\Box^* \bigwedge_i \varphi_i$, onde l_i, l, l' são literais, $a \in A_n$ e uma cláusula φ_i tem uma das quatro seguintes formas:

- Cláusula inicial: $\mathbf{start} \Rightarrow \bigvee_{i=1}^{n} l_i$
- Cláusula literal: $\mathbf{true} \Rightarrow \bigvee_{i=1}^{n} l_i$
- Cláusula modal positiva do agente a: $l' \Rightarrow \Box_a l$
- Cláusula modal negativa do agente a: $l' \Rightarrow \Diamond_a l$

onde **start** é uma nova constante cuja semântica é dada por $\langle M, w\rangle \models \mathbf{start}$ se, e somente se, $w = w_0$, onde M é um modelo. Ou seja, esta constante só é satisfeita na raiz do modelo. O operador universal, $\Box^*$, tem o seguinte significado: existe um modelo M, $M \models \Box^* \varphi$ se, e somente se, $\langle M, w\rangle \models \varphi$, para todo $w \in W$. Isto é, uma fórmula é universalmente satisfeita se é satisfeita em todos os mundos de um modelo.

Definição 11 Literal maximal [9]

Seja Φ o conjunto de cláusulas e L_Φ o conjunto de literais que ocorrem em Φ. Seja $\succ$ uma ordenação total e bem-fundada em L_Φ. Os literais são ordenados da forma $\neg p \succ p$, $p \succ \neg q$ sempre que $p \succ q$ para todo $p, q \in P$.

Um literal l é dito maximal em uma cláusula $C \vee l$, onde C é uma cláusula quando não existe l' em C tal que $l' \succ l$.

Para normalizar as fórmulas é necessário aplicar a função de normalização recursivamente até o caso base. Essa função está definida a seguir.

Definição 12 Função de tradução [1]

Tome φ na forma normal negada. A função τ é definida como:

- $\tau(\varphi) = \Box^*(\mathbf{start} \Rightarrow f) \wedge \tau_1(\Box^*(f \Rightarrow \varphi))$

τ_1 remove operadores clássicos no processo de normalização.

- $\tau_1(\Box^*(x \Rightarrow \phi \wedge \psi)) = \tau_1(\Box^*(x \Rightarrow \phi)) \wedge \tau_1(\Box^*(x \Rightarrow \psi))$

τ_1 renomeia fórmulas complexas no escopo de operadores modais.

- $\tau_1(\Box^*(x \Rightarrow [a]\phi)) = \tau_1(\Box^*(x \Rightarrow [a]y)) \wedge \tau_1(\Box^*(y \Rightarrow \phi))$
- $\tau_1(\Box^*(x \Rightarrow \langle a \rangle \phi)) = \tau_1(\Box^*(x \Rightarrow \langle a \rangle y)) \wedge \tau_1(\Box^*(y \Rightarrow \phi))$

τ_1 é usada desta forma caso A seja uma fórmula complexa.

- $\tau_1(\Box^*(x \Rightarrow D \vee \phi)) = \tau_1(\Box^*(x \Rightarrow D \vee y)) \wedge \tau_1(\Box^*(y \Rightarrow \phi))$

Se D é uma disjunção de literais

- $\tau_1(\Box^*(x \Rightarrow D)) = \Box^*(\mathbf{true} \Rightarrow \neg x \vee D)$

Se D é um literal modal

- $\tau_1(\Box^*(x \Rightarrow D)) = \Box^*(x \Rightarrow D)$

Exemplo 6 Aplicação do renomeamento

Seja $\phi = (\Box p \vee \mathbf{false}) \wedge q \Rightarrow r$. Queremos gerar sua forma normal separada. Primeiramente iremos colocar a fórmula na forma normal separada. Então devemos aplicar as simplificações à função de tradução τ à fórmula.

$(\Box p \vee \mathbf{false}) \wedge q \Rightarrow r$	
$\Box p \wedge q \Rightarrow r$	[simplificação]
$\neg(\Box p \wedge q) \vee r$	[transformação na forma normal negada]
$\neg\Box p \vee \neg q \vee r$	[transformação na forma normal negada]
$\Diamond \neg p \vee \neg q \vee r$	[transformação na forma normal negada]

Agora, apliquemos a função:

$$
\begin{aligned}
\tau((\Diamond\neg p \vee \neg q) \vee r) \quad &= \\
&= \Box^*(\mathbf{start} \Rightarrow f) \wedge \tau_1(\Box^*(f \Rightarrow \Diamond\neg p \vee \neg q \vee r)) \\
&= \Box^*(\mathbf{start} \Rightarrow f) \wedge \tau_1(\Box^*(f \Rightarrow \Diamond\neg p \vee \neg q \vee r)) \\
&= \Box^*(\mathbf{start} \Rightarrow f) \wedge \tau_1(\Box^*(f \Rightarrow x \vee \neg q \vee r)) \wedge \tau_1(\Box^*(x \Rightarrow \Diamond\neg p)) \\
&= \Box^*(\mathbf{start} \Rightarrow f) \wedge \Box^*(\mathbf{true} \Rightarrow \neg f \vee x \vee \neg q \vee r) \wedge \tau_1(\Box^*(x \Rightarrow \Diamond\neg p)) \\
&= \Box^*(\mathbf{start} \Rightarrow f) \wedge \Box^*(\mathbf{true} \Rightarrow \neg f \vee x \vee \neg q \vee r) \wedge \Box^*(x \Rightarrow \Diamond\neg p)
\end{aligned}
$$

3.2 Regras de inferência

Agora que as fórmulas já estão na SNF a próxima etapa para determinar a satisfatibilidade do conjunto de cláusulas é a aplicação das regras de inferência. Para as próximas definições tome l, l', l_i, l'_i como literais e D e D' como disjunções de literais.

3.2.1 Regras de inferência em K_n

Cláusulas iniciais podem ser resolvidas entre si ou com cláusulas literais a partir das regras **IRES1** e **IRES2**.

$$
\textbf{[IRES1]} \quad \frac{\Box^*(\mathbf{true} \Rightarrow D \vee l) \qquad \Box^*(\mathbf{start} \Rightarrow D' \vee \neg l)}{\Box^*(\mathbf{start} \Rightarrow D \vee D')} \qquad
\textbf{[IRES2]} \quad \frac{\Box^*(\mathbf{start} \Rightarrow D \vee l) \qquad \Box^*(\mathbf{start} \Rightarrow D' \vee \neg l)}{\Box^*(\mathbf{start} \Rightarrow D \vee D')}
$$

Cláusulas literais podem ser resolvidas entre si a partir de **LRES**.

$$
\textbf{[LRES]} \quad \frac{\Box^*(\mathbf{true} \Rightarrow D \vee l) \qquad \Box^*(\mathbf{true} \Rightarrow D' \vee \neg l)}{\Box^*(\mathbf{true} \Rightarrow D \vee D')}
$$

Cláusulas modais se resolvem aos pares e aos trios por **MRES** e **GEN2**.

$$[\textbf{MRES}]\quad \frac{\Box^*(l_1 \Rightarrow \boxed{a} l) \quad \Box^*(l_2 \Rightarrow \langle\!\!\diamond\!\!\rangle_a \neg l)}{\Box^*(\textbf{true} \Rightarrow \neg l_1 \vee \neg l_2)} \qquad [\textbf{GEN2}]\quad \frac{\Box^*(l'_1 \Rightarrow \boxed{a} l_1) \quad \Box^*(l'_2 \Rightarrow \boxed{a} \neg l_1) \quad \Box^*(l'_3 \Rightarrow \langle\!\!\diamond\!\!\rangle_a l_2)}{\Box^*(\textbf{true} \Rightarrow \neg l'_1 \vee \neg l'_2 \vee \neg l'_3)}$$

Diversas cláusulas modais são resolvidas por **GEN1** e **GEN3**.

$$[\textbf{GEN1}]\quad \frac{\begin{array}{c}\Box^*(l'_1 \Rightarrow \boxed{a} \neg l_1) \\ \vdots \\ \Box^*(l'_n \Rightarrow \boxed{a} \neg l_n) \\ \Box^*(l' \Rightarrow \langle\!\!\diamond\!\!\rangle_a l) \\ \Box^*(\textbf{true} \Rightarrow l_1 \vee \ldots \vee l_n \vee \neg l)\end{array}}{\Box^*(\textbf{true} \Rightarrow \neg l'_1 \vee \ldots \vee \neg l'_n \vee \neg l')}$$

$$[\textbf{GEN3}]\quad \frac{\begin{array}{c}\Box^*(l'_1 \Rightarrow \boxed{a} \neg l_1) \\ \vdots \\ \Box^*(l'_n \Rightarrow \boxed{a} \neg l_n) \\ \Box^*(l' \Rightarrow \langle\!\!\diamond\!\!\rangle_a l) \\ \Box^*(\textbf{true} \Rightarrow l_1 \vee \ldots \vee l_n)\end{array}}{\Box^*(\textbf{true} \Rightarrow \neg l'_1 \vee \ldots \vee \neg l'_n \vee \neg l')}$$

3.2.2 Regras de inferência para lógicas confluentes

Além das regras de inferência descritas em [1], o artigo [2] traz o método de resolução para lógicas confluentes. Uma lógica confluente $\mathsf{K}_n^{p,q,r,s}$ é uma lógica sem restrições K_n acrescida de axiomas da forma $\langle\!\!\diamond\!\!\rangle_a^p \boxed{a}^q \varphi \Rightarrow \boxed{a}^r \langle\!\!\diamond\!\!\rangle_a^s$, em que $a \in A_n$, $\varphi \in \mathsf{WFF}$ e $p, q, r, s \in \mathbb{N}$. Os operadores iterados são definidos da seguinte forma: $\boxed{a}^0 \varphi = \langle\!\!\diamond\!\!\rangle_a^0 \varphi = \varphi$, $\boxed{a}^n \varphi = \boxed{a}^{n-1} \boxed{a} \varphi$ e $\langle\!\!\diamond\!\!\rangle_a^n \varphi = \langle\!\!\diamond\!\!\rangle_a^{n-1} \langle\!\!\diamond\!\!\rangle_a \varphi$. Em [2] são consideradas somente lógicas confluentes tais que $p, q, r, s \in \{0, 1\}$. As novas regras de inferência seguem as formas mostradas a seguir, onde C é uma conjunção de literais e l e l' são literais.

$$[\textbf{RES}_a^{p,1,r,s}]\quad \frac{\Box^*(l \Rightarrow \boxed{a} l')}{\Box^*(\langle\!\!\diamond\!\!\rangle_a^p l \Rightarrow \boxed{a}^r \langle\!\!\diamond\!\!\rangle_a^s l')} \qquad [\textbf{RES}_a^{p,0,r,s}]\quad \frac{\Box^*(C \Rightarrow \langle\!\!\diamond\!\!\rangle_a^p l')}{\Box^*(C \Rightarrow \boxed{a}^r \langle\!\!\diamond\!\!\rangle_a^s l')}$$

A partir desse novo formato, são introduzidas novas regras de inferência para resolver fórmulas nas diferentes extensões de lógica dispostas na Tabela 2.2. Estas regras de inferência são apresentadas na Tabela 3.3, onde na primeira coluna estão os nomes dos axiomas e na segunda coluna as regras de inferência correspondentes ao axiomas apresentados na Tabela 2.2.

T_a	$[\mathbf{RES}_a^{0,0,0,1}]$	$\dfrac{\Box^*(\mathbf{true} \Rightarrow D \vee l)}{\Box^*(\neg D \Rightarrow \langle a \rangle l)}$	$[\mathbf{RES}_a^{0,1,0,0}]$	$\dfrac{\Box^*(l \Rightarrow [a] l')}{\Box^*(\mathbf{true} \Rightarrow \neg l \vee l')}$
$G_a^{0,1,1,1}$	$[\mathbf{RES}_a^{0,1,1,1}]$	$\dfrac{\Box^*(l \Rightarrow [a] l')}{\Box^*(l \Rightarrow [a] pos_{a,l'})}$	$[\mathbf{RES}_a^{1,1,0,1}]$	$\dfrac{\Box^*(l \Rightarrow [a] l')}{\Box^*(pos_{a,l} \Rightarrow \langle a \rangle l')}$
Ban_a	$[\mathbf{RES}_a^{0,0,1,0}]$	$\dfrac{\Box^*(\mathbf{true} \Rightarrow D \vee l)}{\Box^*(\neg D \Rightarrow [a] l)}$	$[\mathbf{RES}_a^{1,0,0,0}]$	$\dfrac{\Box^*(l \Rightarrow \langle a \rangle l')}{\Box^*(\mathbf{true} \Rightarrow \neg l \vee l')}$
F_a	$[\mathbf{RES}_a^{1,0,1,0}]$	$\dfrac{\Box^*(l \Rightarrow \langle a \rangle l')}{\Box^*(l \Rightarrow [a] l')}$		
B_a	$[\mathbf{RES}_a^{0,0,1,1}]$	$\dfrac{\Box^*(\mathbf{true} \Rightarrow D \vee l)}{\Box^*(\neg D \Rightarrow [a] pos_{a,l})}$	$[\mathbf{RES}_a^{1,1,0,0}]$	$\dfrac{\Box^*(l \Rightarrow [a] l')}{\Box^*(\neg l' \Rightarrow [a] \neg l)}$
5_a	$[\mathbf{RES}_a^{1,0,1,1}]$	$\dfrac{\Box^*(l \Rightarrow \langle a \rangle l')}{\Box^*(l \Rightarrow [a] pos_{a,l'})}$	$[\mathbf{RES}_a^{1,1,1,0}]$	$\dfrac{\Box^*(l \Rightarrow [a] l')}{\Box^*(pos_{a,l} \Rightarrow [a] l')}$
D_a	$[\mathbf{RES}_a^{0,1,0,1}]$	$\dfrac{\Box^*(l \Rightarrow [a] l')}{\Box^*(l \Rightarrow \langle a \rangle l')}$		
$G1_a$	$[\mathbf{RES}_a^{1,1,1,1}]$	$\dfrac{\Box^*(l \Rightarrow [a] l')}{\Box^*(pos_{a,l} \Rightarrow [a] pos_{a,l'})}$		
4_a	$[\mathbf{4}_a]$	$\dfrac{\Box^*(l \Rightarrow [a] l')}{\begin{array}{l}\Box^*(\mathbf{true} \Rightarrow \neg l \vee nec_{a,l'}) \\ \Box^*(nec_{a,l'} \Rightarrow [a] l') \\ \Box^*(nec_{a,l'} \Rightarrow [a] nec_{a,l'})\end{array}}$		

Tabela 3.3: Regras de inferência

Na Tabela 3.3, l e l' são literais e o símbolo proposicional $pos_{a,l}$ é um novo nome para $\langle a \rangle l$, ou seja, $pos_{a,l} \Leftrightarrow \langle a \rangle l$. Da mesma forma, $nec_{a,l}$ é um novo nome para $[a] l$, ou seja $nec_{a,l} \Leftrightarrow [a] l$.

O próximo capítulo apresentará como foi feita a implementação do cálculo aqui descrito.

Capítulo 4

KSP

KSP é um provador automático para fórmulas multimodais que implementa o cálculo apresentado no Capítulo 3. Este capítulo tem como intenção apresentar a implementação do cálculo como um programa computador. A Figura 4.1, adaptada de [8], apresenta o algoritmo do KSP.

A entrada é dada através de dois arquivos: um arquivo de configuração e um arquivo de fórmulas. O arquivo de configuração pode acrescentar novas regras de inferência ao cálculo, selecionar otimizações que serão feitas ao montar as estruturas de dados utilizadas pelo provador, selecionar normalizações, selecionar se o cálculo será aplicado local ou globalmente e a quantidade de provas a ser apresentada, entre outros. Já o arquivo de fórmulas traz conjuntos de fórmulas que serão processadas de modo a apresentar sua satisfatibilidade ou não.

O processador da entrada foi construído na linguagem C com o gerador de analisadores léxicos Flex e com o gerador de analisadores sintáticos Bison. A análise do arquivo de configuração é relativamente simples. As configurações são interpretadas alterando-se valores de parâmetros no programa.

Já a etapa de análise do arquivo de fórmulas é mais complexa. Cada símbolo proposicional lido recebe um número inteiro identificador associado a ele e é adicionado à tabela de símbolos proposicionais. Da mesma forma, cada agente de operador modal lido recebe um identificador inteiro e é adicionado à tabela de símbolos de agentes modais.

A árvore sintática abstrata é montada como uma árvore ternária duplamente encadeada onde cada nó é uma fórmula, ou um conjunto de fórmulas. Cada nó proposicional na árvore contém a identificação da tabela de símbolos, além de conter mais detalhes sobre aquele nó: tipo, profundidade modal, nível modal, polaridade, nome após renomeamento e valores utilizados na normalização. A entrada também é transformada em quatro tabelas de símbolos: duas para símbolos proposicionais e duas para agentes, com chaves nos lexemas e na representação interna.

Tendo a árvore em memória é possível fazer a normalização das fórmulas na forma normal separada. O processo de normalização gera as cláusulas. As cláusulas são armazenadas em três tabelas de dispersão diferentes: tabelas de cláusulas iniciais, de cláusulas modais e de cláusulas literais. Estas tabelas de dispersão consistem de várias tabelas aninhadas, indexadas conforme o tipo de informação sendo tratada. Para cláusulas literais, os níveis das tabelas de dispersão são chaveados pelo nível modal em que ocorrem, pelos seus literais maximais e pelo seu tamanho. Para cláusulas modais, o chaveamento das tabelas de dispersão é dado pelo nível modal, pelo índice do agente, pelo tipo (negativa ou positiva) e, finalmente, pelos seus literais maximais. Em ambos os casos, conflitos são resolvidos no último nível através da implementação de listas.

O algoritmo da Figura 4.1 tem como entrada uma fórmula e como saída a solução se ela é satisfatível ou não.

```
1  Algoritmo: Busca por provas do K_S P
2  processamento_da_entrada;
3  normalização_na_snf;
4  pré-processamento_de_cláusulas;
5  enquanto (Γ^lit_global ≠ ∅) faça
6       cláusula ← seleciona_cláusula();
7       se (não redundante(cláusula)) então
8            GEN1(cláusula);
9            GEN3(cláusula);
10           LRES(cláusula);
11           Λ^lit ← Λ^lit ∪ {cláusula};
12      fim se
13      Γ^lit_global ← Γ^lit_global \ {cláusula};
14      se (true ⇒ false ∈ Γ^lit_0) então retorna insatisfatível; fim se
15 fim enquanto
16 enquanto (Γ^lit_inicial ≠ ∅) faça
17      cláusula ← seleciona_cláusula();
18      se (não redundante(cláusula)) então
19           IRES1(cláusula);
20           IRES2(cláusula);
21           LRES(cláusula);
22           Λ^lit ← Λ^lit ∪ {cláusula};
23      fim se
24      Γ^lit_inicial ← Γ^lit_inicial \ {cláusula};
25      se (start ⇒ false ∈ Γ^lit_inicial) então retorna insatisfatível; fim se
26 fim enquanto
27 retorna satisfatível;
```

Figura 4.1: Algoritmo do $K_S P$

O conjunto de cláusulas Δ é particionado em seis conjuntos conjuntos, $\Gamma^{lit}_{global}, \Gamma^{modal}_{global}$,

$\Gamma^{lit}_{inicial}$, Λ^{lit}_{global}, Λ^{modal}_{global}, $\Lambda^{lit}_{inicial}$ e Λ^{modal}_{global}. Chamamos de Γ o conjunto $\Gamma = \Gamma^{lit}_{global} \cup \Gamma^{modal}_{global} \cup \Gamma^{lit}_{inicial}$ e $\Lambda = \Lambda^{lit}_{global} \cup \Lambda^{modal}_{global} \cup \Lambda^{lit}_{inicial}$. Os conjuntos Γ, Λ e Δ mantêm a seguinte relação: $\Lambda = \Delta \backslash \Gamma$. O conjunto Γ é chamado de conjunto de suporte (*set of support/SOS*) e é o conjunto das cláusulas não processadas. Já Λ é chamado de conjunto utilizável, que é o conjunto das cláusulas que já foram processadas. Na etapa do pré-processamento de cláusulas são aplicados os axiomas de confluência. Percorre-se Γ cláusula a cláusula e tenta-se gerar novas cláusulas a partir dos axiomas selecionados por meio dos arquivos de configuração. As novas cláusulas são acrescentadas a Γ, mas na etapa do pré-processamento Λ permanece vazio.

O laço principal divide-se em duas fases: saturação de cláusulas globais e modais (Linhas 5 a 15) e saturação de cláusulas iniciais (Linhas 16 a 26). Durante o processo de saturação global, cláusulas em Γ^{lit}_{global}, onde cláusulas literais não processadas estão armazenadas, são resolvidas com cláusulas no conjunto Λ^{lit}_{global}, que é o conjunto de cláusulas literais já processadas (Linha 10); e com cláusulas em Λ^{modal}_{global}, que contém o conjunto de cláusulas modais (Linhas 8 e 9). A cláusula escolhida em cada passagem do laço é movida para o conjunto Λ^{lit}_{global} quando todas as regras forem aplicadas. Se uma contradição, na forma **true** $\Rightarrow$ **false** ou **start** $\Rightarrow$ **false** for encontrada, o processo termina com a determinação da insatisfatibilidade do problema. Caso contrário, este laço se repete até que não se possa mais gerar cláusulas literais e o processo de saturação de cláusulas iniciais é realizado. Para resolução inicial, o processamento é análogo, onde cláusulas iniciais processadas são armazenadas em $\Lambda^{lit}_{inicial}$ e as não processadas em $\Gamma^{lit}_{inicial}$.

4.1 Pré-processamento de cláusulas

Regras de convergência não são aplicadas nos laços de cálculo inicial ou global do algoritmo da Figura 4.1, mas durante a fase de pré-processamento. A implementação feita neste trabalho ocorre especificamente na etapa de aplicação das regras de inferência da Tabela 3.3. Este é o único momento em que as regras de confluência são aplicadas. O algoritmo de aplicação das regras confluentes está de acordo com a Figura 4.2.

Os conjuntos $\Omega^{modal}, \Omega^{lit}, \Phi^{modal}$ e Φ^{lit} são implementados usando tabelas de dispersão. Anteriormente cláusulas modais e literais geradas durante o pré-processamento das regras de confluência eram armazenadas em uma mesma estrutura: $\Omega = \Omega^{modal} \cup \Omega^{lit}$ e $\Phi = \Phi^{modal} \cup \Phi^{lit}$. Estes conjuntos, Ω e Φ, eram implementados com listas encadeadas.

É importante observar que para garantir a terminação do algoritmo da Figura 4.2, as funções gera_cláusulas_modais() e gera_cláusulas_literais() precisam gerar as cláusulas e verificar se elas não estão duplicatas nos conjuntos Γ^{modal} e Γ^{lit}. Se forem duplicatas, elas são descartadas. As funções retornam apenas cláusulas não repetidas.

1 **Algoritmo**: Pré-processamento de axiomas confluentes
2 $\Omega^{lit} \leftarrow \varnothing$;
3 $\Omega^{modal} \leftarrow \varnothing$;
4 **para** (cláusula $\in \Gamma^{modal}$) **faça**
5 $\quad \Omega^{modal} \leftarrow \Omega^{modal} \cup$ gera_cláusulas_modais(cláusula);
6 $\quad \Omega^{lit} \leftarrow \Omega^{lit} \cup$ gera_cláusulas_literais(cláusula);
7 **fim para**
8 **para** (cláusula $\in \Gamma^{lit}$) **faça**
9 $\quad \Omega^{modal} \leftarrow \Omega^{modal} \cup$ gera_cláusulas_modais(cláusula);
10 $\quad \Omega^{lit} \leftarrow \Omega^{lit} \cup$ gera_cláusulas_literais(cláusula);
11 **fim para**
12 **enquanto** ($\Omega^{modal} \cup \Omega^{lit} \neq \varnothing$) **faça**
13 $\quad \Phi^{lit} \leftarrow \varnothing$;
14 $\quad \Phi^{modal} \leftarrow \varnothing$;
15 $\quad$ **para** (cláusula $\in \Omega^{modal} \cup \Omega^{lit}$) **faça**
16 $\quad\quad \Phi^{modal} \leftarrow \Phi^{modal} \cup$ gera_cláusulas_modais(cláusula);
17 $\quad\quad \Phi^{lit} \leftarrow \Phi^{lit} \cup$ gera_cláusulas_literais(cláusula);
18 $\quad$ **fim para**
19 $\quad$ **para** (cláusula $\in \Omega^{modal} \cup \Omega^{lit}$) **faça**
20 $\quad\quad$ **se** (cláusula $\notin \Gamma^{lit} \cup \Gamma^{modal}$) **então**
21 $\quad\quad\quad \Gamma^{lit} \cup \Gamma^{modal} \leftarrow \Gamma^{lit} \cup \Gamma^{modal} \cup \{\text{cláusula}\}$
22 $\quad \Gamma^{modal} \leftarrow \Gamma^{modal} \cup \Omega^{modal}$;
23 $\quad \Gamma^{lit} \leftarrow \Gamma^{lit} \cup \Omega^{lit}$;
24 $\quad \Omega^{modal} \leftarrow \Phi^{modal}$;
25 $\quad \Omega^{lit} \leftarrow \Phi^{lit}$;
26 **fim enquanto**

Figura 4.2: Algoritmo da aplicação de regras de confluência

Capítulo 5

Avaliação experimental

A parte de avaliação experimental deste trabalho foi separada em dois experimentos. O primeiro deles foi a verificação da correção das regras de inferência para lógicas confluentes no provador KSP. O segundo foi a refatoração do código-fonte para alterar as estruturas de dados para estruturas mais adequadas e compatíveis com o padrão do programa e possivelmente galgar desempenho com esta alteração.

O experimento foi conduzido em um ambiente experimental com as seguintes características:

- Linux Ubuntu 18.04.3 LTS
- Intel(R) Core(TN) i3-6006U CPU @ 2.00GHz
- 4 GB RAM

Para ambos os experimentos, os testes foram feitos com o conjunto de fórmulas providos em [12]. Esta bateria de testes consiste em um conjunto de 378 fórmulas, divididas em 9 categorias diferentes. Cada categoria possui 42 fórmulas, das quais 21 são satisfatíveis e 21 são insatisfatíveis. A dificuldade de resolução é crescente dentro de cada categoria.

5.1 Experimento 1 - Correção

O primeiro experimento consistiu na verificação da correção da implementação das regras de confluência apresentadas na Tabela 3.3. Como se pode observar, os axiomas $T_a, Ban_a, B_a, G_a^{0,1,1,1}$ e $G1_a$ possuem duas regras de inferência diferentes. Por exemplo, o axioma T_a possui as regras de inferência $\mathbf{RES}_a^{0,0,0,1}$ e $\mathbf{RES}_a^{0,1,0,0}$. Embora as regras sejam distintas, a aplicação do cálculo a uma fórmula deve prover o mesmo resultado de satisfatibilidade, independentemente se uma regra, ou outra, ou ambas estiverem sendo usadas.

O conjunto de testes foi configurado com tempo de expiração de 10 segundos. Sendo assim, cada regra da Tabela 3.3 foi testada individualmente, contra as 378 fórmulas. Os resultados possíveis foram **satisfatível**, **insatisfatível** e **expirado**, quando o tempo de 10 segundos era esgotado e nenhum resultado produzido.

Os resultados do experimento foram compilados em pares para cada axioma a fim de identificar inconsistências internas. Se o cálculo acrescido de $\mathbf{RES}_a^{0,0,0,1}$ retorna **satisfatível** para uma fórmula φ e acrescido de $\mathbf{RES}_a^{0,1,0,0}$ retorna **insatisfatível** para φ sabemos que há uma inconsistência, porque ambas as regras representam o axioma T_a e deveriam prover o mesmo resultado. Então, foi feita uma busca nos dados a fim de identificar estes pares **satisfatível/insatisfatível** indicando essas inconsistências e a quais regras elas pertencem.

Foram encontradas inconsistências apenas no axioma Ban_a. As regras de inferência $\mathbf{RES}_a^{0,0,1,0}$ e $\mathbf{RES}_a^{1,0,0,0}$ se mostraram inconsistentes entre si para as fórmulas **k__lin__n.01, k__path__n.01 e k__path__n.03**. Foi feita uma prova manual para a fórmula **k__lin__n.01** a fim determinar qual das regras estava incorreta. Descobriu-se que o resultado provido por $\mathbf{RES}_a^{1,0,0,0}$ apresentou o resultado correto (insatisfatível nesta lógica) e $\mathbf{RES}_a^{0,0,1,0}$ o resultado incorreto.

Por isso, o código do KSP foi investigado a fim de verificar falhas de implementação na regra $\mathbf{RES}_a^{0,0,1,0}$. Não foi encontrada falha na implementação. Como a regra $\mathbf{RES}_a^{0,0,1,0}$ não possui prova de completude, pode ser possível que de fato ela não seja uma regra completa. Devido a isso, não se viu mais necessidade de empreender esforços de correção no provador e assim se encerrou a primeira etapa experimental.

5.2 Experimento 2 - Refatoração

O segundo experimento consistiu na refatoração do código do programa, na etapa do pré-processamento, na aplicação de regras de inferência de confluência. A implementação das regras de inferência de confluência, o manuseio de cláusulas e a inserção de cláusulas no conjunto Γ^{lit} e Γ^{modal} (conjuntos de suporte) eram feitas utilizando listas ligadas simples. Porém, o programa manipula cláusulas em conjuntos e conjuntos não possuem repetições. Para checar repetições em uma lista ligada, é necessário percorrê-la até encontrar o elemento desejado ou terminar de percorrê-la. Ao usarmos uma função de dispersão, essa busca custa apenas o tempo de cálculo da função.

Então, a ideia do experimento era tentar ganhar desempenho no programa utilizando tabelas de dispersão em vez de listas. Apesar de ambas terem tempo de percorrimento linear, a complexidade assintótica para busca na lista ligada é $O(n)$ enquanto na tabela de dispersão é em tempo constante $O(1)$ [13]. Dessa forma supôs-se ser possível economizar

Axioma	**Regra de inferência**
T_a	res_0001 res_0100
Ban_a	res_0010 res_1000
B_a	res_0011 res_1100
D_a	res_0101
4_a	res_0120
$G_a^{0,1,1,1}$	res_0111 res_1101
F_a	res_1010
5_a	res_1011 res_1110
$G1_a$	res_1111

Tabela 5.1: Regras de inferência de configuração do programa

tempo nas buscas, principalmente quando o conjunto de cláusulas fica muito extenso. A tabela de cláusulas literais é um conjunto de tabelas aninhadas, indexadas por nível modal, literal maximal e tamanho. A tabela de cláusulas modais também é um conjunto de tabelas aninhas e indexadas por nível modal, índice do agente, tipo (positiva ou negativa) e literal maximal. Tanto para a tabela de cláusulas modais quanto de literais as colisões são resolvidas com a implementação de listas.

A avaliação de correção e de desempenho dessas alterações foi feita de forma semelhante à descrita na etapa anterior. O mesmo conjunto de testes com 378 fórmulas foi executado para cada uma das regras de inferência confluentes com 60 segundos de tempo limite. Mas desta vez foi necessário executar os testes para dois programas diferentes: o provador antes das alterações, que chamaremos de programa base, e depois das alterações. Desta forma é possível testar a correção da nova implementação e traçar um parâmetro de comparação para avaliar o impacto do desempenho das novas estruturas de dados no programa.

É possível escolher 14 funções diferentes de cálculo confluente, de acordo com as regras implementadas. Os nomes destas regras, utilizadas na configuração do provador, são apresentadas na Tabela 5.1 com a devida correspondência para os nomes dos axiomas apresentados na Tabela 3.3.

Cada uma destas 14 configurações foi testada individualmente contra as 378 fórmulas para cada uma das implementações, testando um total de 10584 fórmulas. A Tabela 5.2 indica a quantidade fórmulas solucionadas e o tempo total de execução para cada regra tanto no programa base, quanto no programa resultante da refatoração, que chamaremos de programa atual.

Regra	**# Base**	**# Atual**	**Tempo Base**	**Tempo Atual**
res_0001	30	30	72.26	71.30
res_0100	254	254	700.77	699.76
res_0010	27	27	52.47	52.34
res_1000	227	226	1163.15	1135.06
res_0011	43	43	233.42	234.85
res_1100	229	229	742.50	749.69
res_0101	267	267	974.84	963.43
res_0120	177	177	712.52	783.33
res_0111	58	58	348.08	347.26
res_1101	56	56	326.45	324.28
res_1010	108	106	1405.98	1365.63
res_1011	206	206	1058.64	1060.78
res_1110	184	182	1240.05	1193.48
res_1111	61	61	385.82	383.41

Tabela 5.2: Total de fórmulas solucionadas com tempo total em segundos para cada uma das implementações.

A primeira coisa a se notar é que o programa implementado com tabelas de dispersão resolve menos fórmulas do que o programa base. São 5 fórmulas que a nova implementação não soluciona. A programa atual provê resultado de 1922 fórmulas das 5292 testadas, enquanto o antigo solucionava 1927 fórmulas. É interessante perceber também que o conjunto solução do programa atual é um subconjunto do conjunto solução do antigo, ou seja, não houve nenhuma fórmula com a implementação de tabelas de dispersão que o programa antigo não foi capaz de resolver.

As regras **res_1000, res_1010, res_1110** solucionaram mais fórmulas na implementação base que na atual. Entretanto, esta diferença de desempenho não é significativa. Várias soluções a mais da implementação base se deram nos últimos três segundos do limite máximo de tempo utilizado no experimento. Porém, algumas regras tiveram uma redução drástica de desempenho com a alteração das estruturas de dados. Regras como **res_1010** tiveram incrementos de 30 segundos na versão atual e casos onde o tempo de solução atual chega a nove vezes o tempo base. Mas vale ressaltar que esses são casos limítrofes, em casos normais houve pouca variação.

A Tabela 5.3 mostra o tempo médio em segundos de solução das fórmulas por categoria. Nota-se que com a nova implementação ganha-se alguns centésimos de tempo médio em seis regras de inferência, enquanto perde-se em sete e mantém-se o desempenho em uma.

Regra	**Base**	**Atual**
res_0001	2.41	2.38
res_0100	2.76	2.75
res_0010	1.94	1.94
res_1000	4.88	5.02
res_0011	5.43	5.46
res_1100	3.24	3.27
res_0101	3.65	3.61
res_0120	4.03	4.43
res_0111	6.00	5.99
res_1101	5.83	5.79
res_1010	12.39	12.88
res_1011	5.14	5.15
res_1110	6.18	6.56
res_1111	6.32	6.29

Tabela 5.3: Tempo médio de solução de fórmula por família em segundos

Capítulo 6

Conclusão

Foram feitos dois experimentos na etapa prática do trabalho: a verificação de correção e a refatoração do código. Na etapa de verificação, as regras oriundas de um mesmo axioma tiveram seus resultados comparados. A única inconsistência encontrada neste experimento foi para a regra de inferência que não possui prova de completude. Devido a isso, é fundamental notar como o aspecto teórico foi importante tanto para ajudar a solucionar o problema do experimento, quanto para poupar os esforços de depurar implementação que não temos garantia de produção do resultado.

O segundo experimento, de refatoração do código visando otimizar o tempo solução, teve resultados interessantes. Os ganhos e perdas variaram bastante de acordo com as regras de inferência e com as famílias de fórmulas que elas solucionavam. Analisando o caso geral, os incrementos e decrementos de desempenho foram marginais, mas os casos limite são no mínimo curiosos. Saltar de 2 segundos para 30 segundos no tempo de solução de uma fórmula, apenas porque as estruturas de dados foram alteradas, não parece um comportamento correto, apesar de que as soluções estão todas compatíveis entre os dois testes.

Uma investigação preliminar foi feita acerca desse comportamento errático do programa e, ao que tudo indica, isso é fruto da ordenação das cláusulas no conjunto Γ. Entretanto, é preciso estudar isso mais profundamente. Provavelmente isso é passível de correção, otimização ou pode ser até que seja possível implementar uma heurística de ordenação de cláusulas nas configurações do programa. Isto fica como sugestão para trabalhos futuros.

É difícil categorizar o saldo como positivo ou negativo, houve ganhos de um lado e perdas de outro. Além disso, os testes foram feitos com apenas uma regra por vez. Um aprimoramento do experimento seria testar combinações de regras diferentes, trazendo assim, uma medida mais significativa. Junto com a sugestão do parágrafo anterior, esta também é uma sugestão de continuidade desse trabalho em desenvolvimento futuro.

Referências

[1] Nalon, Cláudia e Clare Dixon: *Clausal Resolution for Normal Modal Logics*. Journal of Algorithms, 62(3–4):117–134, julho 2007, ISSN 0196-6774. `https://doi.org/10.1016/j.jalgor.2007.04.001`. 1, 11, 13, 15

[2] Nalon, Cláudia, João Marcos e Clare Dixon: *Clausal Resolution for Modal Logics of Confluence*. Em Demri, Stéphane, Deepak Kapur e Christoph Weidenbach (editores): *7th International Joint Conference on Automated Reasoning, LNAI, Automated Reasoning*, páginas 322–336, Cham, 2014. Springer International Publishing, ISBN 978-3-319-08587-6. 1, 10, 15

[3] Bochmann, Gregor V.: *Hardware specification with temporal logic: An example*. IEEE Transactions on Computers, C-31(3):223–231, 1982. 1

[4] Hailpern, Brent T.: *Verifying Concurrent Processes Using Temporal Logic*. Springer-Verlag, Berlin, Heidelberg, 1982, ISBN 0387112057. 1

[5] Garcez, Artur, Luís Lamb e Dov Gabbay: *Connectionist modal logic: Representing modalities in neural networks*. Theoretical Computer Science, 371:34–53, fevereiro 2007. 1

[6] Fitting, Melvin e Richard L. Mendelsohn: *First-Order Modal Logic*. Kluwer Academic Publishers, 1998. 1, 11

[7] Martins, Ana Teresa e Lília Ramalho Martins: *Natural deduction for full S5 modal logic with weak normalization*. Em Queiroz, Ruy de, Angus Macintyre e Guilherme Bittencourt (editores): *Proceedings of the 12th Workshop on Logic, Language, Information and Computation (WoLLIC 2005)*, volume 143, páginas 129–140, 2006. 1, 11

[8] Nalon, Cláudia, Ullrich Hustadt e Clare Dixon: *$K_{\mathcal{S}}P$ A Resolution-Based Theorem Prover for K_n: Architecture, Refinements, Strategies and Experiments*. Journal of Automated Reasoning, 64(3):461–484, 2020. 2, 17

[9] Nalon, Cláudia, Clare Dixon e Ullrich Hustadt: *Modal Resolution: Proofs, Layers, and Refinements*. ACM Trans. Comput. Logic, 20(4), agosto 2019, ISSN 1529-3785. `https://doi.org/10.1145/3331448`. 5, 6, 11, 12

[10] Blackburn, Patrick, Maarten de Rijke e Yde Venema: *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001. 10

[11] Casanova, Marco, Antonio Furtado e Fernando Giorno: *Programação em lógica e a linguagem Prolog*. E. Blucher, janeiro 1987. 11

[12] Balsiger, Peter, Alain Heuerding e Stefan Schwendimann: *A benchmark method for the propositional modal logics K, KT, S4*. Journal of Automated Reasoning, 24:297–317, 2000. 21

[13] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest e Clifford Stein: *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edição, 2009, ISBN 0262033844. 22