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

Preprocessamento de cláusulas para raciocínio local e global no $K_S P$

João Victor C. de Melo

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

Orientadora
Prof.a Dr.a Cláudia Nalon

Brasília
2023

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

Preprocessamento de cláusulas para raciocínio local e global no KSP

João Victor C. de Melo

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

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

Prof. Dr. Flávio Leonardo Calvacanti de Moura
CIC/UnB

Prof. Dr. Bruno Lopes Vieira
IC/UFF

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

Brasília, 16 de fevereiro de 2023

Dedicatória

Dedico este trabalho às minhas duas vós (in memoriam), que me ensinaram o que realmente importa na vida.

"A vida é assim: esquenta e esfria, aperta e daí afrouxa, sossega e depois desinquieta. O que ela quer da gente é coragem." (Guimarães Rosa)

Agradecimentos

Agradeço a minha orientadora, Cláudia Nalon, por entender e simpatizar com todas as dificuldades pela qual passei até a entrega do presente trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

O foco deste trabalho foi a implementação no KSP, um provador de teoremas baseado em resolução para a linguagem multimodal K_n, do preprocessamento de cláusulas necessário para a realização da combinação do raciocínio global e local pelo provador. Especificamente, foram implementadas a entrada de cláusulas com rótulos (análises léxica e sintática) e os procedimentos de subsunção (remoção de cláusulas por redundância).

Palavras-chave: Lógica Modal, Raciocínio Automatizado, Resolução, Preprocessamento de Cláusulas

Abstract

The focus of this work is to implement the preprocessing phase for the combination of clausal global and local reasoning in KSP, a resolution based prover for the basic multimodal logic K_n. Specifically, it was implemented the input of labelled clauses (lexical analysis and syntactic analysis) and the subsumption procedures (redundancy removal of clauses).

Keywords: Modal Logic, Automated Reasoning, Resolution, Clause Preprocessing

Sumário

Lista de Tabelas

Capítulo 1

Introdução

A área de foco deste trabalho é a área de raciocínio automatizado, onde o objetivo é produzir programas que fazem raciocínios lógicos completamente ou quase completamente automático, isto é descrever um algoritmo para um cálculo formal de uma maneira que possa ser implementado no computador.

O trabalho tem como problema a implementação da combinação do raciocínio local e global no KSP, um provador de teoremas baseado em resolução para a lógica multimodal K_n. Uma vez que essa lógica tem várias aplicações, por exemplo raciocínio sobre representação de conhecimento [1], a implementação do cálculo com a combinação de raciocínio local e global no provador representa um avanço na resolução de alguns problemas de forma automática.

Especificamente, o foco do trabalho é implementar procedimentos necessários para lidar com cláusulas utilizadas na combinação de raciocínio local e global no provador, durante a fase de preprocessamento do provador.

Para se introduzir cláusulas para o raciocínio local e global, são adicionados dois novos padrões no arquivo de entrada do analisador léxico. São adicionadas também duas novas regras no arquivo de entrada do analisador sintático para cada novo caso de cláusula possíveis (iniciais, literais e modais).

A estrutura da árvore sintática abstrata foi alterada, com a introdução de um campo para armazenar os rótulos das cláusulas em cada nó da árvore. Os valores possíveis desse rótulo são determinado (local ou global) ou não-determinado.

Para o armazenamento das cláusulas globais foram utilizadas quatro novas variáveis de conjuntos: o conjunto global de suporte para literais; conjunto de usáveis para literais; conjunto de suporte modal; e o conjunto de usáveis modal. Foram implementados os procedimentos para que novas cláusulas fossem armazenadas em tais conjuntos dependendo dos seus rótulos.

Para redundância foram implementadas novas verificações dentro das funções de subsunção: *forward_subsumes*, *backward_subsumes*, *forward_subsumed* e *backward_subsumed*. Estas alterações foram necessárias para garantir que subsunção entre cláusulas globais e locais fosse corretamente executada. Também foi alterada a função para lidar com *self_subsumption* nos novos conjuntos criados. Além disso também foram adicionadas verificações de repetição de cláusulas aos conjuntos globais.

Não foi implementado nenhum dos procedimentos relativos ao preprocessamento de fórmulas, nem os procedimentos para realização de resolução unitária e eliminação de literal puro, que também fazem parte do preprocessamento. Ainda, sabe-se que há problemas com resultados obtidos, quando se usa subsunção e alguns argumentos de entrada como eliminação por literal puro a cada nível modal.

A estrutura que este trabalho segue é a seguinte: no Capítulo 2 é feita a apresentação da linguagem modal para representação de problemas, onde discute-se a sintaxe, semântica e cálculo. Em seguida, no Capítulo 3, é apresentada a estrutura do provador KSP demonstrando o algoritmo base, quais estruturas de dados são utilizadas, possíveis opções e o efeito de cada opção durante a execução do provador. No Capítulo 4 é descrito o que foi implementado e as mudanças necessárias no código. Conclusões e trabalhos futuros são apresentados no Capítulo 5.

Capítulo 2

Lógica Modal

A lógica modal em sua essência estuda métodos de raciocínio, com o propósito de classificar a verdade de um julgamento [2]. Quer isto dizer que a lógica modal é o estudo do comportamento dedutivo de expressões como *é necessariamente que*, como também *é possível que*. Logo, este termo pode ser utilizado como uma forma de representar uma família de lógicas, por exemplo:

- Lógica Deôntica
- Lógica Temporal
- Lógica Doxástica

Neste capítulo serão abordadas a sintaxe e a semântica da lógica multimodal K_n utilizada no KSP, como também a transformação para forma normal e o cálculo. Todas as definições apresentadas neste capítulo foram adaptadas de [3].

2.1 Sintaxe

A linguagem da lógica multimodal K_n, pode ser definida da seguinte forma [3]:

Definição 1 Seja $A_n = \{1, \cdots, n\}$, $n \in \mathbb{N}$ um conjunto finito de índices e seja $P = \{p, q, s, p', q', \cdots\}$ um conjunto enumerável de símbolos proposicionais. O conjunto de *fórmulas bem formadas*, $\mathsf{WFF_K}$, é dada pelo menor conjunto tal que:

- cada $p \in P$ está em $\mathsf{WFF_K}$;
- se φ e ψ estão em $\mathsf{WFF_K}$, logo também estão $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\boxed{a}\varphi$ e $\diamondsuit\!\!\!\!\!\!\diamondsuit_a\varphi$ para cada $a \in A_n$.

As fórmulas **false**, **true**, $(\varphi \rightarrow \psi)$ e $(\varphi \leftrightarrow \psi)$ são introduzidas como abreviações usuais para $(\varphi \wedge \neg\varphi)$, $\neg$**false**, $(\neg\varphi \vee \psi)$ e $((\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi))$ respectivamente, (onde $\varphi, \psi \in \mathsf{WFF_K}$).

Definição 2 Literal é um símbolo proposicional ou sua negação. O conjunto de literais é denotado por L. O *complemento* de um literal $l \in L$ é $\neg l$, ou seja se l é p então seu complemento é $\neg p$; se l é $\neg p$ então seu complemento é p. Um *literal modal* é da forma $\boxed{a}l$ ou $\diamondsuit\!\!\!\!\!\!\diamondsuit_a l$ onde $l \in L$ e $a \in A_n$.

A profundidade modal de uma fórmula corresponde ao aninhamento máximo de operadores modais, conforme apresentado a seguir.

Definição 3 Sejam $\varphi, \psi \in \mathsf{WFF_K}$, $a \in A_n$ e $\mathsf{mdepth} : \mathsf{WFF_K} \longrightarrow \mathbb{N}$. A profundidade modal de φ, $\mathsf{mdepth}(\varphi)$, é dada por:

- $\mathsf{mdepth}(p) = 0, p \in P$
- $\mathsf{mdepth}(\neg\varphi) = \mathsf{mdepth}(\varphi)$
- $\mathsf{mdepth}(\varphi \wedge \psi) = max(\mathsf{mdepth}(\varphi), \mathsf{mdepth}(\psi))$
- $\mathsf{mdepth}(\varphi \vee \psi) = max(\mathsf{mdepth}(\varphi), \mathsf{mdepth}(\psi))$
- $\mathsf{mdepth}(\boxed{a}\varphi) = 1 + \mathsf{mdepth}(\varphi)$
- $\mathsf{mdepth}(\diamondsuit\!\!\!\!\!\!\diamondsuit_a\varphi) = 1 + \mathsf{mdepth}(\varphi)$

Para a construção da árvore sintática se utiliza uma função $\tau : \mathsf{WFF_K} \times \Sigma^\star \times \{+,-\} \times \mathbb{N} \longrightarrow 2^{\mathsf{WFF_K} \times \Sigma^\star \times \{+,-\} \times \mathbb{N}}$. Os parâmetros de entrada são uma fórmula bem formada; uma sequência finita $\Sigma^\star$ sobre o alfabeto $\Sigma = \{1, 2, \cdot\}$; a polaridade (que pode ser positiva ou negativa) representada por $pol \in \{+,-\}$; e o seu nível modal representado por ml. Para a polaridade, complementação é dada pela função $\mathsf{comp} : \{+,-\} \longrightarrow \{+,-\}$ onde $\mathsf{comp}(+) = -$ e $\mathsf{comp}(-) = +$.

Definição 4 A função $\tau : \mathsf{WFF_K} \times \Sigma^\star \times \{+,-\} \times \mathbb{N} \longrightarrow 2^{\mathsf{WFF_K} \times \Sigma^\star \times \{+,-\} \times \mathbb{N}}$ é definida por:

- $\tau(p, \lambda, pol, ml) = \{(p, \lambda, pol, ml)\}$, para $p \in P$;
- $\tau(\neg\varphi, \lambda, pol, ml) = \{(\neg\varphi, \lambda, pol, ml)\} \cup \tau(\varphi, \lambda.1, \mathsf{comp}(pol)), ml)$;
- $\tau(\boxed{a}\varphi, \lambda, pol, ml) = \{(\boxed{a}\varphi, \lambda, pol, ml)\} \cup \tau(\varphi, \lambda.1, pol, ml + 1)$;

- $\tau(\diamondsuit_a\varphi, \lambda, pol, ml) = \{(\diamondsuit_a\varphi, \lambda, pol, ml)\} \cup \tau(\varphi, \lambda.1, pol, ml + 1)$;
- $\tau(\varphi \vee \varphi', \lambda, pol, ml) = \{(\varphi \vee \varphi', \lambda, pol, ml)\} \cup \tau(\varphi, \lambda.1, pol, ml) \cup \tau(\varphi', \lambda.2, pol, ml)$;
- $\tau(\varphi \wedge \varphi', \lambda, pol, ml) = \{(\varphi \wedge \varphi', \lambda, pol, ml)\} \cup \tau(\varphi, \lambda.1, pol, ml) \cup \tau(\varphi', \lambda.2, pol, ml)$.

Definição 5 Seja φ uma fórmula bem-formada, $\tau(\varphi, \epsilon, +, 0)$ sua árvore anotada e φ' uma subfórmula de φ. Se $(\varphi', \lambda, pol, ml) \in \tau(\varphi, \epsilon, +, 0)$ então o nível modal de φ' em φ é dado por $\mathsf{ml}_(\varphi, \varphi', \lambda) = ml$ e a polaridade de φ' em φ é dado por $\mathsf{pol}(\varphi, \varphi', \lambda) = pol$.

Isto é, se $\mathsf{ml}_(\varphi, \varphi', \lambda) = ml$ então φ' na posição λ ocorre com o valor de ml. Analogamente, para $\mathsf{pol}(\varphi, \varphi', \lambda) = +$ ou $\mathsf{pol}(\varphi, \varphi', \lambda) = -$ que significa que φ' na posição λ tem polaridade positiva ou negativa.

Para que φ seja *pura*, deve ocorrer com a mesma polaridade em todas as posições em uma fórmula.

2.2 Semântica

Definição 6 Um *modelo de Kripke para n agentes sobre P* é dado pela tupla:

$$(W, w_0, R_1, \cdots, R_n, \pi)$$

onde W é um conjunto não-vazio de mundos possíveis; w_0 o mundo inicial (a raiz); $R_a \subseteq W \times W$, com $a \in A_n$, uma relação binária de acessibilidade entre mundos; e $\pi : W \longrightarrow (P \longrightarrow \{true, false\})$ uma função que associa a cada mundo e a cada proposição uma interpretação.

Definição 7 A satisfatibilidade de uma fórmula em um mundo w de um modelo M é indutivamente definida por:

- $(M, w) \models p$, se somente se, $\pi(w)(p) = true$, aonde $p \in P$;
- $(M, w) \models \neg\varphi$, se somente se, $(M, w) \not\models \varphi$;
- $(M, w) \models (\varphi \wedge \psi)$, se somente se, $(M, w) \vDash \varphi$ e $(M, w) \vDash \psi$;

- $(M, w) \models (\varphi \vee \psi)$, se somente se, $(M, w) \vDash \varphi$ ou $(M, w) \vDash \psi$;
- $(M, w) \models \boxed{a}\varphi$, se somente se, para todo w', wR_aw' implica em $(M, w') \vDash \varphi$;
- $(M, w) \vDash \diamondsuit\!\!\!\!\!\!\diamondsuit_{@}\varphi$, se somente se, existe w', tal que wR_aw' e $(M, w') \vDash \varphi$;

Seja $M = (W, w_0, R_1, \cdots, R_n, \pi)$ um modelo. Uma fórmula φ é *satisfeita* em M se, e somente se, $(M, w_0) \vDash \varphi$. É *satisfatível localmente*, se existir $M, w_0 \in M$ tal que $(M, w_0) \vDash \varphi$. Denota-se por $M \vDash_L \varphi$ a satisfação local de φ em M. A satisfatibilidade global em M é denotada por $M \vDash_G \varphi$. A fórmula φ é *satisfatível globalmente*, se existe um modelo M, tal que M satisfaz globalmente φ, isto é se para todo $w \in W, (M, w) \vDash \varphi$. A definição da satisfatibilidade de uma fórmula sob restrições globais é da seguinte forma: dado um conjunto de fórmulas Γ, uma fórmula φ é *localmente satisfatível sob restrições globais em* Γ se existe um modelo M tal que, $M \vDash_G \Gamma$ e $M \vDash_L \varphi$.

A *camada modal* de um modelo é o conjunto de mundos com o mesma profundidade modal. Em [3] nota-se que o problema de verificar a satisfatibilidade local de φ, corresponde ao problema de verificar a satisfatibilidade local de uma subfórmula φ' em uma *camada modal* correspondente ao *nível modal* aonde φ' ocorre. Logo os termos *camada modal* e *nível modal* vão ser utilizados como sinônimos. Também nota-se que o problema de verificar a satisfatibilidade global de φ pode ser reduzido ao problema de verificar a satisfatibilidade local de φ em todas as camadas modais.

As seguintes definições são necessárias para a introdução da forma normal na próxima seção. Seja $K_n^\star$ uma extensão de K_n com um operador adicional $\boxed{\star}$ (um operador universal). Seja $M = (W, w_0, R_1, \cdots, R_n, \pi)$ um modelo para K_n e seja $M^\star = (W, w_0, R_1, \cdots, R_n, R_\star, \pi)$, sua extensão para $K_n^\star$, onde $R_\star = (W \times W)$.

Para fórmulas em K_n, a satisfatibilidade em um mundo é dada como na Definição 7. A fórmula $\boxed{\star}\varphi$ é localmente satisfatível em um mundo w no modelo $M^\star$, denotado por $(M^\star, w) \vDash_L \boxed{\star}\varphi$, se e somente se, para todos os $w' \in W$, temos que $(M^\star, w') \vDash \varphi$.

Dadas estas definições, obtém-se que $M \vDash_G \varphi$ equivale a $M^\star \vDash_L \boxed{\star}\varphi$ [4].

2.3 Formal normal em camadas

Forma normal anotada é utilizada como formato das fórmulas sobre as quais é aplicado o cálculo apresentado na Seção 2.4. Uma fórmula está na *Forma Normal Separada em Camadas* (SNF_{ml}), se for da forma $ml : \varphi$, onde φ é uma fórmula bem formada em K_n e $ml \in \mathbb{N} \cup \{*\}$ um rótulo, representa o nível modal em que a fórmula ocorre. Denota-se por WFF_K o conjunto de fórmulas bem-formadas rotuladas.

Dados uma fórmula bem formada φ e $M^\star = (W, w_0, R_1, \cdots, R_n, R_\star, \pi)$ um modelo em $K^\star$, a satisfatibilidade de uma fórmula $ml : \varphi$ é dada por:

- $M^\star \vDash ml : \varphi$, se e somente se, para todos os mundos $w \in W$ tal que $\mathsf{depth}(w) = ml$, temos que $(M^\star, w) \vDash \varphi$.
- $M^\star \vDash \star : \varphi$, se e somente se, $M^\star \vDash \boxed{\star}\varphi$.

Uma cláusula em SNF_{ml} está em uma das seguintes formas:

- cláusula literal: $ml : \bigvee_{b=1}^{r} l_b$
- a-cláusula positiva: $ml : l' \Rightarrow \boxed{a}l$
- a-Cláusula negativa: $ml : l' \Rightarrow \diamondsuit\!\!\!\!\!\!\diamondsuit_a l$

onde $ml \in \mathbb{N} \cup \{\star\}$ e $l, l', l_b \in L$, a-cláusulas são conhecidas também como cláusulas a-modais, onde $a \in A_n$. Como operador de conjunção é comutativo, associativo e idempotente refere-se à SNF_{ml} de uma fórmula como um *conjunto* de cláusulas.

Cláusulas são mantidas na forma simplificada, ou seja não há literais repetidos dentro da cláusula e cláusulas na forma $ml : C \vee l \vee \neg l$ são reduzidas para ml : **true**. Uma cláusula literal $ml : C$ é dita *positiva* se todos os símbolos proposicionais ocorrendo em C têm polaridade positiva (ou seja, são da forma $p \in P$); e é dita *negativa* se todos os símbolos proposicionais ocorrendo em C têm polaridade negativa (ou seja, são da forma $\neg p$ para algum $p \in P$).

Em [5], mostra-se que toda fórmula em K_n pode ser transformada na SNF_{ml}. Dada uma fórmula φ, $0 : t \wedge \rho(0 : t \Rightarrow \varphi)$ produz sua forma normal local, $\star : t \wedge \rho(\star : t \Rightarrow \varphi)$ sua forma normal global e $\star : t \wedge \rho(0 : t \Rightarrow \varphi) \wedge \rho(\star : t \Rightarrow \gamma_1, \cdots, \gamma_m)$ produz a forma normal sob restrições $\Gamma = \{\gamma_1, \cdots, \gamma_m\}, m \in \mathbb{N}$, onde a função de transformação, $\rho : \mathsf{WFF_K} \longrightarrow \mathsf{WFF_K}$, é dada por (com $\star + 1 = \star$):

- $\rho(ml : t \Rightarrow \varphi \wedge \varphi') = \rho(ml : t \Rightarrow \varphi) \wedge \rho(ml : t \Rightarrow \varphi')$
- $\rho(ml : t \Rightarrow \boxed{a}\varphi) = (ml : t \Rightarrow \boxed{a}\varphi)$ se φ é literal; $\rho(ml : t \Rightarrow \boxed{a}t') \wedge \rho(ml+1 : t' \Rightarrow \varphi)$, caso contrário;
- $\rho(ml : t \Rightarrow \diamondsuit\!\!\!\!\!\!\diamondsuit_a \varphi) = (ml : t \Rightarrow \diamondsuit\!\!\!\!\!\!\diamondsuit_a \varphi)$ se φ é literal; $\rho(ml : t \Rightarrow \diamondsuit\!\!\!\!\!\!\diamondsuit_a t') \wedge \rho(ml+1 : t' \Rightarrow \varphi)$, caso contrário;
- $\rho(ml : t \Rightarrow \varphi \vee \varphi') = (ml : \neg t \vee \varphi \vee \varphi')$ se for uma disjunção de literais; $\rho(ml : t' \Rightarrow \varphi \vee t') \wedge \rho(ml : t' \Rightarrow \varphi')$, caso contrário.

Lema 1 *Seja $\varphi \in \mathsf{WFF_K}$ uma fórmula, $\Gamma = \{\gamma_1, \cdots, \gamma_m\}, m \in \mathbb{N}$ e t um novo símbolo proposicional que não ocorre em φ.*

1. *φ é localmente satisfatível, se e somente se, $0 : t \wedge \rho(0 : t \Rightarrow \varphi)$ é satisfatível.*
2. *φ é globalmente satisfatível, se e somente se, $\star : t \wedge \rho(\star : t \Rightarrow \varphi)$ é satisfatível.*
3. *φ é localmente satisfatível sob restrições globais Γ se, e somente se, $\star : t \wedge \rho(0 : t \Rightarrow \varphi) \wedge \rho(\star : t \Rightarrow \gamma_1, \cdots, \gamma_m)$ é satisfatível.*

A prova deste lema pode ser encontrada em [5].

Algumas transformações extras são necessárias no conjunto de cláusulas para garantir, por exemplo, a completude de resolução negativa que requer que os literais no escopo dos operadores modais sejam todos positivos. Dado um conjunto de cláusulas em SNF_{ml}, às cláusulas modais é aplicada a seguinte função:

- $\rho(ml : t \Rightarrow \boxed{a}\neg p) = (ml : t \Rightarrow \boxed{a}t') \wedge \rho(ml + 1 : t' \Rightarrow \neg p)$
- $\rho(ml : t \Rightarrow \diamondsuit_{a}\neg p) = (ml : t \Rightarrow \diamondsuit_{a}t') \wedge \rho(ml + 1 : t' \Rightarrow \neg p)$

onde $ml \in \mathbb{N} \cup \{\star\}$ $t, p \in P$ e t' é um novo símbolo proposicional. A forma normal resultante é denotada por SNF^{+}_{ml}.

Para completude de resolução ordenada também é preciso que os literais no escopo dos operadores modais sejam *pequenos o suficiente* em acordo com uma ordem específica sobre os literais que ocorrem no conjunto de cláusulas. Pode se garantir estas condições fazendo mais processamentos no conjunto de cláusulas SNF_{ml}. Seja ϕ um conjunto de cláusulas e P_ϕ um conjunto de símbolos proposicionais ocorrendo em ϕ. Seja $\prec$ uma ordem total em P_ϕ, onde essa ordem podem ser estendida para os literais L_ϕ em P_ϕ colocando $\neg p \prec p$ e $p \prec \neg q$ sempre que $p \prec q$ para todo $p, q \in P_\phi$. Um literal l é dito *maximal* em uma cláusula $ml : C \vee l$, se não há nenhum $l' \in C$ tal que $l' \prec l$. Dado um conjunto de cláusulas Φ em SNF_{ml} e uma ordem $\prec$ dos literais ocorrendo em ϕ, aplica-se a seguinte função às cláusulas modais:

- $\rho(ml : t \Rightarrow \boxed{a}l) = (ml : t \Rightarrow \boxed{a}t') \wedge \rho(ml + 1 : t' \Rightarrow l)$
- $\rho(ml : t \Rightarrow \diamondsuit_{a}l) = (ml : t \Rightarrow \diamondsuit_{a}t') \wedge \rho(ml + 1 : t' \Rightarrow l)$

onde $ml \in \mathbb{N} \cup \{\star\}$, $t \in P$, $l \in L$, t' é um novo símbolo proposicional e $p \prec t'$ para todo p ocorrendo em ϕ.

Em [5] mostra-se que o resultado produzido, denotado por SNF^{++}_{ml}, preserva satisfatibilidade, isto é, que Φ é satisfatível, se e somente se, o conjunto de cláusulas em SNF^{++}_{ml} produzido a partir de Φ é satisfatível.

2.4 Cálculo

Esta seção apresenta o cálculo baseado em resolução para lógica modal aqui considerada. No que se segue a função parcial de unificação dos rótulos das cláusulas é dada por $\sigma : 2^{\mathbb{N}\cup\{\star\}} \longrightarrow \mathbb{N} \cup \{\star\}$, aonde $\sigma(\{ml, \star\}) = ml$ e $\sigma(\{ml\}) = ml$; caso contrário, σ não é definido. Define-se a subtração com respeito a rótulos como $\star - 1 = \star$.

Primeiramente serão apresentadas as seguintes definições do cálculo. As regras do cálculo LRES , MRES , GEN2 , GEN1 e GEN3 são apresentadas na Tabela 2.1.

Definição 8 Seja Φ um conjunto de cláusulas em SNF_{ml}. Uma *derivação de* Φ é uma sequência de conjuntos $\Phi_0, \Phi_1, \cdots$ onde $\Phi_0 = \Phi$ e para cada $i > 0$, $\Phi_{i+1} = \Phi_i \cup \{D\}$, aonde $D \notin \Phi_i$ é o resolvente obtido de Φ_i pela aplicação das regras LRES, MRES, GEN2, GEN1 ou GEN3, D esteja na forma simplificada e não seja uma tautologia.

Definição 9 Um conjunto de cláusulas em SNF_{ml} Φ é *saturado*, se todo resolvente obtido de Φ a partir da aplicação de uma das regras de inferência, LRES, MRES, GEN2, GEN1 ou GEN3 já estão em Φ ou é uma tautologia.

Definição 10 Seja Φ um conjunto de cláusulas em SNF_{ml}. Uma *refutação local* para Φ é uma derivação $\Phi_0, \cdots, \Phi_k, k \in \mathbb{N}$, onde $0 : \mathbf{false} \in \Phi_k$. Uma *refutação global* de Φ é uma derivação $\Phi_0, \cdots, \Phi_k, k \in \mathbb{N}$, onde $\star : \mathbf{false} \in \Phi_k$.

No caso de satisfatibilidade local com condições globais, a refutação pode ser tanto global quanto local.

Definição 11 Uma derivação para Φ é dita *terminante* se é uma refutação global ou local para Φ ou se $\Phi_i, i \in \mathbb{N}$, tal que Φ_i é saturado.

Os próximos teoremas garantem que o cálculo seja correto, completo e terminante.

Teorema 1 (Correção) *Seja Φ um conjunto de cláusulas em SNF_{ml} e $\Phi_0, \cdots, \Phi_k$ $k \in \mathbb{N}$ uma derivação para Φ. Se Φ é satisfatível, então todo $\Phi_i, 0 \leq i \leq k$, é satisfatível.*

A prova em [5] mostra que cada regra de inferência preserva satisfatibilidade, isto é, se as premissas são satisfatíveis então o resolvente também é.

Teorema 2 (Completude) *Seja Φ um conjunto insatisfatível de cláusulas em SNF_{ml}. Então existe uma refutação para Φ pela aplicação das regras dadas na Tabela 2.1.*

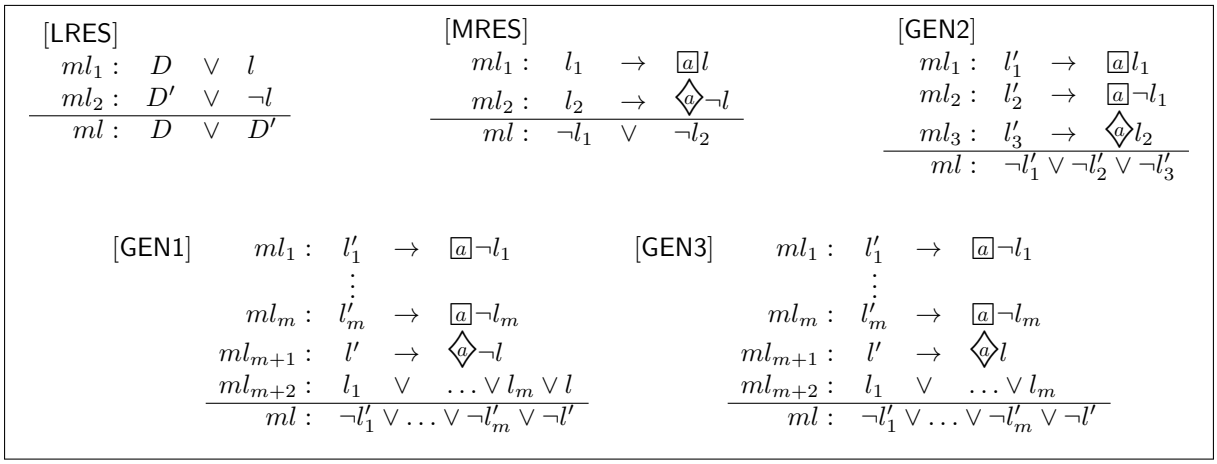

Tabela 2.1: Regras de inferência, aonde $ml = \sigma(\{ml_1, \ldots, ml_{m+1}, ml_{m+2} - 1\})$ em GEN1, GEN3, onde $m \geq 0$; $ml = \sigma(\{ml_1, ml_2\})$ em LRES, MRES; e $ml = \sigma(\{ml_1, ml_2, ml_3\})$ em GEN2.

A prova de completude do cálculo, conforme [5], mostra que se Φ é um conjunto insatisfatível de cláusulas em SNF_{ml}, então uma refutação é produzida pelo cálculo. Essa prova é baseada na construção de um grafo, chamado de *grafo comportamental* de Φ, de tal forma que cada vértice representa um mundo e suas arestas relações de acessibilidade. A remoção de vértices no grafo é equivalente a aplicação das regras na Tabela 2.1. Mostra-se em [5] que se o grafo obtido for vazio então o conjunto de cláusulas é insatisfatível e exite uma refutação para Φ.

Teorema 3 (Terminação) *Seja Φ um conjunto de cláusulas em SNF_{ml}, então qualquer derivação para Φ termina.*

Esse resultado, conforme [5], vem da percepção que nenhuma das regras apresentadas na Tabela 2.1 gera novos literais; novos literais modais; ou novos rótulos. Logo, existe um número finito de cláusulas que podem ser criadas a partir de Φ.

Capítulo 3

KSP

KSP é um provador de teoremas para a lógica multimodal K_n que implementa o cálculo visto no capítulo anterior. Neste capítulo iremos apresentar o algoritmo implementado e as principais opções para o provador, com ênfase na parte de preprocessamento, objeto deste trabalho.

O procedimento para busca de uma prova é apresentada no Algoritmo 1. O algoritmo apresenta busca por provas para satisfatibilidade local ou global e pode ser dividido em duas grandes seções: as etapas de preprocessamento e o laço principal. O preprocessamento, que é o foco da implementação deste trabalho, é apresentado nas Linhas 1 a 3. O laço principal, Linhas 4 a 16, baseia-se no algoritmo de cláusulas dadas, implementado em *Otter* [6], que restringe o conjunto de escolhas de cláusulas participando em uma etapa de derivação, sendo variação da estratégia de conjunto de suporte.

Seja Δ um conjunto de cláusulas. No algoritmo original para a lógica clássica, temos que Δ é dividido em dois conjuntos Γ e $\Lambda = \Delta \setminus \Gamma$. O conjunto Γ é o conjunto de *suporte* ou normalmente chamado pelo seu acrônimo em inglês *sos* (*Set Of Support*). O conjunto Λ é chamado de *ativo* (*usable*, em inglês). A restrição imposta pela estratégia é a de que uma cláusula em Γ pode tão somente ser resolvida com cláusulas em Λ. Durante o processamento, à medida que são processadas, as cláusula em Γ são movidas para Λ. Os resolventes gerados durante o processamento são colocados em Γ.

Para o cálculo modal há uma mudança na partição do conjunto de cláusulas: um novo conjunto ativo Λ_{ml}^{mod} para as cláusulas modais, em cada nível ml; um conjunto Γ_{ml}^{lit} de suporte e um conjunto ativo Λ_{ml}^{lit}, em cada nível ml, para as cláusulas literais. Observa-se que o cálculo não gera novas cláusulas modais mas também porque o conjunto de cláusulas modais por si mesmo é satisfatível, portanto não é necessário um conjunto de suporte para cláusulas modais.

Resumidamente, a tentativa de aplicar uma regra de inferência em cada nível modal ml é feita pela escolha de uma cláusula no conjunto de suporte de cláusulas literais.

A cláusula escolhida pode ser resolvida com cláusulas ou do conjunto ativo de cláusula literais desta camada (Λ^{lit}_{ml}) ou do conjunto ativo de cláusulas modais na camada anterior (Λ^{mod}_{ml-1}).

Detalhadamente, como nenhuma cláusula modal é gerada pelo cálculo apresentado na Seção 2.4, a implementação da aplicação das regras MRES e GEN2 é realizado antes do laço principal, durante o preprocessamento de cláusulas (Linha 3). Um ciclo definido pelo laço mais externo (Linha 5) vai ser executado enquanto a união de todos os conjuntos de suporte não for vazio. O laço interno é executado para todos os níveis modais (Linha 6). Para cada nível modal, é então obtida uma cláusula (função given, Linha 7), que em seguida é testada para redundância (Linha 8). Se for redundante, quer isto dizer que não afeta se deletada do conjunto de cláusulas; então a cláusula é removida do conjunto de suporte deste nível modal (Linha 14). Se não for redundante, resolução modal (GEN1 E GEN3) é aplicada entre a cláusula escolhida e todas as cláusulas modais do nível anterior (Linha 9 e 10). Resolução literal (LRES) é aplicada entre a cláusula escolhida e todas as cláusulas literais do mesmo nível modal (Linha 11). Após ser processada, a cláusula escolhida é colocada no conjunto ativo de mesmo nível modal (Linha 12).

Se a cláusula vazia for gerada (condição na Linha 15), o algoritmo retorna que o conjunto é insatisfatível (Linha 16); caso contrário retorna que é satisfatível (Linha 20).

Algorithm 1 KsP-Proof-Search

```
 1: input_processing;
 2: snf_transformation;
 3: clause_preprocessing;
 4: Γ^lit ← ∪Γ^lit_ml;
 5: while Γ^lit ≠ ∅ do
 6:    for (all modal levels ml) do
 7:       clause ← given(ml);
 8:       if not_reduntant(clause) then
 9:          GEN1(clause, ml, ml − 1);
10:          GEN3(clause, ml, ml − 1);
11:          LRES(clause, ml, ml);
12:          Λ^lit_ml ← Λ^lit_ml ∪ {clause};
13:       end if
14:       Γ^lit_ml ← Γ^lit_ml \ {clause};
15:       if 0 : false ∈ Γ^lit_0 ∨ ⋆ : false ∈ Γ^lit then
16:          return unsatisfiable;
17:       end if
18:    end for
19: end while
20: return satisfiable;
```

$(\varphi \vee \varphi) \to \varphi$	$(\varphi \wedge \varphi) \to \varphi$	$(\varphi \vee \neg\varphi) \to \mathbf{true}$
$(\varphi \wedge \neg\varphi) \to \mathbf{false}$	$(\varphi \vee \mathbf{true}) \to \mathbf{true}$	$(\varphi \wedge \mathbf{false}) \to \mathbf{false}$
$(\varphi \vee \mathbf{false}) \to \varphi$	$(\varphi \wedge \mathbf{true}) \to \varphi$	$\neg\mathbf{false} \to \mathbf{true}$
$\neg\mathbf{true} \to \mathbf{false}$		
$[a]\mathbf{true} \to \mathbf{true}$	$\langle a\rangle\mathbf{false} \to \mathbf{false}$	$([a]\varphi \wedge \langle a\rangle\neg\varphi) \to \mathbf{false}$
$([a]\varphi \wedge \neg[a]\varphi) \to [a]\mathbf{false}$	$([a]\mathbf{false} \wedge \langle a\rangle\varphi) \to \mathbf{false}$	$([a]\mathbf{false} \wedge [a]\varphi) \to [a]\mathbf{false}$
$(\langle a\rangle\varphi \vee \langle a\rangle\neg\varphi) \to \langle a\rangle\mathbf{true}$	$(\langle a\rangle\mathbf{true} \vee [a]\varphi) \to \mathbf{true}$	$(\langle a\rangle\mathbf{true} \vee \langle a\rangle\varphi) \to \langle a\rangle\mathbf{true}$

Tabela 3.1: Simplificações aplicadas a fórmulas em NNF.

3.1 Processamento de entrada

No processamento de entrada é possível a introdução de configurações na linha de comando ou por um arquivo de configuração. Caso os dois tipos de entrada de configuração sejam fornecidos, a entrada por linha de comando sempre sobrepõe a entrada do arquivo. O arquivo de entrada pode conter conjuntos de fórmulas modais e/ou conjunto de cláusulas. Tanto cláusulas quanto fórmulas podem ser definidas no arquivo de entrada como *usable* ou *sos*.

3.1.1 Análise léxica e sintática

O processamento de entrada é dividido em duas partes, o léxico (*tokeniser*) e o sintático (*parser*). O analisador léxico é obtido a partir de um gerador genérico *Flex* [7] e o analisador sintático é gerado por outro gerador genérico *Bison* [8]. A gramática da linguagem de entrada do KSP é LR (*left-to-right scanning, rightmost derivation*) que pode ser tratada pelo *Bison*. A saída da análise sintática é uma árvore abstrata anotada duplamente ligada e uma tabela de símbolos.

3.1.2 Simplificação

Cada nó na árvore é anotado com um número que pode ser não-único, porém a repetição de fórmulas é só verificada quando se dá o mesmo número a nós diferentes, portanto evitando assim possíveis percorrimentos desnecessários da árvore enquanto se faz a simplificação.

Conjunções são tratadas como operadores n-ários, com eliminação de conjunções aninhadas (*flattening*). Por exemplo, $((p \wedge q) \wedge (r \wedge (s \wedge t)))$ corresponde à conjunção $p \wedge q \wedge r \wedge s \wedge t$. Além disso as subfórmulas de uma conjunção são ordenadas para facilitar simplificação. A ordem leva em consideração a complexidade das subfórmulas: constantes lógicas são menores do que símbolos proposicionais; estas são menores do que

fórmulas clássicas as quais, por sua vez, são menores do que fórmulas modais. Disjunções têm tratamento análogo.

As tabelas de símbolos contém as informações necessárias para o cálculo. Existem duas tabelas. Uma tabela de símbolos armazena informações sobre símbolos proposicionais e constantes e outra armazena informações sobre operadores modais. Nestas tabelas são guardadas, por exemplo, o tipo do símbolo (símbolo proposicional, constante, agente), um identificador (a representação interna de um símbolo proposicional, constante, agente, etc), o número de ocorrências positivas e o número de ocorrências negativas de símbolos proposicionais, sejam elas globais ou por nível modal, a posição de constates na árvore abstrata, etc.

3.1.3 Configurações de entrada

A árvore sintática, devido à natureza do analisador sintático (ascendente), é construída das folhas para raiz. Como o cálculo da polaridade precisa ser feito a partir da raiz, pelo menos uma outra passagem pela árvore sintática é necessária.

O usuário pode escolher o tipo de simplificação nas configurações como *nnfsimp* que fará uma simplificação em NNF (*Negation Normal Form*) ou poderá escolher a simplificação *bnfsimp* que é do tipo BNF (*Box Normal Form*). A diferença entre NNF e BNF é que a BNF também remove o operador modal $\diamondsuit$. Por exemplo, para a fórmula $\neg\diamondsuit\Box\neg\varphi$, a NNF é da seguinte forma $\Box\diamondsuit\varphi$, enquanto que sua BNF é $\Box\neg\Box\neg\varphi$.

Outros parâmetros utilizados também no preprocessamento são as opções *aprenex* e *prenex*, que correspondem às funções de transformação em que operadores modais são movidos, quando possível, para dentro ou fora da fórmula, respectivamente. Por exemplo com a opção *aprenex* ativa, a fórmula $\Box(p \wedge q)$ é transformada em $(\Box p \wedge \Box q)$, enquanto que no caso *prenex* da fórmula $\diamondsuit p \vee \diamondsuit q$ obtém-se $\diamondsuit(p \vee q)$.

Além disso pode-se fazer simplificações com as opções *early_ple* e *early_mlple*, onde *early_ple* corresponde à eliminação de literais que são puros globalmente e *early_mlple* corresponde à eliminação de literais que são puros por nível modal. Outras regras de simplificações de fórmulas estão definidas na Tabela 3.1.

3.2 Transformação para forma normal

Por padrão as fórmulas são transformadas para a SNF_{ml}, apresentada na Seção 2.3. As formas estendidas, também apresentadas naquela seção, também podem ser aplicadas pelo provador. Resumidamente são quatro opções para transformação normal SNF^{+}_{ml}, SNF^{++}_{ml}, SNF^{-}_{ml} e SNF^{--}_{ml}. A transformação em todas essas formas precisam utilizar re-

$$\frac{ml: \quad l' \Rightarrow \Diamond_a l}{ml+1: \quad l}$$

Tabela 3.2: Regra de inferência para propagação de literal unitário, opção *propdia*.

nomeamento, o que é feito de forma ascendente com ajuda de um dicionário. As opções para renomeamento são as seguintes:

- *normal_renaming*
- *limited_reuse_renaming*
- *extensive_reuse_renaming*

Com *normal_renaming* produz-se um novo símbolo proposicional para cada subfórmula; com *limited_reuse_renaming* o mesmo novo símbolo proposicional vai ser usado para ocorrências da mesma subfórmula em um nível modal; e com *extensive_reuse_renaming* também se utiliza o mesmo novo símbolo proposicional para ocorrências da mesma subfórmula e, além disso, se uma fórmula φ for renomeada para t, então $\neg\varphi$ será renomeada para $\neg t$. Há também uma opção extra para tratamento de conjunções com o parâmetro *conjunct_renaming*, onde subfórmulas modais que ocorrem em conjunções são renomeadas ao invés de aplicar a regra de reescrita usual.

O conjunto de cláusulas é guardado em uma *trie*, implementada como uma tabela de dispersão em múltiplos níveis, onde estes níveis correspondem ao seu nível modal, índice do operador modal (no caso de cláusulas modais), seu literal máximo e seu tamanho. A representação de cláusulas é feita por uma lista de inteiros, correspondente aos literais ordenados.

Também para cada símbolo proposicional, uma lista de cláusulas por nível modal é colocada na tabela de símbolos. O armazenamento redundante de informação ajuda a melhorar a busca por cláusulas durante a aplicação da resolução de unidade (*unit* ou *lhs_unit*), assim como ajuda na eliminação de literal puro nas opções (*ple* e *ml_ple*).

3.3 Preprocessamento de cláusulas

Por padrão se faz uma verificação se a cláusula é repetida, mas somente no caso de cláusulas no mesmo conjunto. Por exemplo, se uma cláusula é colocada em Γ^{lit}_{ml} ela só vai ser testada para repetição nesse conjunto. Contudo com a opção *check_full_repeated* a repetição é checada contra todos os conjuntos de cláusulas do mesmo nível modal.

Propagação de literal é aplicada nesta fase com a utilização do parâmetro *propdia*, cuja regra de inferência correspondente é mostrada na Tabela 3.2, com a condição de que a regra somente seja aplicada se possui o Λ_{ml}^{mod} contendo apenas uma cláusula modal negativa. Outra opção para simplificação é *mle*, que elimina todos os níveis modais superiores a *ml*, se no nível *ml* não contiver nenhuma cláusula modal negativa. Logo, os demais níveis não são mais acessíveis, eles podem ser deletados. Formalmente, a justificativa para aplicação desta regra resulta do fato de que se temos um mundo w que satisfaça $\boxed{a}l$ e se não tiver w', tal que $(w, w') \in R_a$, podemos usar o caso que a relação de acessibilidade é vazia para todos os mundos em *ml*, ou seja satisfaz todas suas cláusulas positivas.

Como nenhuma cláusula modal é gerada pelo cálculo apresentado na Seção 2.4, como já mencionado, a implementação da aplicação das regras de inferência MRES e GEN2 é realizada antes do laço principal (Linha 3 do Algoritmo 1). Note que a transformação para SNF_{ml}^{+}, SNF_{ml}^{++}, SNF_{ml}^{-} e SNF_{ml}^{--} são feitas depois do pré processamento das cláusulas. Se uma destas opções relacionadas à transformação para forma normal for escolhida pelo usuário então todos os literais no escopo dos operadores modais terão a mesma polaridade, logo as regras MRES e GEN2 não podem ser aplicadas. Contudo as regras de inferência MRES e GEN2 geram resolventes de tamanho pequeno, o que pode ser muito útil para reduzir o número de cláusulas se subsunção estiver ativa. Assim sendo, o usuário pode manter a aplicação destas regras configurando os parâmetros *mres* e *gen2*, embora a aplicação destas regras não sejam necessárias para completude.

Se a subsunção para frente ou para trás é aplicada, respectivamente representado por *fsub* e *bsub*, então auto-subsunção também é aplicada nesse ponto. Observa-se que *fsub* e *bsub* é aplicada em modos preguiçoso e apenas levando em consideração o conjunto ativo de mesmo nível modal, enquanto que auto-subsunção é aplicado contra todos os conjuntos.

Por padrão, o conjunto ativo Λ_{ml}^{lit} (*usable*) inicia vazio. Todavia o usuário pode configurar o provador para automaticamente preencher o conjunto *usable* com cláusulas literais. Para isso existem seis opções, sendo elas: *populate_non_negative* que move cláusulas literais não-negativas de Γ_{ml}^{lit} para Λ_{ml}^{lit}; *populate_non_positive* que move cláusulas literais não-positivas de Γ_{ml}^{lit} para Λ_{ml}^{lit}; *populate_negative* move cláusulas literais negativas de Γ_{ml}^{lit} para Λ_{ml}^{lit}; *populate_positive* move cláusulas literais positivas de Γ_{ml}^{lit} para Λ_{ml}^{lit}; *populate_max_lit_negative* move cláusulas literais onde seu literal maximal é negativo de Γ_{ml}^{lit} para Λ_{ml}^{lit}; *populate_max_lit_positive* move cláusulas literais onde seu literal maximal é positivo de Γ_{ml}^{lit} para Λ_{ml}^{lit}.

3.4 Seleção de cláusula

Além da estratégia de conjunto de suporte, apresentada na Seção 3.1, que limita a escolha da cláusula candidata para resolução apenas para o conjunto *sos*, existem cinco heurísticas que afetam a escolha das cláusulas literais:

- *shortest*: escolhe-se a menor cláusula em tamanho de um dado nível modal;
- *newest*: a seleção de cláusulas é utilizando uma pilha, sendo escolhida a cláusula mais nova;
- *oldest*: a seleção de cláusulas é utilizando uma fila, sendo escolhida a cláusula mais antiga;
- *smallest*: escolhe-se a menor cláusula com o menor literal maximal; e
- *greatest*: escolhe-se a menor cláusula com o maior literal maximal.

3.5 Refinamentos

O usuário também pode restringir a regra LRES escolhendo umas das seguintes opções:

- *ordered*: cláusulas só podem ser resolvidas em seu literais maximais com uma ordem escolhida pelo $\mathrm{K_SP}$ que preserva a completude;
- *negative*: a cláusula literal das premissas é negativa;
- *positive*: a cláusula literal das premissas é positiva;
- *negord*: onde tanto as restrições de *negative* e *ordered* são aplicadas.

3.6 Regras de inferência

Além das regras de inferência já apresentadas na Tabela 2.1, o usuário pode optar por aplicar resolução unitária, com a opção *unit*, que são baseadas nas três regras de inferência, dadas na Tabela 3.3: UNIT , UNIT-GEN1 , UNIT-GEN3 . Pode se notar também que estás novas três regras tratam casos especiais de LRES, GEN1 e GEN3, mas a implementação com o uso da opção utiliza procedimentos específicos, especializados e otimizados para o tratamento destes casos. Além disso, as premissas de UNIT com $ml_1 : l'_1 \vee \cdots \vee l_m \vee l$ e UNIT-GEN1 com $ml_1 : l_1 \rightarrow \diamondsuit l$ são deletadas imediatamente com a aplicação da regra, ou seja, aplicando subsunção antes mesmo da etapa de eliminação por redundância.

A opção *lhs-unit* propaga cláusulas unitárias no lado esquerdo das cláusulas modais, correspondendo às regras de inferência dadas na Tabela 3.4: LHS-UNIT-1 e LHS-UNIT-2

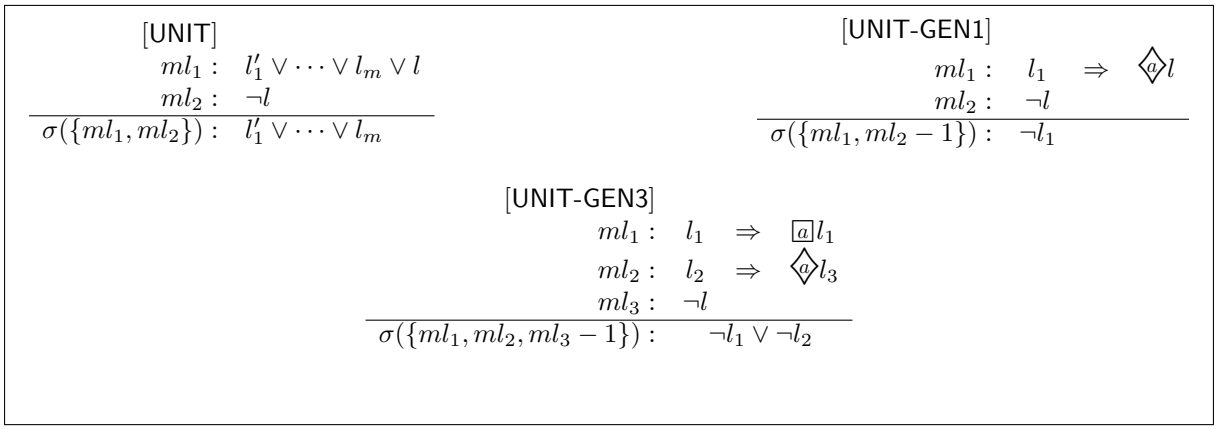

Tabela 3.3: Regras de inferência para resolução unitária, opção *unit*.

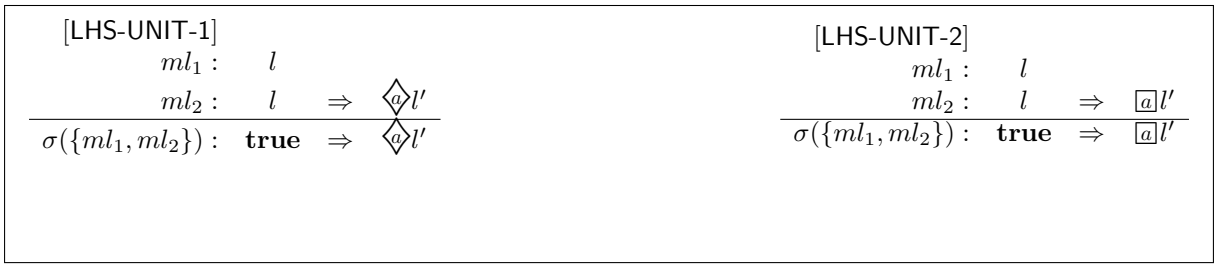

Tabela 3.4: Regras de inferência para resolução unitária no antecedente de cláusulas modais, opção *lhs-unit*.

. Embora estas regras não sejam necessárias para a completude, sua aplicação exaustiva juntamente com resolução unitária resulta na remoção de todas as ocorrências do literal $\neg l$ no nível modal aplicado. Além disso, caso um dos parâmetros *ple* ou *mlple* também tenham sido ativados, possivelmente mais cláusulas podem ser removidas dos conjuntos por eliminação de literal puro.

3.7 Eliminação da redundância

Existem opções para eliminação de literais puros aplicada globalmente (*ple*) e a eliminação de literais puros por nível modal (*mlple*). Se o literal l é puro, ele pode ser substituído por **true**. Porque isto gera tautologias, qualquer cláusula literal em que ocorra l pode ser deletada. Por exemplo, se ocorre l no escopo de $\Box_a$ de uma cláusula modal positiva, então será substituído por $\Box_a$ **true** que é uma tautologia. Se l ocorre em $\Diamond_a$ de uma cláusula modal positiva, tal que $ml - 1 : l' \rightarrow \Diamond_a l$ então está cláusula é removida e a nova cláusula $ml - 1 : l' \rightarrow \Diamond_a$ **true** é gerada. Porque $\Diamond_a$ **true** não é tautologia a cláusula é mantida.

Uma cláusula literal $ml : C$ é subsumida por uma cláusula $ml' : D$ se, e somente se, $ml' : D$ implica em $ml : C$. Tanto a subsunção para frente (*fsub*), quanto para trás (*bsub*) são implementadas e aplicadas em modo preguiçoso, entre a cláusula selecionada de Γ^{lit}_{ml} e verificando a condição de sua subsunção com as cláusulas de Λ^{lit}_{ml}. Com *fsub*, uma cláusula só é removida se for subsumida por qualquer cláusula literal mais antiga. Com *bsub*, uma cláusula só é removida se for subsumida por qualquer cláusula literal mais nova.

Dadas duas cláusulas $ml : C$ e $ml' : D$, para melhorar a seleção de candidatos para subsunção são checadas as seguintes condições:

- $\sigma(\{ml, ml'\})$ é definido;
- O tamanho de $ml : C$ é maior ou igual a $ml' : D$;
- O literal maximal de $ml : C$ é menor ou igual ao literal maximal de $ml' : D$;
- O literal minimal de $ml : C$ é maior ou igual ao literal minimal de $ml' : D$.

Considerando que as três primeiras condições são chaves da estrutura de dados que implementa os conjuntos de cláusulas (*trie*), a obtenção do conjunto de candidatos para subsunção é realizada em tempo constante. A última condição consiste em testar todas as cláusulas que satisfaçam as três primeiras condições, o que é linear para o conjunto de cláusulas.

3.8 Controle do laço interno

Para o laço interno apresentado no Algoritmo 1, Linhas 6 a 18, o conjunto de cláusulas literais Γ^{lit}_{ml} é percorrido. Como o nível modal é uma das chaves da *trie*, estrutura de dados que implementa conjuntos de cláusulas, esta é percorrida na ordem em que seus itens ocorrem na estrutura. Todavia a ordem de percorrimento pode ser alterada com uma das seguintes opções:

- *ordlevel_ascend*: itera do nível modal 0 até nível modal maximal.
- *ordlevel_descend*: itera do nível modal maximal até o nível modal 0.
- *ordlevel_shuffle*: a lista de níveis modais é particionado em dois e as duas listas resultantes são intercaladas antes de entrar no laço.

Capítulo 4

Implementação

Neste capítulo vai ser apresentado detalhadamente como foi feito o preprocessamento de cláusulas rotuladas utilizadas como entrada no KSP. Antes de tudo, já havia a implementação do cálculo para cláusulas globais e locais, separadamente. Desse modo vão ser apresentadas as novas definições para os valores dos rótulos das cláusulas; novos padrões adicionados no arquivo de entrada do analisador léxico; novas regras gramaticais adicionadas ao arquivo de entrada do analisador sintático, com intuito de abordar os casos de cláusulas locais e globais como entrada; inserção de um novo atributo rótulo para cada nó na árvore sintática; inserção das novas cláusulas de entrada em dois conjuntos globais já presentes anteriormente; subsunção entre cláusulas globais e locais.

4.1 Novos valores: global e não-determinado

Antes de tudo foram adicionadas duas novas definições no arquivo *prover.c*, na forma de constantes, com o objetivo de representar rótulos com os seguintes valores:

- **GLOBAL** equivalente com -1
- **UNDERTERMINED** equivalente com -2

Os valores foram escolhidos para que não entrem em conflito com os valores positivos das cláusulas locais.

4.2 Léxico

Ao arquivo utilizado como entrada para geração do analisador léxico, foram adicionados mais dois padrões para representação de níveis modais: **TSTAR** representado por ***** e **TCOLON** retratado por **:**.

```
clause:
        |  inital_clause
        |  literal_clause
        |  modal_clause
        |  TNUMBER    TCOLON   initial_clause
        |  TSTAR      TCOLON   initial_clause
        |  TNUMBER    TCOLON   literal_clause
        |  TSTAR      TCOLON   literal_clause
        |  TNUMBER    TCOLON   modal_clause
        |  TSTAR      TCOLON   modal_clause
```

Tabela 4.1: Introdução das novas regras em *clauses.c*.

4.3 Sintático

Foi alterado o arquivo de entrada para geração do analisador sintático, como apresentado na Tabela 4.1, onde seis novas regras foram adicionadas. As novas regras têm o intuito de conseguir representar cláusulas iguais ao formato apresentado na Seção 2.3. A definição do início da regra é representado por **TSTAR TCOLON** ou **TNUMBER TCOLON**. Destaca-se que **TNUMBER** já era presente no arquivo utilizado como entrada para o gerador do analisador léxico, com o objetivo de criar padrões de números. Porém, nestas regras é utilizado para representar cláusulas locais.

Nota-se que essa nova implementação preserva a funcionalidade para as entradas sem rótulos e fórmulas já existentes no provador.

Observa-se que a tentativa de fatoração das regras gerou ambiguidades na gramática. Para preservar funcionalidades existentes, a única solução encontrada foi a aqui apresentada.

4.4 Árvore sintática

4.4.1 Introdução de rótulos aos nós da árvore

Um novo atributo para rótulo foi introduzido à estrutura *tnode* que implementa os nós na árvore na sintática. Durante a criação da árvore sintática, por padrão, todo novo nó tem seu rótulo inicializado com o valor não-determinado, definido na Seção 4.1.

Quando cláusulas rotuladas (Seção 4.3) são encontradas, este valor passa a ser definido. Dessa forma o rótulo pode assumir o valor associado ao padrão **TNUMBER**, como pode receber o valor da definição **GLOBAL**,

4.4.2 Atualização dos nós

Foi criada uma nova função update_label(tnode * root), que espera a raiz de uma árvore como parâmetro. A finalidade desta função é propagar o rótulo dos nós pais para os nós filhos sempre que seja necessário, ou seja, caso o nó filho tenha um rótulo não-determinado e o nó pai possua um rótulo determinado, então o nó filho recebe o rótulo do nó pai. Isto foi necessário porque cláusulas eram inseridas dentro da árvore sintática com rótulos, porém seus literais não herdavam os rótulos das cláusulas que pertenciam.

Por último, a função update_label(tnode * root) foi adicionada à função print_out dentro do arquivo *output.c*, com o propósito de atualizar a árvore caso qualquer nova fase dentro do *prover* seja chamada. Todavia seguramente seria mais otimizado caso fosse movido para apenas a última fase que a árvore é mudada, entretanto devido a questões de tempo essa otimização não foi explorada nesse trabalho.

4.5 Cláusulas

4.5.1 Inserção no conjunto de cláusulas

Dentro da função insert_clause_set já implementada no arquivo *clauses.c*, foi feita modificação para que cláusulas literais possam ser inseridas nos conjuntos de suporte ou ativo, sejam elas locais ou globais, dependendo do seus rótulos. As mesmas modificações foram feitas para que cláusulas modais possam ser inseridas no conjuntos de suporte ou ativo, sejam elas locais ou globais, também dependendo de seus rótulos. Em conclusão uma cláusula literal pode ser inserida no conjunto de suporte global, no conjunto de suporte local, no conjunto de usáveis global ou no conjunto de usáveis locais dependendo do seu rótulo. O mesmo se aplica a cláusulas modais.

4.6 Tabela de símbolos

No momento em que entra na fase de transformação na forma normal em camada, (fase SNF na implementação), insere-se novas informações sobre as cláusulas na tabela de símbolos. Esta funcionalidade é dada pelas funções insert_clause_list e insert_clause_p_node que já estavam implementadas. Para tratar os novos formatos de cláusulas, estas funções foram modificadas de modo que é apenas verificado se a cláusula possui um rótulo determinado. Por exemplo, caso uma cláusula possua um rótulo não-determinado então se passa o seu nível modal como argumento quando inserido na tabela de símbolos, no entanto se seu rótulo for determinado, se passa seu rótulo como argumento.

4.7 Redundância

4.7.1 Repetição

Foram adicionados mais casos de verificação de repetição aos conjuntos *g_usable* e *g_sos*, caso a opção de entrada *full_check_repeated* esteja ativa, já que a função *check_repeated_prop* em *clauses.c* somente faz verificações de repetição entre cláusulas do mesmo conjunto.

4.7.2 Subsunção

Primeiramente, teve-se que alterar a função base *subsumes* em *subsumption.c*, com o intuito de conter o caso de que uma cláusula global subsume uma cláusula local. Em seguida são tratados os dois casos de *forward_subsumption* e os dois casos de *backward_subsumption*. Foi também realizada uma alteração na função *self_subsumption*.

Para cada tipo de subsunção há duas funções já implementadas. Em *forward_subsumption*, temos *forward_subsumes* com objetivo de apenas deletar a cláusula, escrever sua justificativa como ainda verificar se a cláusula, dada como argumento, é subsumida com outra cláusula no conjunto ; e *forwad_subsumed* que também deleta a cláusula, dá uma justificativa porém verifica se a cláusula, dada como argumento, subsume uma cláusula no conjunto, ademais ele retorna se a cláusula foi subsumida ou não durante sua execução. Para *backward_subsumption*, existem função equivalentes: *backward_subsumes* e *backward_subsumed*.

Com a finalidade de introduzir subsunção entre conjuntos diferentes, a exemplo de uma cláusula global subsumir uma cláusula local, teve-se que acrescentar uma nova verificação. A primeira iteração do algoritmo para subsunção procura entre cláusulas do mesmo nível modal, como anteriormente já implementado. Acrescentou-se uma nova iteração onde é testado se o conjunto dado como argumento é global. Caso seja verdadeiro, é conferido qual conjunto global, isto é *g_usable* ou *g_sos*, será usado para verificar subsunção. Se este conjunto não for vazio, é feita a iteração com o conjunto global de forma análoga ao caso padrão, mas usando o fato de que o nível global unifica com qualquer nível modal local.

Para terminar dentro da função *self_subsumption* é testada uma sucessão de *forward_subsumed*. Em cada chamada da função, verifica-se subsunção com um dos conjuntos proposicionais, incluindo *g_sos* e *g_usable*. Em seguida, verifica-se subsunção com *backward_subsumes*.

$sos(clauses).$ $* : p \vee q.$ $* : p.$ $1 : r.$ $end\_of\_list.$

Tabela 4.2: Exemplo de uma subsunção para frente entre as cláusula $* : p$ e $* : p \vee q$.

$sos(clauses).$ $* : p \vee q \vee r.$ $* : p \vee q.$ $1 : s \vee t.$ $* : s.$ $end\_of\_list.$

Tabela 4.3: Exemplo de uma subsunção para frente entre uma cláusula global $* : s$ e uma cláusula local $1 : s \vee t$.

4.8 Experimentação

A seguir são apresentados alguns dos testes que foram utilizados para verificar se subsunção estava sendo corretamente implementada.

Destaca-se que o número de cada cláusula é assumido para ser contado de baixo para cima, ou seja no primeiro exemplo na Tabela 4.2 a primeira cláusula é para ser $1 : r$, a segunda cláusula $* : p$ e assim por diante. Todavia dentro do KSP a ordem das cláusulas é dado de forma aleatória. Portanto encontrar casos consistentes em que ocorra *backward_subsumption* acaba sendo dificultado, haja vista que a ordem pode ser mudada.

No exemplo na Tabela 4.3 temos uma subsunção entre uma cláusula global $* : s$ e uma cláusula local $1 : s \vee t$.

O programa resultante deste trabalho foi também testado utilizando o conjunto de testes LWB [9]. Entretanto, para nenhuma das fórmulas insatisfatíveis, a nova implementação produziu resultado correto. Isto se deve, supõe-se, ao fato de que rótulos (que foram introduzidos neste trabalho) e níveis modais (que são usados em todas as fases do provador, incluído a implementação do cálculo em si) não estejam corretamente relacionados.

$sos(clauses).$
$* : p \vee q.$
$2 : s.$
$* : p \vee q.$
$1 : r.$
$end\_of\_list.$

Tabela 4.4: Exemplo de uma repetição no conjunto global.

$sos(clauses).$
$* : p|q.$
$1 : r|s.$
$* : p \rightarrow []r.$
$1 : s \rightarrow < 45 > x.$
$end\_of\_list.$

Tabela 4.5: Exemplo de um conjunto de cláusulas com rótulos como entrada.

$sos(clauses).$
$p|q.$
$r|s.$
$p \rightarrow []r.$
$s \rightarrow < 45 > x.$
$end\_of\_list.$

Tabela 4.6: Exemplo de um conjunto de cláusulas sem rótulos como entrada.

Capítulo 5

Conclusão

O problema tratado neste trabalho é a combinação do raciocínio local e global no provador KSP, com foco na fase de preprocessamento.

Primeiramente para atingir este objetivo foram introduzidos dois novos padrões no arquivo de entrada para o gerador do analisador léxico, com o propósito de criar a estrutura de cláusula apresentada na Seção 2.3. Em seguida foi alterado o arquivo de entrada para o gerador do analisador sintático, a fim de adicionar regras para cada novo caso de cláusula. Em outras palavras, cláusula inicial, cláusula literal e cláusula modal receberam mais dois casos com rótulos locais e rótulos globais cada, totalizando mais seis regras. Destaca-se que essa abordagem foi utilizada, com o intuito de preservar o comportamento do programa.

Na sequência é acrescentado um rótulo a cada nó na árvore sintática, este rótulo podendo assumir valores determinados, por exemplo, no caso de uma cláusula local e uma cláusula global ou não-determinado. É também introduzida uma função que atualiza os rótulos dos nós filhos, caso necessário, com os rótulos dos nós pais. O objetivo dessa função é fazer os literais de uma cláusula herdarem o rótulo dela.

São utilizadas quatro variáveis para representar os conjuntos globais, sendo eles: o conjunto global de suporte literal, o conjunto global de usáveis literal, o conjunto global de suporte modal e o conjunto global de usáveis modal. Cada nova cláusula gerada na fase de forma normal do provador é inserida dependendo do seu rótulo em alguns desses conjuntos ou em alguns conjuntos já presentes no KSP.

Na parte de redundância são adicionadas novas verificações para os conjuntos globais, como também são inseridas novas verificações nos procedimentos relativos à subsunção, para tratamento dos casos em que uma cláusula global possa subsumir uma cláusula local, ou uma cláusula local ser subsumida por uma cláusula global. Por último são adicionadas novas operações para subsunções com os conjuntos globais na função *self_subsumption*.

Em suma, espera-se com as novas implementações que o provador reconheça como entrada a combinação de cláusulas locais e globais, como também faça procedimentos para preencher suas estrutura de dados e remova cláusulas no caso de redundância entre cláusulas locais e cláusulas globais.

Considerando que as novas implementações foram testadas sobre o LWB (*Logics Workbench*), que é um conjunto básico de fórmulas em lógica modal [9], sabe-se que o resultado obtido não é o esperado no caso de fórmulas insatisfatíveis. Isto ocorre com uma certa combinação de argumentos, por exemplo eliminação de literais puros por nível modal quando junto com a nova implementação de subsunção gera resultados inesperados para os casos de teste.

Por último, o ciclo de preprocessamento não foi fechado completamente para a combinação do raciocínio local e global, faltando ainda a implementação de procedimentos relativos a resolução unitária, eliminação de literais puros e o processamento de fórmulas. Como trabalho futuro, o foco é terminar o ciclo de preprocessamento, como ainda fazer uma integração da fase de preprocessamento com a fase de cálculo, para que o provador esteja apto a lidar com a combinação do raciocínio global e local.

Referências

[1] Fagin, Ronald, Joseph Y. Halpern, Yoram Moses e Moshe Y. Vardi: *Reasoning About Knowledge*. MIT Press, Cambridge, MA, USA, 2003, ISBN 0262562006. 1

[2] Garson, James: *Modal Logic*. Em Zalta, Edward N. (editor): *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2021 edição, 2021. 3

[3] Cláudia Nalon, Ullrich Hustadt, Clare Dixon: *$K_S P$ a resolution-based theorem prover for Kn: Architecture, refinements, strategies and expirements*. Journal of Automated Reasoning, 64:461–484, 2020. 3, 6

[4] Goranko, Valentin e Solomon Passy: *Using the universal modality: gains and questions*. Journal of Logic and Computation, 2(1):5–30, 1992. 6

[5] Nalon, Cláudia, Clare Dixon e Ullrich Hustadt: *Modal resolution: Proofs, layers, and refinements*. ACM Transactions on Computational Logic, 20(4):23:1–23:38, agosto 2019, ISSN 1529-3785. `http://doi.acm.org/10.1145/3331448`. 7, 8, 9, 10

[6] McCune, William: *Otter 3.3 reference manual*. arXiv preprint cs/0310056, 2003. 11

[7] *The fast lexical analyzer*. `https://westes.github.io/flex/manual/`. Última vez acessado: 06/10/2022. 13

[8] *Gnu bison*. `https://www.gnu.org/software/bison/manual/bison.html`. Última vez acessado: 06/10/2022. 13

[9] Balsiger, Peter, Alain Heuerding e Stefan Schwendimann: *A benchmark method for the propositional modal logics K, KT, S4*. Journal of Automated Reasoning, 24(3):297–317, Apr 2000. 24, 27