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

# Algoritmos de seleção de cláusulas em provas por resolução para lógicas modais

José Marcos da Silva Leite

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

# Algoritmos de seleção de cláusulas em provas por resolução para lógicas modais

José Marcos da Silva Leite

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

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

Prof. Dr. Bruno César Ribas
FGA/UnB

Prof. Dr. Thiago de Paulo Faleiros
CIC/UnB

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

Brasília, 5 de novembro de 2021

# Dedicatória

Dedico este trabalho à minha orientadora Prof.a Dr.a Cláudia Nalon que sempre me incentivou e sem a qual não teria conseguido concluir esta tarefa.

# Agradecimentos

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

Neste trabalho, propomos algumas heurísticas de seleção de cláusulas para provadores de Lógica Modal baseado em resolução e as comparamos a algumas já existentes. A implementação foi feita modificando o provador KSP. Ao adaptar estratégias de lógicas superiores conseguimos pequena melhora que se manteve após expandir a técnica de forma dinâmica. Analisamos ainda uma estratégia para as características específicas do problema, mas não houve melhoria. Um possível trabalho futuro consiste em analisar os motivos de não haver melhora, bem como a seleção do conjunto de candidatos.

**Palavras-chave:** lógica modal, provador de teoremas, resolução, seleção de cláusula

# Abstract

This work proposes new clause selection methods for resolution-based provers and compares them to existing ones. All algorithms were implemented and tested in KSP. First-order methods were adapted to modal logic and there was some improvement that was maintaned after dynamic scheduling. A new method was developed exploiting the particular structure of modal problems but it did not perform well. Future work includes finer analysis of the new method and a closer look into the candidate clause selection for resolution rules GEN1 and GEN3 to see it they can contribute to further performance improvement.

**Keywords:** modal logic, automated theorem prover, resolution, clause selection

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

Em computação, estudamos classes de problemas que ajudam a expressar quão difícil resolver um problema é, classificando-o por seu consumo de tempo ou espaço. P é a classe de problemas que podem ser resolvidos em tempo polinomial em uma máquina de Turing determinística. NP é a classe de problemas que podem ser resolvidos em tempo polinomial em uma máquina não determinística. PSPACE é a classe de problemas que podem ser resolvidos em espaço polinomial. Dentre outras, as classes P e NP estão contidas dentro da classe PSPACE, o que demonstra a sua representatividade [Pap94].

O problema de satisfatibilidade na lógica modal proposicional, chamada $K_n$, é PSPACE-completo [Lad77]. Isto significa que todo problema na classe PSPACE pode ser representado nesta lógica. A grande expressividade de $K_n$ permite representar minimização de Autômatos Finitos Não-determinísticos [JR91], formalizar fluxos de protocolos de autenticação [OGL09], computar o Equilíbrio de Nash para qualquer jogo de dois jogadores em forma normal [GPS13], ou até representar problemas de outras lógicas [Sch91], o que possibilita seu uso desde engenharia de requisitos [CBCG10] a geração contos de fadas [PD04].

Símbolos representam fatos, fórmulas representam sentenças com fatos. Uma fórmula ser satisfatível significa que existe uma combinação de cada fato ser verdadeiro ou falso que faz a fórmula inteira ser verdadeira. Uma fórmula é satisfatível se existe uma testemunha de valor de verdade verdadeiro. Em lógica proposicional clássica, basta determinar o valor de cada símbolo proposicional de tal forma que a fórmula seja verdadeira. Caso não exista uma testemunha, a fórmula é insatisfatível. Para lógica modal, uma testemunha precisa também de um conjunto de mundos e de relações entre eles, onde um mundo representa um conjunto de fatos.

Um cálculo para uma lógica é formado por um conjunto de axiomas e um conjunto de regras de inferências. Axiomas são sentenças assumidas como válidas, como $0 + 1 = 1$. Regras de inferência geram fórmulas a partir de um conjunto dado, apenas analisando a

sintaxe das mesmas. Por exemplo, para o conjunto o $\{p, p \rightarrow q\}$ uma regra de inferência poderia gerar $q$ enquanto outra pode gerar $p \vee r$. Normalmente nos interessamos por regras de inferência que preservem satisfatibilidade. Uma prova é uma sequência de fórmulas obtidas a partir de axiomas ou da aplicação das regras de inferência a fórmulas anteriores na sequência. Fórmulas para as quais existe uma prova são chamadas de teoremas. Em cálculos completos e corretos, o conjunto de teoremas e o conjunto de tautologias coincidem.

Para provar um teorema, podemos usar prova por contradição verificando se sua negação é insatisfatível. Neste tipo de prova, assumimos que a fórmula seja falsa e aplicamos regras de inferências até que uma contradição seja produzida. Caso não seja possível produzi-la, a fórmula não é um teorema. Podemos, então, reduzir prova de teoremas a verificação de satisfatibilidade. Provadores automáticos são interessantes por diminuir o risco de erro humano na prova e não precisarem de qualquer interação para produzir uma prova.

Resolução é um cálculo criado para verificar a satisfatibilidade de fórmulas da lógica proposicional e de primeira-ordem [Rob65]. No caso clássico, um problema é transformado em um conjunto de cláusulas. As regras de inferência são então aplicadas exaustivamente até que uma prova seja encontrada ou não seja mais possível gerar novas cláusulas. As implementações deste cálculo, em geral, são baseadas em um laço, onde uma cláusula é escolhida e outras cláusulas candidatas são selecionadas para que as regras de inferência sejam aplicadas. As provas de corretude e completude do cálculo independem de como as cláusulas são selecionadas e, naturalmente, métodos diferentes levam a desempenhos diferentes. Heurísticas para tal seleção já foram estudadas para problemas em primeira-ordem [SM16], mas ainda há muito a ser melhorado mesmo ao estado da arte.

É desejável ter ferramentas para resolver problemas em $K_n$ e que respeitando os limites teóricos produzam resposta rápida. Para isso, muitos provadores de teoremas foram construídos para tal lógica, por exemplo, [PSV06, TH06, GKS10, KT13, GOT14, GSB17]. KSP é um provador automático para o problema de satisfatibilidade modal [NHD20], cujo cálculo baseado em resolução é proposto em [NHD16, NDH19]. Seu desempenho, quando comparado a provadores modernos, é muito bom. Entretanto, os métodos de seleção de cláusulas são razoavelmente simples e fixas durante toda a busca por uma prova. Neste trabalho, abordamos o problema de seleção de cláusulas neste provador, propondo um escalonador entre as diferentes heurísticas. O escalonador implementado tem dois modos, podendo escolher entre diferentes estratégias em uma razão fixa ou variar esta razão (e estratégias) de acordo com o tamanho do conjunto em que estão as cláusulas disponíveis para a escolha. Uma terceira e nova estratégia, baseada na estrutura modal do problema, também é proposta.

No Capítulo 2 apresentamos toda a teoria necessária para o trabalho, incluindo a sintaxe e semântica da lógica $K_n$, bem como o cálculo baseado em resolução para esta lógica [NDH19] e descrição sumária de sua implementação [NHD20]. O Capítulo 3 descreve as heurísticas de seleção de cláusulas já implementadas e apresenta os resultados da performance de cada uma delas. No Capítulo 4, apresentamos as heurísticas que foram desenvolvidas neste trabalho, experimentos feitos e suas respectivas análises. Conclusões e trabalhos futuros são apresentados no Capítulo 5.

# Capítulo 2

# Definições

Neste capítulo apresentamos as definições básicas para o resto do texto. O exemplo usado abaixo foi baseado em [Bak16].

Lógica clássica é razoavelmente expressiva: podemos expressar qualquer problema em NP como o problema de satisfatibilidade booleana [Coo71]. A formalização de sentenças no formato "se $\phi$, então $\psi$" é feita pela implicação, $\phi \to \psi$. Esta sentença é verdadeira caso o antecedente, $\phi$, seja falso ou o consequente, $\psi$, seja verdadeiro. Dessa forma, sempre que $\phi$ acontece, $\psi$ também acontece. Essa formalização não é adequada para sentenças cujo antecedente nega algum fato, pois toda sentença nesse formato será verdadeira. Por exemplo, em "Se Nalon não tivesse feito KSP, então alguém teria feito" não parece intuitivamente correto, embora seja verdade em lógica clássica. Mais ainda, em "Se Nalon nunca tivesse estudado lógica, então ela teria feito o KSP" parece absurda, mas ainda assim seria verdadeira.

Lógica modal introduz conceitos de possibilidade e necessidade através de mundos possíveis. Um mundo pode ser entendido como um conjunto de fatos consistentes. Estes estão relacionados entre si e caso um mundo $w$ esteja relacionado com $w'$, então $w'$ é visível a partir de $w$. Uma sentença é possível se for verdadeira em um mundo visível e é necessária se for verdadeira em todos.

A interpretação da sentença "Se Nalon nunca tivesse estudado lógica, então ela teria feito o KSP" na lógica modal seria considerar um mundo visível onde Nalon nunca estudou lógica e verificar se lá ela fez o KSP. É uma interpretação bem mais fiel do sentido comumente usado. Em linguagem natural, implicações cujo antecedente contradiz um fato $\psi$, em geral, verificam a verdade do consequente em um mundo similar ao nosso, mas em que $\psi$ não acontece.

Lógica modal também pode representar, e raciocinar, sobre conhecimento e crença. Isso é especialmente interessante por introduzir o conceito de agentes, ou quem vê os mundos. Sentenças como "João acredita que vacina não funciona" são interpretadas como

"Em todo mundo que João vê, a vacina não funciona". Os mundos visíveis por cada agente podem ser diferentes. Podemos, também, encadear conceitos como em "João acredita que Maria sabe que $\mathsf{P} = \mathsf{NP}$".

## 2.1 Lógicas modais

A seguir, apresentamos a descrição formal desta linguagem bem como o cálculo. As definições foram adaptadas de [NDH19].

## 2.2 Linguagem

Trabalharemos com a linguagem lógica modal $K_n$. Nesta seção descreveremos primeiro a sintaxe da linguagem, isto é, como podemos construir e reconhecer fórmulas nesta linguagem. Depois descrevemos a sua semântica, isto é, como atribuímos significado a fórmulas a partir de estruturas de Kripke [BdRV02].

**Definição 1** Sejam $\mathcal{P} = \{p, q, r, \dots\}$ um conjunto enumerável de símbolos proposicionais e $\mathcal{A} = \{1, 2, 3, \dots, n\}, n \in \mathbb{N}$. Definimos o conjunto de fórmulas bem formadas $\mathcal{FBF}$ indutivamente.

- Se $\varphi \in \mathcal{P}$ então $\varphi \in \mathcal{FBF}$
- Se $\varphi \in \mathcal{FBF}$, $\psi \in \mathcal{FBF}$ e $a \in \mathcal{A}$, então $(\varphi \wedge \psi) \in \mathcal{FBF}$, $(\varphi \vee \psi) \in \mathcal{FBF}$, $(\varphi \rightarrow \psi) \in \mathcal{FBF}$, $\boxed{a}\varphi \in \mathcal{FBF}$, $\diamondsuit_a\varphi \in \mathcal{FBF}$ e $\neg\varphi \in \mathcal{FBF}$

**Definição 2** Denotamos por $\mathcal{LP}$ o conjunto de literais proposicionais e por $\mathcal{LM}$ o conjunto de literais modais: $\forall p \in \mathcal{P}, \forall a \in \mathcal{A}$, temos que $p \in \mathcal{LP}, \neg p \in \mathcal{LP}, \boxed{a}p \in \mathcal{LM}, \boxed{a}\neg p \in \mathcal{LM}, \diamondsuit_a p \in \mathcal{LM}, \diamondsuit_a \neg p \in \mathcal{LM}$

**Definição 3** Uma cláusula proposicional é uma disjunção de literais proposicionais, ou seja, tem a forma $p_1 \vee p_2 \vee \cdots \vee p_m$ com $p_i \in \mathcal{P}$ e $m \in \mathbf{N}$. Caso $m = 0$, a cláusula é dita vazia.

Seja $\Sigma = \{0, 1\}$. $\Sigma^*$ é o conjunto de todas as cadeias formadas com elementos de $\Sigma$. Em particular, $\epsilon$ representa a cadeia vazia. Construiremos cadeias em $\Sigma^*$ para codificar a posi-

ção de ocorrência de uma subfórmula em uma fórmula. Seja $inv$: {positiva, negativa} $\mapsto$ {positiva, negativa} tal que $inv$(positiva) = negativa e $inv$(negativa) = positiva.

**Definição 4** Definimos a polaridade de uma subfórmula em uma fórmula pela função $pol: \mathcal{FBF} \times \mathcal{FBF} \times \Sigma^* \mapsto$ {positiva, negativa}. Para $\varphi, \chi_1, \chi_2 \in \mathcal{FBF}, s \in \Sigma^*, a \in \mathcal{A}, val \in$ {positiva, negativa}.

- $pol(\varphi, \varphi, \epsilon) =$ positiva.
- Se $pol(\varphi, \chi_1 \vee \chi_2, s) = val$, então $pol(\varphi, \chi_1, s0) = pol(\varphi, \chi_2, s1) = val$
- Se $pol(\varphi, \chi_1 \wedge \chi_2, s) = val$, então $pol(\varphi, \chi_1, s0) = pol(\varphi, \chi_2, s1) = val$
- Se $pol(\varphi, \chi_1 \rightarrow \chi_2, s) = val$, então $pol(\varphi, \chi_1, s0) = inv(val)$ e $pol(\varphi, \chi_2, s1) = val$
- Se $pol(\varphi, \diamondsuit_a \chi_1, s) = val$, então $pol(\varphi, \chi_1, s0) = val$
- Se $pol(\varphi, \boxed{a}\chi_1, s) = val$, então $pol(\varphi, \chi_1, s0) = val$
- Se $pol(\varphi, \neg\chi_1, s) = val$, então $pol(\varphi, \chi_1, s0) = inv(val)$

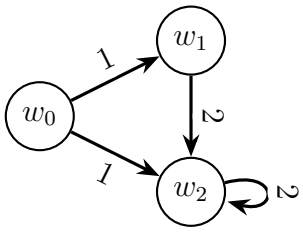
Dizemos que a polaridade de $\psi$ em $\varphi$ na posição $s$ é $pol(\varphi, \psi, s)$.

**Definição 5** Definimos o nível modal de uma subfórmula em uma fórmula pela função $mlevel: \mathcal{FBF} \times \mathcal{FBF} \times \Sigma^* \mapsto \mathbb{N}$. Para $\varphi, \chi_1, \chi_2 \in \mathcal{FBF}, s \in \Sigma^*, a \in \mathcal{A}, val \in \mathbb{N}$.

- $mlevel(\varphi, \varphi, \epsilon) = 0$.
- Se $mlevel(\varphi, \chi_1 \vee \chi_2, s) = val$ ou $mlevel(\varphi, \chi_1 \wedge \chi_2, s) = val$ ou $mlevel(\varphi, \chi_1 \rightarrow \chi_2, s) = val$, então $mlevel(\varphi, \chi_1, s0) = mlevel(\varphi, \chi_2, s1) = val$
- Se $mlevel(\varphi, \diamondsuit_a \chi_1, s) = val$ ou $mlevel(\varphi, \boxed{a}\chi_1, s) = val$, então $mlevel(\varphi, \chi_1, s0) = val + 1$
- Se $mlevel(\varphi, \neg\chi_1, s) = val$, então $mlevel(\varphi, \chi_1, s0) = val$

Dizemos que o nível modal de $\psi$ em $\varphi$ na posição $s$ é $mlevel(\varphi, \psi, s)$.

A semântica para lógica modal proposicional é dada por estruturas de Kripke. Uma estrutura de Kripke $M$ é da forma $M = (\mathcal{W}, w_0, \mathcal{R}_1, \ldots, \mathcal{R}_{|\mathcal{A}|}, \pi)$, onde $\mathcal{W}$ é um conjunto de mundos possíveis, $w_0 \in \mathcal{W}$, $\pi : \mathcal{W} \times \mathcal{P} \rightarrow \{true, false\}$ e $\mathcal{R}_a \subseteq \mathcal{W} \times \mathcal{W}$ para todo




| $\pi$ | $w_0$ | $w_1$ | $w_2$ |
|---|---|---|---|
| p | *false* | *true* | *true* |
| q | *false* | *true* | *true* |

Figura 2.1: Primeiro exemplo de modelo.

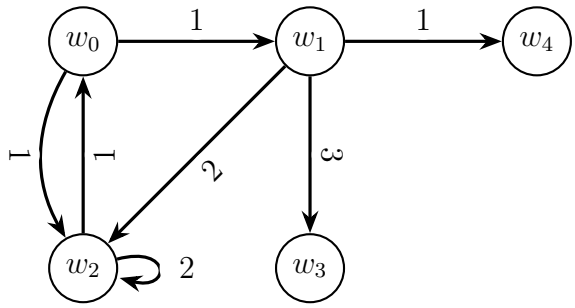
$a \in \mathcal{A}$. Dizemos que uma fórmula $\varphi$ é satisfeita na lógica modal $K_n$ no modelo $M$ no mundo $w$ se, e somente se, $\langle M, w\rangle \models \varphi$, conforme segue:

- $\langle M, w\rangle \models p$ se, e somente se, $\pi(w, p) = true$, para $p \in \mathcal{P}$
- $\langle M, w\rangle \models \neg\varphi$ se, e somente se, $\langle M, w\rangle \not\models \varphi$
- $\langle M, w\rangle \models (\varphi \wedge \psi)$ se, e somente se, $\langle M, w\rangle \models \varphi$ e $\langle M, w\rangle \models \psi$
- $\langle M, w\rangle \models (\varphi \vee \psi)$ se, e somente se, $\langle M, w\rangle \models \varphi$ ou $\langle M, w\rangle \models \psi$
- $\langle M, w\rangle \models (\varphi \rightarrow \psi)$ se, e somente se, $\langle M, w\rangle \not\models \varphi$ ou $\langle M, w\rangle \models \psi$
- $\langle M, w\rangle \models \diamondsuit_a\varphi$ se, e somente se, $\exists w', (w, w') \in \mathcal{R}_a$ e $\langle M, w'\rangle \models \varphi$
- $\langle M, w\rangle \models \boxed{a}\varphi$ se, e somente se, $\forall w'$, se $(w, w') \in \mathcal{R}_a$, então $\langle M, w'\rangle \models \varphi$

Uma fórmula $\varphi$ é localmente satisfatível se existe um modelo $M$ tal que $\langle M, w_0\rangle \models \varphi$. Uma formula $\varphi$ é globalmente satisfatível se existe um modelo $M$ tal que para todo $w \in \mathcal{W}$ temos que $\langle M, w\rangle \models \varphi$. Escrevemos $M \models \varphi$ se, e somente se, $\langle M, w_0\rangle \models \varphi$. Uma fórmula é dita insatisfatível caso não seja satisfatível.

Na Figura 2.1 apresentamos a visualização de um dos modelos para a fórmula $(p \vee \boxed{1}(p \wedge \diamondsuit_2 q))$. Como $\langle M, w_2\rangle \models q$ e ambos $w_1$ e $w_2$ estão relacionados com $w_2$, temos que $\langle M, w_1\rangle \models \diamondsuit_2 q$ e $\langle M, w_2\rangle \models \diamondsuit_2 q$; como $\langle M, w_1\rangle \models p$, temos que $\langle M, w_1\rangle \models (p \wedge \diamondsuit_2 q)$; como $\langle M, w_2\rangle \models p$ temos que $\langle M, w_2\rangle \models (p \wedge \diamondsuit_2 q)$; como $\langle M, w_1\rangle \models (p \wedge \diamondsuit_2 q)$ e $\langle M, w_2\rangle \models (p \wedge \diamondsuit_2 q)$, temos então $\langle M, w_0\rangle \models \boxed{1}(p \wedge \diamondsuit_2 q)$ e, finalmente, porque $\langle M, w_0 \models p$, temos que $\langle M, w_0\rangle \models (p \vee \boxed{1}(p \wedge \diamondsuit_2 q))$. Na Figura 2.2 é apresentado um modelo para $\boxed{1}\boxed{2}\diamondsuit_1\diamondsuit_1\boxed{3}(p \rightarrow q) \vee \diamondsuit_1\boxed{3}(p \rightarrow q) \wedge \boxed{1}\boxed{1}(\diamondsuit_1\neg q \vee \boxed{1}q)$.

**Definição 6** Para um modelo $M$, definimos a profundidade de um mundo pela função $depth\colon \mathcal{W} \mapsto \mathbb{N}$, onde $depth(w)$ é tamanho do menor caminho de $w_0$ a $w$ no grafo $(\mathcal{W}, \bigcup_{i=1}^{n} \mathcal{R}_i)$. Chamamos de camada modal a classe de equivalência com todos os mundos com mesma profundidade.




| $\pi$ | $w_0$ | $w_1$ | $w_2$ | $w_3$ | $w_4$ |
|---|---|---|---|---|---|
| p | *false* | *false* | *true* | *true* | *false* |
| q | *false* | *false* | *false* | *false* | *false* |

Figura 2.2: Segundo exemplo de modelo.

Podemos reduzir o problema de satisfatibilidade global ao problema de satisfatibilidade local com a extensão da linguagem $K_n$ pelo operador universal $\boxed{*}$. A semântica deste operador é definida como $\langle M, w\rangle \models \boxed{*}\varphi$ se, e somente se, para todo $w' \in \mathcal{W}$, $\langle M, w'\rangle \models \varphi$, onde $M = (\mathcal{W}, w_0, \mathcal{R}_1, \ldots, \mathcal{R}_{|\mathcal{A}|}, \pi)$.

**Definição 7** Uma fórmula está na forma normal negada caso seja formada somente por símbolos proposicionais, $\neg$, $\wedge$, $\vee$, $\boxed{a}$ e $\diamondsuit_a$ para $a \in \mathcal{A}$, e a negação só é aplicada a símbolos proposicionais.

É importante ressaltar que se $\varphi \in \mathcal{FBF}$ não está na forma normal negada, então $\varphi$ pode ser reescrita como $\psi \in \mathcal{FBF}$ na forma normal negada com semântica equivalente. Isto é, para todo $\langle M, w\rangle$, $\langle M, w\rangle \models \varphi$ se, e somente se, $\langle M, w\rangle \models \psi$.

## 2.3 Forma Normal Separada em Níveis Modais

O cálculo a ser apresentado utiliza uma outra linguagem chamada de Forma Normal Separada em Níveis Modais ($SNF_{ml}$).

**Definição 8** Uma fórmula em $SNF_{ml}$ é uma conjunção de cláusulas. Para $ml \in \mathbb{N} \cup \{*\}$ e $l_1, l_2 \in \mathcal{LP}$, cada cláusula está em um dos três formatos:

- $ml : c$, onde $c$ é uma cláusula proposicional
- $ml : l_1 \rightarrow \boxed{a} l_2$
- $ml : l_1 \rightarrow \diamondsuit_a l_2$

A satisfatibilidade de uma fórmula em $SNF_{ml}$ é definida a partir da satisfatibilidade em $K_n$. Sejam $ml : \varphi$ e $ml : \psi$ cláusulas em $SNF_{ml}$ e $M$ um modelo na lógica $K_n$.

- $M \models * : \varphi$ se, e somente se, $M \models \boxed{*}\varphi$.
- $M \models (ml : \varphi) \wedge (ml : \psi)$ se, e somente se, $M \models ml : \varphi$ e $M \models ml : \psi$.
- $M \models ml : \varphi$ se, e somente se, para todo $w$ tal que $depth(w) = ml$, temos $\langle M, w\rangle \models \varphi$.

Uma função de tradução de $K_n$ para $SNF_{ml}$ bem como a prova de que a tradução de uma fórmula preserva satisfatibilidade podem ser encontradas em [NDH19].

Seja $\geq$ uma ordem total sobre os símbolos proposicionais. Estendemos esta ordem para os literais da seguinte forma: Se $p \in \mathcal{P}$, então $\neg p \geq p$; se $p, q \in \mathcal{P}, p \neq q$ e $p \geq q$, então $p \geq \neg q$.

**Definição 9** O literal $l$ é máximo em $\varphi \in SNF_{ml}$ se, e somente se, $l$ ocorre em $\varphi$ e não há $l_2 \neq l$ em $\varphi$ tal que $l_2 \geq l$.

Por exemplo, seja a ordem total sobre os símbolos proposicionais $(p, q, r, ...)$, a ordem sobre os literais será $(\neg p, p, \neg q, q, \neg r, r, ...)$. Para esta ordem e algum $ml \in \mathbb{N} \cup \{*\}$, o literal máximo de $ml : (p \vee \neg q)$ é $p$; o literal máximo de $ml : (\neg q \vee r)$ é $\neg q$; e o literal máximo de $ml : ((p \vee \neg q) \wedge (\neg q \vee r))$ é $p$.

**Definição 10** O tamanho de uma cláusula em $SNF_{ml}$ é a cardinalidade do conjunto contendo somente os literais na cláusula.

## 2.4 Regras de inferência

Uma regra de inferência é um recurso sintático que gera fórmulas a partir de um conjunto de premissas dado.

O cálculo dedutivo baseado em resolução $RES_{ml}$ para lógica $K_n$ foi descrito em [NDH19]. Para simplificar a descrição das regras de inferência faremos uso de uma função parcial de unificação $\sigma\colon P(\mathbb{N} \cup \{*\}) \mapsto \mathbb{N} \cup \{*\}$, tal que $\sigma(\{ml, *\}) = ml$, $\sigma(\{ml\}) = ml$ e indefinida caso contrário. As regras de inferência de $RES_{ml}$ são apresentadas na Figura 2.3 em formato usual com as premissas representadas acima da linha e a conclusão, chamada de resolvente, abaixo da linha. Nesta descrição temos que $* - 1 = *$ e $m$ pode ser 0. Essas regras só podem ser aplicadas se o resultado da unificação for definido. Demostrações de correção e corretude do cálculo podem ser encontradas em [NDH19].

[LRES]

$$\frac{ml : (D \vee l) \quad ml' : (D' \vee \neg l)}{\sigma(\{ml, ml'\}) : D \vee D'}$$

[MRES]

$$\frac{ml : (l_1 \rightarrow \boxed{a} l) \quad ml' : (l_2 \rightarrow \diamondsuit_a \neg l)}{\sigma(\{ml, ml'\}) : \neg l_1 \vee \neg l_2}$$

[GEN2]

$$\frac{\begin{array}{c} ml_1 : (l'_1 \rightarrow \boxed{a} l_1) \\ ml_2 : (l'_2 \rightarrow \boxed{a} \neg l_1) \\ ml_3 : (l'_3 \rightarrow \diamondsuit_a l_2) \end{array}}{\sigma(\{ml_1, ml_2, ml_3\}) : \neg l'_1 \vee \neg l'_2 \vee \neg l'_3}$$

[GEN1]

$$\frac{\begin{array}{c} ml_1 : (l'_1 \rightarrow \boxed{a} \neg l_1) \\ \vdots \\ ml_m : (l'_m \rightarrow \boxed{a} \neg l_m) \\ ml_{m+1} : (l' \rightarrow \diamondsuit_a \neg l) \\ ml_{m+2} : (l_1 \vee \ldots \vee l_m \vee l) \end{array}}{\begin{array}{c} ml : \neg l'_1 \vee \ldots \vee \neg l'_m \vee \neg l' \\ ml = \sigma(\{ml_1, \ldots, ml_{m+1}, ml_{m+2} - 1\}) \end{array}}$$

[GEN3]

$$\frac{\begin{array}{c} ml_1 : (l'_1 \rightarrow \boxed{a} \neg l_1) \\ \vdots \\ ml_m : (l'_m \rightarrow \boxed{a} \neg l_m) \\ ml_{m+1} : (l' \rightarrow \diamondsuit_a l) \\ ml_{m+2} : (l_1 \vee \ldots \vee l_m) \end{array}}{\begin{array}{c} ml : \neg l'_1 \vee \ldots \vee \neg l'_m \vee \neg l' \\ ml = \sigma(\{ml_1, \ldots, ml_{m+1}, ml_{m+2} - 1\}) \end{array}}$$

Figura 2.3: Regras de inferência

Note que, exceto por MRES e GEN2, todas as regras de inferência têm pelo menos uma premissa que é uma cláusula literal. A implementação usa esta cláusula para procurar as outras candidatas para a premissa: ou seja, determinar, no conjunto de cláusulas, as demais premissas que são utilizadas para a aplicação da regra. Quanto melhor for a escolha desta cláusula literal, encontramos uma prova mais rápido para fórmulas insatisfatíveis e, talvez, determinamos mais rápido caso seja satisfatível.

No próximo capítulo iremos apresentar a implementação deste cálculo.

# Capítulo 3

# KSP

Neste capítulo apresentamos a implementação do cálculo visto na Seção 2.4, apresentamos os métodos de seleção de cláusula já existentes e avaliamos experimentalmente os resultados de suas aplicações a um conjunto de testes [BHS00].

## 3.1 Algoritmo do KSP

Neste trabalho usaremos o KSP [NHD20], um provador baseado no cálculo visto na Seção 2.4 e que determina a satisfatibilidade de fórmulas em $K_n$. Caso insatisfatível, uma prova é fornecida. Embora o KSP tenha como entrada fórmulas em $K_n$, este faz a tradução para $SNF_{ml}$ e todo o cálculo é feito nessa linguagem.

KSP utiliza uma variação de conjunto de suporte [WRC65], técnica que restringe os candidatos possíveis para resolução. Na versão para lógica clássica de conjunto de suporte, o conjunto de cláusulas $\Delta$ é particionado em dois conjuntos: $\Gamma$, o conjunto de suporte ou não processado, e $\Lambda$, o conjunto *usable* ou processado. Cláusulas são selecionadas de $\Gamma$ e resolvidas com cláusulas em $\Lambda$. A cláusula selecionada é removida de $\Gamma$ e acrescentada a $\Lambda$. Os resolventes produzidos são inseridos em $\Gamma$.

Na extensão para $SNF_{ml}$, onde as cláusulas são rotuladas pelo nível modal, as cláusulas de todo nível modal $ml$ são particionadas em três conjuntos $\Gamma_{ml}^{lit}$, $\Lambda_{ml}^{lit}$ e $\Lambda_{ml}^{mod}$. Cláusulas proposicionais são particionadas em $\Gamma_{ml}^{lit}$ e $\Lambda_{ml}^{lit}$ como no caso em lógica clássica. Cláusulas modais são armazenadas em $\Lambda_{ml}^{mod}$. Note que nenhuma regra de inferência descrita na Figura 2.3 produz novas cláusulas modais.

No Algoritmo 1, descrevemos o algoritmo de busca por uma prova implementado no KSP.

**Algoritmo 1:** KSP-Proof-Search

**Entrada:** Fórmula em $K_n$

**Saída:** Satisfatibilidade da fórmula

```
1  preprocessamento-da-entrada;
2  tradução-para-SNF;
3  preprocessamento-de-clausulas;
4  Γ^lit ← ⋃ Γ^lit_ml;
5  enquanto Γ^lit ≠ ∅ faça
6      para todo nível modal ml faça
7          clausula ← given(ml);
8          se não redundante(clausula) então
9              LRES(clausula, ml, ml);
10             GEN3(clausula, ml, ml − 1);
11             GEN1(clausula, ml, ml − 1);
12             Λ^lit_ml ← Λ^lit_ml ∪ {clausula};
13         fim
14         Γ^lit_ml ← Γ^lit_ml \ {clausula};
15         se 0 : false ∈ Γ^lit_0 então
16             retorna insatisfatível;
17         fim
18         Γ^lit ← ⋃ Γ^lit_ml;
19     fim
20     retorna satisfatível;
21 fim
```

As Linhas 1-3 aplicam algumas regras de simplificação, traduzem a fórmula para linguagem $SNF_{ml}$, constroem os conjuntos *usable* e de suporte e aplicam as regras de inferência MRES e GEN2. As Linhas 9-11 aplicam as demais regras de inferências descritas na Seção 2.4.

A função given, Linha 7, é responsável por escolher uma cláusula dentre todas as candidatas possíveis do conjunto de suporte. Cada nível modal é independente. Naturalmente, a função given só considera as cláusulas do nível modal pedido. KSP implementa cinco variações dessa função: *menor*, *mais antiga*, *mais nova*, *mínima* e *máxima*; e o usuário pode escolher qual deseja utilizar. Em mais detalhe:

- Na variação *menor*, é selecionada uma cláusula com o menor tamanho em $\Gamma^{lit}_{ml}$.

- Na variação *mais antiga*, o provador armazena a ordem nas quais as cláusulas foram adicionadas ao conjunto de cláusulas e é selecionada a que foi adicionada antes de todas em $\Gamma_{ml}^{lit}$.
- *Mais nova* é análoga a *mais antiga*, mas é selecionada a que foi adicionada depois de todas as outras.
- Em *mínima*, é escolhida uma cláusula com o menor tamanho dentre as cláusulas com o menor literal máximo em $\Gamma_{ml}^{lit}$.
- Em *máxima*, é feita escolha análoga a *mínima* mas dentre cláusulas com o maior literal máximo em $\Gamma_{ml}^{lit}$.

Cada uma destas estratégias pode ser utilizada tão somente de forma isolada e é fixa durante toda a busca por uma prova. Na próxima seção, iremos apresentar o resultado da avaliação experimental destas estratégias.

## 3.2 Avaliação experimental

O sistema utilizado em todos os experimentos tem processador AMD FX-6300 com clock base de 3.5 GHz, 6 Gigabyte de memória RAM no Sistema Operacional Ubuntu 18.04. Escolhemos a versão 0.1.2 do KSP, a versão pública mais recente na data do experimento [NHD20]. A execução para cada fórmula teve 300 segundos de tempo limite.

O *benchmark* é o LWB [BHS00] com 21 fórmulas satisfatíveis e 21 fórmulas insatisfatíveis para cada uma de nove famílias. Cada família corresponde a um problema estruturado e cada instância $n$ em uma família representa um problema parametrizado em $n$ nessa estrutura. As estruturas variam de família para família. Por exemplo, o problema $n$ da família branch requer um modelo de $n + 1$ níveis, onde há muita ramificação em cada nível. O problema $n$ da família ph requer um modelo até três níveis, onde o último tem um problema da casa dos pombos com $n$ casas e $n + 1$ pombos. As famílias de fórmulas branch e ph são de maior interesse, pois estão no limite de problemas teóricos referentes à satisfatibilidade de fórmulas na lógica $K_n$ [HM92] e também são particularmente difíceis para cálculos baseados em resolução [Hak85].

Neste, como no capítulo seguinte, serão apresentados os resultados para fórmulas insatisfatíveis. Os resultados para fórmulas satisfatíveis podem ser encontrados no Apêndice A. Os resultados agregados para cada família estão na mesma linha de cada tabela.

Para fórmulas insatisfatíveis, os resultados para uma estratégia estão em quatro colunas rotulada R(esolvidas), G(eradas), P(rova), U(sada); respectivamente, representando o número de instâncias resolvidas dentro do tempo limite, a média do número de cláusulas

geradas dividida por mil, a média do tamanho das provas dividido por mil e a porcentagem entre a média do tamanho de prova e média de cláusulas geradas. Adicionalmente temos, na última linha, a quantidade total de fórmulas resolvidas, média total da quantidade de cláusulas geradas dividida por mil, a média total dos tamanhos das provas e porcentagem de cláusulas que foram usadas em provas no total.

Em cada tabela, a estratégia vencedora para cada família está destacada em azul. Uma estratégia é vencedora se resolve um maior número de problemas. Para fórmulas insatisfatíveis, caso haja empate, é considerada vencedora aquela que tiver maior razão entre o tamanho da prova e o número de cláusulas; persistindo o empate, considera-se vencedora a que produzir menor número de cláusulas. Quando não há estratégia vencedora, nenhuma é destacada na tabela.

Para poder comparar os diferentes algoritmos de seleção vamos usar a configuração mais básica do KSP com somente resolução sem nenhuma restrição especial sobre as cláusulas candidatas para as premissas. Nessa versão, KSP gera muito mais cláusulas por não utilizar muitas das técnicas de simplificação e, por isso, a diferença na seleção se torna mais perceptível. Ainda não há nenhum resultado publicado com esta configuração, portanto fizemos um experimento inicial para extrair o desempenho dos algoritmos já implementados. Neste experimento, como nos demais, coletamos o número de cláusulas geradas e o tamanho da prova, nos casos insatisfatíveis, por serem boas medidas para análise de desempenho de estratégias de prova [Pla21, PZ97].

| | *Mais antiga* | | | | *Mais nova* | | | | *Mínima* | | | | *Máxima* | | | | *Menor* | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Família | R | G | P | U | R | G | P | U | R | G | P | U | R | G | P | U | R | G | P | U |
| branch_p | 2 | 24.0 | 28.0 | 0.1 | 2 | 2464.0 | 33.5 | 0.0 | 2 | 42.9 | 28.0 | 0.1 | 1 | 0.1 | 15.0 | 22.7 | 3 | 28.6 | 36.7 | 0.1 |
| d4_p | 21 | 10.9 | 77.0 | 0.7 | 21 | 2.7 | 77.0 | 2.9 | 21 | 2.2 | 77.0 | 3.4 | 21 | 12.2 | 94.2 | 0.8 | 21 | 2.2 | 77.0 | 3.4 |
| dum_p | 21 | 16.1 | 88.7 | 0.5 | 21 | 12.3 | 90.9 | 0.7 | 21 | 32.7 | 124.7 | 0.4 | 21 | 20.3 | 87.7 | 0.4 | 21 | 2.7 | 88.7 | 3.3 |
| grz_p | 21 | 0.6 | 86.0 | 14.2 | 21 | 0.3 | 87.0 | 26.6 | 21 | 0.5 | 89.0 | 17.2 | 21 | 0.5 | 86.0 | 15.7 | 21 | 0.3 | 84.0 | 28.3 |
| lin_p | 21 | 0.0 | 0.0 | 0.0 | 21 | 0.0 | 0.0 | 0.0 | 21 | 0.0 | 0.0 | 0.0 | 21 | 0.0 | 0.0 | 0.0 | 21 | 0.0 | 0.0 | 0.0 |
| path_p | 21 | 0.8 | 139.0 | 17.1 | 21 | 0.8 | 139.0 | 17.1 | 21 | 0.8 | 139.0 | 17.1 | 21 | 0.8 | 139.0 | 17.1 | 21 | 0.8 | 139.0 | 17.1 |
| ph_p | 2 | 0.5 | 15.5 | 3.2 | 2 | 0.3 | 20.0 | 7.8 | 3 | 126.5 | 51.0 | 0.0 | 3 | 526.7 | 56.3 | 0.0 | 3 | 1.0 | 43.7 | 4.4 |
| poly_p | 15 | 105.4 | 570.9 | 0.5 | 10 | 132.4 | 314.0 | 0.2 | 15 | 148.4 | 570.9 | 0.4 | 15 | 137.8 | 570.9 | 0.4 | 16 | 118.1 | 633.5 | 0.5 |
| t4p_p | 21 | 3.1 | 83.0 | 2.7 | 21 | 6.0 | 82.0 | 1.4 | 21 | 2.2 | 83.0 | 3.7 | 21 | 3.1 | 83.0 | 2.7 | 21 | 2.0 | 83.0 | 4.1 |
| Total | 145 | 15.81 | 128.28 | 0.81 | 140 | 47.97 | 94.59 | 0.20 | 146 | 23.98 | 133.84 | 0.56 | 145 | 30.50 | 131.28 | 0.43 | 148 | 14.51 | 137.05 | 0.95 |

Tabela 3.1: Fórmulas insatisfatíveis resolvidas em até 300 segundos, número médio de cláusulas geradas dividido por mil, tamanho médio da prova dividido por mil e porcentagem de cláusulas geradas usadas na prova.

Pela Tabela 3.1 é evidente que uma porção muito pequena das cláusulas geradas são de fato usadas para a prova. Queremos aumentar este fator para que pouco tempo de processamento seja desperdiçado. Como nos interessamos pela provas de teoremas, temos um interesse maior na melhoria do desempenho para fórmulas insatisfatíveis. A melhor estratégia, neste caso, foi majoritariamente *menor* e uma intuição possível é a de que queremos gerar uma cláusula vazia, logo sempre trabalhar com cláusulas pequenas deve levar a isto mais rápido.

## 3.3 Trabalho anterior

Vamos primeiro entender algumas estratégias de seleção de cláusula para um provador baseado em saturação, assim como KSP, mas de primeira-ordem. Como lógica modal $K_n$ pode ser traduzida para primeira-ordem [BdRV02], vamos tentar reproduzir as estratégias em nosso provador.

A análise foi feita em [SM16]. O experimento foi feito no provador E, um provador para Lógica de Primeira-Ordem com igualdade baseado em superposição, uma variação de resolução para esta lógica, sobre 13774 fórmulas do *benchmark* TPTP [Sut09] com 300 segundos de tempo limite.

As heurísticas avaliadas foram: *Mais antiga*, *Contagem de Símbolos* e *Ordenada*. Em *Contagem de Símbolos*, é atribuído um peso a cada símbolo e é selecionada uma cláusula com a menor soma de pesos. No caso em que todos os símbolos têm peso um esta variação é idêntica a *menor* usada no KSP. *Ordenada* é uma variação de *Contagem de Símbolos* onde são preferida cláusulas com o menor número de literais maximais. Diferentemente de $SNF_{ml}$ que utiliza uma ordem total, aqui é usada uma ordem parcial e, portanto, podem existir vários literais maximais.

Também foram analisadas várias intercalações de duas dessas heurísticas em distribuições diferentes. Por exemplo, a cada 11 seleções de cláusulas, 10 são feitas pela heurística *Contagem de Símbolo* e uma é feita pela heurística *Mais antiga*.

São apresentados número de fórmulas resolvidas, tamanho da prova, número de inferências na prova, para todas as estratégias utilizadas. Vemos que a maioria das fórmulas que tiveram solução dentro do tempo limite foram resolvidas em poucos segundos mesmo com 300 segundos disponíveis. Os experimentos apontam não haver melhora em performance ao usar diferentes funções de peso para os literais. Todas as estratégias de contagem de símbolos tiveram ganho significativo de desempenho quando intercaladas com *Mais antiga*. Selecionar sempre cláusulas dadas na entrada primeiro melhorou performance no geral, mas não tanto com estratégias utilizando *Mais antiga*.

# Capítulo 4

# Heurísticas Propostas para Seleção de Cláusulas

Neste capítulo descrevemos a implementação de novos métodos de seleção de cláusula para o KSP e apresentamos os resultados da comparação entre os diferentes métodos propostos e também em relação aos métodos já existentes (descritos no Capítulo 4).

## 4.1 Intercalação estática de estratégias

Baseado no trabalho de [SM16], nossa primeira proposta de seleção de cláusula será intercalação de *menor* e *mais antiga*, já implementadas no KSP, numa razão $p$:$q$ informada pelo usuário. Dessa forma, as primeiras $p$ execuções da função given serão conforme *menor*, as próximas $q$ serão conforme *mais antiga*, as próximas $p$ conforme *menor* e assim por diante.

Testamos razões 1:10, a melhor encontrada por [SM16], bem como 1:5 e 1:13.

Na Tabela 4.1, podemos ver que os resultados não mudaram tanto em relação às implementações já existentes. Numa inspeção um pouco mais meticulosa e considerando também o apresentado na Tabela 3.1, vemos que a razão 1:13 foi melhor que simplesmente *menor* em branch_p, dum_p e ph_p.

Comparando os resultados da Tabela 4.1 com a melhor das estratégias da Tabela 3.1, podemos notar que em poly_p resolvemos menos problemas; em branch_p, d4_p, grz_p não houve melhoria: mais cláusulas foram geradas para todas as razões; em lin_p, path_p, poly_p, t4p_p obtivemos o mesmo resultado; em dum_p houve melhoria com maior razão entre o número de cláusulas geradas e aquelas utilizadas na prova. Em ph_p conseguimos resolver um problema a mais com um maior aproveitamento das cláusulas geradas em provas.

| | Razão 1:5 | | | | Razão 1:10 | | | | Razão 1:13 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Família | R | G | P | U | R | G | P | U | R | G | P | U |
| branch_p | 3 | 43.2 | 36.7 | 0.1 | 3 | 32.5 | 36.7 | 0.1 | 3 | 30.9 | 36.7 | 0.1 |
| d4_p | 21 | 2.3 | 77.0 | 3.4 | 21 | 2.3 | 77.0 | 3.4 | 21 | 2.3 | 77.0 | 3.4 |
| dum_p | 21 | 2.3 | 88.7 | 3.8 | 21 | 2.7 | 88.7 | 3.3 | 21 | 2.4 | 88.7 | 3.8 |
| grz_p | 21 | 0.4 | 89.0 | 22.1 | 21 | 0.3 | 84.0 | 27.2 | 21 | 0.4 | 89.0 | 22.9 |
| lin_p | 21 | 0.0 | 0.0 | 0.0 | 21 | 0.0 | 0.0 | 0.0 | 21 | 0.0 | 0.0 | 0.0 |
| path_p | 21 | 0.8 | 139.0 | 17.1 | 21 | 0.8 | 139.0 | 17.1 | 21 | 0.8 | 139.0 | 17.1 |
| ph_p | 3 | 0.8 | 43.3 | 5.7 | 4 | 0.4 | 31.5 | 7.2 | 4 | 0.4 | 31.5 | 7.6 |
| poly_p | 15 | 105.3 | 570.9 | 0.5 | 15 | 105.3 | 570.9 | 0.5 | 15 | 105.3 | 570.9 | 0.5 |
| t4p_p | 21 | 2.0 | 83.0 | 4.1 | 21 | 2.0 | 83.0 | 4.1 | 21 | 2.0 | 83.0 | 4.1 |
| Total | 147 | 12.77 | 128.00 | 1.00 | 148 | 12.50 | 126.40 | 1.01 | 148 | 12.42 | 127.11 | 1.02 |

Tabela 4.1: Fórmulas insatisfatíveis resolvidas com razão 1:5, 1:10 e 1:13 entre *menor* e *mais antiga* em até 300 segundos, número de cláusulas geradas dividido por mil, tamanho médio da prova dividido por mil e porcentagem de uso de cláusulas nas provas.

## 4.2 Intercalação dinâmica de estratégias

Como a seleção de cláusula em cada nível é feita de forma independente, podemos também usar algoritmos distintos em níveis distintos.

Com base nos resultados dos experimentos descritos na Seção 4.1, propomos intercalação de *menor* e *mais antiga* com razão dinâmica para todo nível.

Cada nível tem um escalonador responsável por escolher uma razão. Como não conhecemos a melhor solução a priori, esta razão mudará ao longo da execução do provador.

Para o nível modal $ml$, o escalonador usará apenas a cardinalidade do conjunto de candidatos (o conjunto de suporte $\Gamma^{lit}_{ml}$) para determinar qual razão aplicar. O usuário define $r_1, l_1, r_2, l_2, \ldots, r_x$, onde $r_i$ é uma razão, como descrita na Seção 4.1, e $l_i$ é um limiar para troca de razões. Para todo $i$, temos que $l_i < l_{i+1}$, ou seja, estes limites formam uma sequência estritamente crescente de limiares para troca de razões. Para cardinalidade menor ou igual a $l_1$ é usada razão $r_1$; para cardinalidade maior que $l_1$ e menor ou igual a $l_2$ é usada razão $r_2$; e assim por diante. Note que não há limite superior para a última razão, a $r_x$.

Os resultados apresentados na Tabela 4.2 foram obtidos com uso da seguinte estratégia, onde as razões estão no formato onde a primeira é *mais antiga* e a segunda é *menor*.

- 0:1, para conjunto de suporte até 100
- 1:13, para conjunto de suporte maior que 100 e menor ou igual a 1000
- 1:21, para conjunto de suporte maior que 1000 e menor ou igual a 10000

- 1:34, para conjunto de suporte maior que 10000 e menor ou igual a 50000
- 0:1, para conjunto de suporte maior que 50000

| Família | R | G | P | U |
|---|---|---|---|---|
| branch_p | 3 | 28.1 | 36.7 | 0.1 |
| d4_p | 21 | 2.2 | 77.0 | 3.4 |
| dum_p | 21 | 2.7 | 88.7 | 3.3 |
| grz_p | 21 | 0.3 | 84.0 | 28.3 |
| lin_p | 21 | 0.0 | 0.0 | 0.0 |
| path_p | 21 | 0.8 | 139.0 | 17.1 |
| ph_p | 4 | 76.9 | 122.0 | 0.2 |
| poly_p | 15 | 105.3 | 570.9 | 0.5 |
| t4p_p | 21 | 2.0 | 83.0 | 4.1 |
| Total | 148 | 14.46 | 128.85 | 0.89 |

Tabela 4.2: Fórmulas insatisfatíveis resolvidas em até 300 segundos, número de cláusulas geradas dividido por mil, tamanho médio da prova dividido por mil e porcentagem de uso de cláusulas nas provas com intercalação dinâmica.

Comparando os resultados da Tabela 4.2 com a melhor das estratégias da Tabela 4.1, podemos notar que em ph_p tivemos uma razão de aproveitamento menor por gerar muito mais cláusulas. Nas outras famílias não houve melhoria.

## 4.3 Intercalação dinâmica agregada à preferência de literais

A regra de inferência GEN1 no laço principal do KSP precisa de alguma cláusula com um literal que aparece num operador $\Diamond$, para qualquer $\alpha$. Para facilitar esta aplicação, vamos fazer seleção entre cláusulas com algum literal destes caso exista. Por exemplo, para o algoritmo de *menor*, vamos selecionar a cláusula de menor tamanho dentre as que têm algum literal no escopo de $\Diamond$.

Pelos resultados da Tabela 4.3, não houve melhora em relação à Tabela 4.2. Para branch_p, houve um aumento no número de cláusulas em relação a *menor*, com mais cláusulas sendo geradas; menos cláusulas, entretanto, que na estratégia de razões fixas (ver Tabela 4.1). Para d4_p, dum_p e t4_p, houve um aumento no número de cláusulas geradas e consequente piora do uso de cláusulas em provas (em relação a menor e às outras estratégias propostas). Em ph_p em específico resolvemos um problema a menos. Para poly_p, foram menos problemas resolvidos que *menor*, mas melhor uso de cláusulas em provas do que razões fixas e dinâmicas. Para grz_p, o uso de cláusulas em provas é melhor do que razões fixas, mas pior do que *menor* e razões dinâmicas.

| Família | R | G | P | U |
|---|---|---|---|---|
| branch_p | 3 | 29.6 | 36.7 | 0.1 |
| d4_p | 21 | 3.1 | 77.0 | 2.5 |
| dum_p | 21 | 3.9 | 88.7 | 2.3 |
| grz_p | 21 | 0.4 | 88.0 | 22.7 |
| lin_p | 21 | 0.0 | 0.0 | 0.0 |
| path_p | 21 | 0.8 | 139.0 | 17.1 |
| ph_p | 3 | 1.1 | 44.3 | 4.0 |
| poly_p | 15 | 96.9 | 570.9 | 0.6 |
| t4p_p | 21 | 2.1 | 83.0 | 4.0 |
| Total | 147 | 11.99 | 127.88 | 1.07 |

Tabela 4.3: Fórmulas insatisfatíveis resolvidas em até 300 segundos, número de cláusulas geradas dividido por mil, tamanho médio da prova dividido por mil e porcentagem de uso de cláusulas nas provas com intercalação dinâmica agregada.

# Capítulo 5

# Conclusão

Um ponto crucial para a eficiência de provadores de teoremas baseados em resolução é o método de seleção de cláusulas. Neste trabalho, primeiro estudamos como este problema foi resolvido para um provador de primeira-ordem e adaptamos a solução para o KSP. Não houve melhorias com as melhores razões para primeira-ordem, entretanto com diferentes razões houve um pequena melhora para o problema da casa dos pombos. Tentamos, então, uma abortagem dinâmica com base no tamanho do problema e esta manteve a melhoria anterior. Por fim, construímos outra estratégia depois de analisar a estrutura específica do problema mas não foi o suficiente para um melhor desempenho.

Vimos que nem sempre uma estratégia que funciona para um lógica também têm bons resultados para outra, mesmo com cálculo e lógicas parecidas. Apesar do ajuste para deixar a técnica mais dinâmica, conseguimos apenas manter o mesmo desempenho. Por último, ao fazer uma estratégia voltada para o nosso cálculo, ainda não conseguimos ver uma melhoria.

Como trabalho futuro, podemos analisar especificamente as razões para a última estratégia não ter melhora. Por exemplo, analisar a seleção do conjunto de candidatos feita em GEN1 e GEN3. Podemos, ainda, modificar a última estratégia para considerar a ordem topológica dos símbolos dentre as cláusulas modais de vários níveis.

# Referências


[Bak16] Kane Baker. Modal realism 1. `https://www.youtube.com/watch?v=gcC5g4A9IM4`. Last visited: 20/09/2021, 2016. 4

[BdRV02] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2002. 5, 16

[BHS00] Peter Balsiger, Alain Heuerding, and Stefan Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. *J. Autom. Reason.*, 24(3):297–317, April 2000. 11, 13

[CBCG10] Veronica Castaneda, Luciana Ballejos, Ma. Laura Caliusco, and Ma. Rosa Galli. The use of ontologies in requirements engineering. *Global Journal of Research In Engineering*, 10(6), 2010. 1

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery. 4

[GKS10] Daniel Götzmann, Mark Kaminski, and Gert Smolka. Spartacus: A tableau prover for hybrid logic. *Electronic Notes on Theoretical Computer Science*, 262:127–139, 2010. 2

[GOT14] Rajeev Goré, Kerry Olesen, and Jimmy Thomson. Implementing tableau calculi using BDDs: BDDTab system description. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning*, pages 337–343, Cham, 2014. Springer International Publishing. 2

[GPS13] Paul W. Goldberg, Christos H. Papadimitriou, and Rahul Savani. The complexity of the homotopy method, equilibrium selection, and Lemke-Howson solutions. *ACM Trans. Econ. Comput.*, 1(2), May 2013. 1

[GSB17] Tobias Gleißner, Alexander Steen, and Christoph Benzmüller. Theorem provers for every normal modal logic. In Thomas Eiter and David Sands, editors, *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46 of *EPiC Series in Computing*, pages 14–30. EasyChair, 2017. 2

[Hak85] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985. Third Conference on Foundations of Software Technology and Theoretical Computer Science. 13

[HM92] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, April 1992. 13

[JR91] Tao Jiang and B. Ravikumar. Minimal NFA problems are hard. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez Artalejo, editors, *Automata, Languages and Programming*, pages 629–640, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg. 1

[KT13] Mark Kaminski and Tobias Tebbi. InKreSAT: Modal reasoning via incremental reduction to SAT. In *CADE 2013*, volume 7898 of *LNCS*, pages 436–442. Springer, 2013. 2

[Lad77] Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, 1977. 1

[NDH19] Cláudia Nalon, Clare Dixon, and Ullrich Hustadt. Modal resolution: Proofs, layers, and refinements. *ACM Transactions on Computational Logic*, 20(4), August 2019. 2, 3, 5, 9

[NHD16] Cláudia Nalon, Ullrich Hustadt, and Clare Dixon. KSP: A resolution-based prover for multimodal K. In Nicola Olivetti and Ashish Tiwari, editors, *IJCAR 2016*, volume 9706 of *LNCS*, pages 406–415. Springer, 2016. 2

[NHD20] Cláudia Nalon, Ullrich Hustadt, and Clare Dixon. KSP a resolution-based theorem prover for $\mathsf{K}_n$: Architecture, refinements, strategies and experiments. *Journal of Automated Reasoning*, 64(3):461–484, Mar 2020. 2, 3, 11, 13

[OGL09] Mehmet A. Orgun, Guido Governatori, and Chuchang Liu. Modal tableaux for verifying stream authentication protocols. *Autonomous Agents and Multi-Agent Systems*, 19(1):53–75, August 2009. 1

[Pap94] Christos H. Papadimitriou. *Computatational Complexity*. Addison Wesley, 1994. 1

[PD04] Federico Peinado and Belen Díaz-Agudo. A description logic ontology for fairy tale generation. In *Language Resource for Linguistic Creativity Workshop, 4th LREC Conference*, 01 2004. 1

[Pla21] David A. Plaisted. Search spaces for theorem proving strategies. In Martin Suda and Sarah Winkler, editors, *ARCADE 2021 Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements*, 06 2021. 14

[PSV06] Guoqiang Pan, Uli Sattler, and Moshe Vardi. BDD-based decision procedures for the modal logic K. *Journal of Applied Non-Classical Logics*, 16:169–208, 01 2006. 2

[PZ97] David A. Plaisted and Yunshan Zhu. *The efficiency of theorem proving strategies - a comparative and asymptotic analysis*. Computational intelligence. Vieweg, 1997. 14

[Rob65] John A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, January 1965. 2

[Sch91] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, page 466–471, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. 1

[SM16] Stephan Schulz and Martin Möhrmann. Performance of clause selection heuristics for saturation-based theorem proving. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning*, pages 330–345, Cham, 2016. Springer International Publishing. 2, 16, 17

[Sut09] Geoff Sutcliffe. The TPTP problem library and associated infrastructure: the FOF and CNF parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, January 2009. 16

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR 2006*, volume 4130 of *LNCS*, pages 292–297. Springer, 2006. 2

[WRC65] Lawrence Wos, George A. Robinson, and Daniel F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the ACM*, 12:536–541, 1965. 11

# Apêndice A

# Avaliação Experimental: Fórmulas Satisfatíveis

Para fórmulas satisfatíveis, os resultados para uma estratégia estão em duas colunas rotuladas R(esolvidas) e G(eradas), representando o número de instâncias resolvidas dentro do tempo limite e a média do número de cláusulas geradas dividida por mil; na última linha da tabela, é adicionada a quantidade total de fórmulas resolvidas, média total da quantidade de cláusulas geradas dividida por mil. Em cada tabela, a estratégia vencedora para cada família está destacada em azul. Uma estratégia é vencedora se resolve um maior número de problemas. Em caso de empate, para fórmulas satisfatíveis é considerada vencedora a estratégia que produz menor número de cláusulas.

## A.1 Estratégias existentes

Para as estratégias existentes, o resultado de desempenho é apresentado na Tabela A.1. De forma geral, *menor* foi a melhor das estratégias.

## A.2 Novas estratégias

Na Tabela A.2, em comparação com a Tabela A.1, percebemos uma pequena melhora, na razão 1:13, em relação a *menor*. Para d4_n e dum_n, os resultados foram iguais. Houve uma piora significativa em ph_n, com quase 70% de cláusulas a mais sendo geradas. Entretanto, houve uma melhora significativa na razão 1:10 para a família grz_n, com apenas 10% de cláusulas sendo geradas em relação a *mínima* e também um pouco melhor do que *menor*. Esta queda no número de cláusulas faz com que, no resultado geral, a razão 1:10 seja, em geral, a melhor entre as estratégias. Vale lembrar que não esperamos muitas diferenças para fórmulas satisfatíveis por focarmos mais em prova de teoremas.

| | *Mais antiga* | | *Mais nova* | | *Mínima* | | *Máxima* | | *Menor* | |
|---|---|---|---|---|---|---|---|---|---|---|
| Família | R | G | R | G | R | G | R | G | R | G |
| branch_n | 2 | 157.1 | 1 | 1.2 | 1 | 0.9 | 1 | 0.7 | 2 | 50.4 |
| d4_n | 9 | 78.7 | 7 | 290.5 | 8 | 173.7 | 8 | 93.7 | 9 | 47.7 |
| dum_n | 21 | 4.2 | 21 | 6.7 | 21 | 7.0 | 21 | 5.2 | 21 | 3.3 |
| grz_n | 21 | 82.8 | 17 | 58.0 | 21 | 142.0 | 17 | 51.8 | 21 | 15.5 |
| lin_n | 21 | 0.0 | 21 | 0.0 | 21 | 0.0 | 21 | 0.0 | 21 | 0.0 |
| path_n | 21 | 0.7 | 21 | 0.7 | 21 | 0.7 | 21 | 0.7 | 21 | 0.7 |
| ph_n | 3 | 73.6 | 3 | 848.7 | 3 | 50.4 | 3 | 107.2 | 3 | 18.9 |
| poly_n | 16 | 123.3 | 8 | 367.5 | 14 | 109.7 | 14 | 144.2 | 16 | 123.3 |
| t4p_n | 21 | 4.9 | 21 | 8.7 | 21 | 7.3 | 21 | 4.6 | 21 | 3.9 |
| Total | 135 | 38.23 | 120 | 73.69 | 131 | 48.67 | 127 | 33.02 | 135 | 22.60 |

Tabela A.1: Fórmulas satisfatíveis resolvidas em até 300 segundos, número médio de cláusulas geradas dividido por mil.

| | Razão 1:5 | | Razão 1:10 | | Razão 1:13 | |
|---|---|---|---|---|---|---|
| Família | R | G | R | G | R | G |
| branch_n | 2 | 52.1 | 2 | 49.4 | 2 | 49.2 |
| d4_n | 9 | 48.0 | 9 | 48.0 | 9 | 47.7 |
| dum_n | 21 | 3.5 | 21 | 3.4 | 21 | 3.3 |
| grz_n | 21 | 16.1 | 21 | 14.2 | 21 | 14.9 |
| lin_n | 21 | 0.0 | 21 | 0.0 | 21 | 0.0 |
| path_n | 21 | 0.7 | 21 | 0.7 | 21 | 0.7 |
| ph_n | 3 | 36.0 | 3 | 32.7 | 3 | 32.0 |
| poly_n | 16 | 123.3 | 16 | 123.3 | 16 | 123.3 |
| t4p_n | 21 | 3.9 | 21 | 3.9 | 21 | 3.9 |
| Total | 135 | 23.16 | 135 | 22.73 | 135 | 22.80 |

Tabela A.2: Fórmulas satisfatíveis resolvidas com razão 1:5, 1:10 e 1:13 entre *menor* e *mais antiga* em até 300 segundos e quantidade de cláusulas geradas dividido por mil.

Os resultados apresentados na Tabela A.3 foram obtidos com uso da seguinte estratégia, onde as razões estão no formato onde a primeira é *mais antiga* e a segunda é *menor*.

- 0:1, para conjunto de suporte até 100
- 1:13, para conjunto de suporte maior que 100 e menor ou igual a 1000
- 1:21, para conjunto de suporte maior que 1000 e menor ou igual a 10000
- 1:34, para conjunto de suporte maior que 10000 e menor ou igual a 50000
- 0:1, para conjunto de suporte maior que 50000

Na Tabela A.3, os resultados foram piores para a família branch_n. Para grz_n, houve uma melhora em relação a *menor*, mas o resultado não foi tão bom quanto em

| Família | R | G |
|---|---|---|
| branch_n | 2 | 50.5 |
| d4_n | 9 | 47.7 |
| dum_n | 21 | 3.3 |
| grz_n | 21 | 15.4 |
| lin_n | 21 | 0.0 |
| path_n | 21 | 0.7 |
| ph_n | 3 | 32.0 |
| poly_n | 16 | 123.3 |
| t4p_n | 21 | 3.9 |
| Total | 135 | 22.89 |

Tabela A.3: Fórmulas satisfatíveis resolvidas em até 300 segundos e número de cláusulas gerado dividido por mil com intercalação dinâmica.

| Família | R | G |
|---|---|---|
| branch_n | 2 | 49.8 |
| d4_n | 9 | 57.2 |
| dum_n | 21 | 3.5 |
| grz_n | 21 | 15.7 |
| lin_n | 21 | 0.0 |
| path_n | 21 | 0.7 |
| ph_n | 3 | 26.8 |
| poly_n | 16 | 123.3 |
| t4p_n | 21 | 3.8 |
| Total | 135 | 23.47 |

Tabela A.4: Fórmulas satisfatíveis resolvidas em até 300 segundos e número de cláusulas gerado dividido por mil com intercalação dinâmica agregada.

1:10. Para ph_n, o resultado é o mesmo que 1:13, ou seja, ainda gerando mais cláusulas que *menor*. Para as outras famílias, o resultado foi o mesmo das estratégias anteriores.

Na Tabela A.4, houve menor geração de cláusulas para branch_n em relação a *menor*, mas o resultado não é tão bom quanto 1:13. Para ph_n, o resultado é pior do que *menor*, mas melhor do que os obtidos com as estratégias de intercalação estática e dinâmica. A família t4p_n apresenta o melhor resultado quando comparado com todas as estratégias anteriores. As demais famílias, isto é, d4_n, dum_n e grz_n apresentaram piores resultados do que os apresentados nas Tabelas A.3 e A.2.