Implementações de Tableaux para Raciocínio por Aproximações

Wagner Dias

Orientador:

Prof. Dr. Marcelo Finger

Dissertação apresentada ao Instituto de Matemática e Estatística da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Mestre em Ciências na área de Computação.

IME - USP

Aos meus pais
Vicentina e Osvaldo
e aos meus irmãos
Natália e Ricardo

Agradecimentos

Meus agradecimentos ao professor Marcelo Finger, pela paciência e disponibilidade durante a orientação.

À professora Renata Wassermann, pela dica dos slides e outros comentários preciosos.

Aos meus pais, Osvaldo e Vicentina, por toda a preocupação, cuidado e amor desde sempre.

Aos meus irmãos, Ricardo e Natália, pela companhia indispensável.

Ao Marcos Haftel, pelas longas conversas que me ajudaram muito.

Aos amigos Phalkon; Shigueo, Alexandre Noma, Telmo, Tadao, Lau, Kubo, Marcelo Vinagreiro; Leo Inoue e Cris Uchida; Janaína; Tati e Fábio Doy; Dani, Pat Oda, Pat Tárcia, Fê Etlinger, Vinho, Sabrina, Well. Pelo incentivo e pelas reuniões na lanchonete, na praça e no banco azul...

Ao Sifu Wagner Irineu e à União Ton Lon, por me aceitarem como filho e irmão.

Ao Yapapá, ao Todos os Cantos e ao Coral da Poli; e ao Luther King, por ter proporcionado momentos únicos de felicidade.

À Lucy, à Sira e ao Martinho Lutero, por terem acreditado em mim e por terem feito de mim uma pessoa muito mais feliz.

Resumo

Atualmente não se conhece nenhum método de prova para a lógica proposicional clássica que tenha tempo polinomial. Os métodos de tabela-verdade e o de tableaux analíticos são imediatamente implementáveis em uma máquina, mas já foi provado [D'A90] que nenhum dos dois é mais eficiente que o outro no caso geral. Por outro lado, o sistema de tableaux **KE** de D'Agostino é essencialmente mais eficiente que ambos. Uma outra abordagem para resolver o problema da validade de fórmulas é o *raciocínio por aproximações*. Cadoli e Schaerf [SC95] propuseram um método de raciocínio por aproximações com respostas aproximadas que: (a) dão informações semanticamente claras sobre o problema a cada passo de aproximação; (b) cada resposta aproximada é mais fácil de computar que a resposta do problema original; e (c) podem ser melhoradas, e eventualmente convergem para a resposta correta. No entanto este método está restrito ao formato clausal e não fornece uma heurística de aproximação. Finger e Wassermann [FW01] propuseram um método que generaliza o raciocínio por aproximações de Cadoli e Schaerf, eliminando a restrição a cláusulas e introduzindo a heurística de aproximação. Eles estendem a semântica da lógica S_3 usada no método de Cadoli e Schaerf para toda a lógica proposicional, e propõem o método de tableaux KE-S_3 para essa lógica — baseado nos tableaux KE de D'Agostino. O objetivo deste trabalho é implementar o método de Tableaux Analíticos, o método de Tableaux KE de D'Agostino e o Método de Tableaux KE-S_3 e fazer um teste comparativo dos métodos.

Abstract

At the present moment there is no knowledge of a polynomial proof method for classic propositional logic. The truth table and the analytic tableau methods are readily implementable in a machine, but none of these two methods is more efficient than the other, as proved in [D'A90]. On the other hand, D'Agostino's KE tableau is essentially more efficient than both methods. Another approach to solve the problem of determining the validity of formulae is *approximate reasoning*. Cadoli and Schaerf [SC95] have proposed one such method with approximate answers that: (a) provide semantically clear information at each approximation step; (b) are easier to compute than the answer of the original problem; and (c) can be improved, and eventually converge to the right answer. Finger and Wassermann [FW01] have proposed a method that generalizes Cadoli and Schaerf's approximate reasoning, eliminating the restriction to clauses. They extend the semantic of the logic S_3 used in Cadoli and Schaerf's method to the propositional logic, and propose the KE-S_3 tableau for this logic — based in D'Agostino's KE tableau. The goal of this work is to implement the analytic tableau, D'Agostino's KE tableau and Finger and Wassermann's KE-S_3 tableau, and compare these methods using benchmark formulae.

Sumário

Lista de Figuras

Lista de Tabelas

Introdução

Atualmente, uma das maiores questões no campo da Teoria da Computação é o problema $\mathcal{P} = \mathcal{NP}$.

$\mathcal{P}$ é uma *classe de complexidade* definida como o conjunto de *problemas de decisão* (problemas que admitem resposta do tipo *sim* ou *não*) que podem ser resolvidos em tempo polinomial — $O(n^k)$ para alguma constante k.

$\mathcal{NP}$ é a classe de complexidade dos problemas de decisão cuja resposta pode ser verificada em tempo polinomial através de um *certificado*.

Existe uma classe de problemas — a classe dos problemas $\mathcal{NP}$*-completos* — com a propriedade de que, se qualquer um dos problemas puder ser resolvido em tempo polinomial, então todo problema de $\mathcal{NP}$ é polinomial, ou seja, $\mathcal{P} = \mathcal{NP}$.

Mas até hoje nenhum problema $\mathcal{NP}$-completo tem solução polinomial conhecida, o que leva a maioria dos pesquisadores de Teoria da Computação a crer que $\mathcal{P} \neq \mathcal{NP}$, ou seja, que há problemas em $\mathcal{NP}$ de natureza inerentemente superpolinomial.

Um resultado conhecido é a prova de que o problema SAT é $\mathcal{NP}$-completo [Coo71, CLR90], ou seja, o problema de determinar se uma fórmula da lógica proposicional clássica é satisfatível (ou válida) é difícil.

De fato, não se conhece nenhum método de prova para a lógica proposicional clássica que tenha tempo polinomial.

O método da *tabela-verdade* — atribuído a Wittgenstein por [D'A90] — é imediatamente implementável em uma máquina, mas é exponencial: para uma fórmula com n átomos distintos, a tabela-verdade tem 2^n linhas.

O método de tableaux analíticos[2] é uma alternativa também facilmente imple-

[2] O método original é de autoria de Charles Peirce. Outros aperfeiçoaram o método, como Beth e Hintikka. O método consolidou-se a partir da versão de Smullyan, que é a versão implementada neste trabalho.

mentável. Mas há fórmulas (as *fórmulas "gordas"* — fórmulas que têm tamanho total grande comparado com o número de símbolos proposicionais distintos) cuja prova gera um número grande de ramos no tableau analítico comparado ao número de linhas da tabela-verdade correspondente. Em outras palavras, *tableaux analíticos não p-simulam*[3] *tabelas-verdade.*

Por outro lado, as tabelas-verdade não p-simulam os tableaux analíticos, e portanto os dois métodos são incomparáveis — um não é mais eficiente que o outro.

D'Agostino [D'A90] aponta uma razão para esse fenômeno como sendo uma inadequação das regras de ramificação do tableau analítico que afeta a complexidade das provas no caso geral. Ele propõe o sistema de tableaux **KE**, e mostra que **KE** p-simula tableau analítico, mas o inverso não é verdade, ou seja, o sistema **KE** é essencialmente mais eficiente que o sistema de tableaux analíticos.

Uma outra abordagem para resolver o problema da satisfatibilidade de fórmulas é o *raciocínio aproximado*.

O raciocínio aproximado apresenta vantagens particularmente quando se está num cenário de recursos limitados: se o tempo é limitado, pode-se usar a informação da resposta até então obtida como uma aproximação da resposta exata do problema original.

Cadoli e Schaerf [SC95] propuseram uma técnica de raciocínio aproximado. Em seu método, eles definem duas famílias de lógicas que servem como um limitante superior (incorreto e completo) e um limitante inferior (correto e incompleto) para a lógica clássica. O método busca respostas aproximadas nessas duas famílias de lógicas que satisfazem as seguintes propriedades: (a) dão informações semanticamente claras sobre o problema a cada passo de aproximação; (b) são mais fáceis de computar que a resposta do problema original; e (c) podem ser melhoradas, e eventualmente convergem para a resposta correta.

O método faz uma restrição à linguagem — fórmulas na forma clausal —, de modo a tornar viável a computação das respostas aproximadas. Além disso, não existe uma heurística que indica qual o próximo passo de aproximação.

[3] [D'A90] e [D'A92] descrevem a relação (de ordem parcial) de *p-simulação* (atribuída a Cook) como um modo de se medir a *complexidade relativa* de sistemas de prova. Sistemas que se p-simulam mutuamente podem ser considerados essencialmente de mesma complexidade. Um sistema **S** que p-simula um sistema $\mathbf{S}'$ mas o contrário não é verdade, pode ser considerado essencialmente mais eficiente que $\mathbf{S}'$.

Finger e Wassermann [FW01] propuseram um método que generaliza o raciocínio aproximado de Cadoli e Schaerf, eliminando a restrição a cláusulas. Eles estendem a semântica da lógica S_3 usada no método de Cadoli e Schaerf para toda a lógica proposicional, e propõem o método de tableaux KE-S_3 para essa lógica — baseado nos tableaux KE de D'Agostino —, e que além disso exibe uma heurística que indica a seqüência de passos de aproximação.

O objetivo deste trabalho é implementar o método de Tableaux Analíticos (versão de Smullyan), o método de Tableaux KE de D'Agostino e o Método de Tableaux KE-S_3 e fazer um teste comparativo dos métodos.

No Capítulo 1, são descritos os fundamentos básicos da lógica clássica proposicional em seus aspectos sintáticos e semânticos.

No Capítulo 2, são descritos o método de Tableaux Analíticos de Smullyan e o método de Tableaux KE de D'Agostino, ambos para a lógica proposicional clássica.

No Capítulo 3, são descritos o método de raciocínio aproximado de Cadoli e Schaerf e o método de Tableaux KE-S_3 de Finger e Wassermann.

No Capítulo 4, é discutida a implementação dos métodos de tableaux.

No Capítulo 5, são apresentados os resultados da comparação entre os métodos de tableaux.

Finalmente, no Capítulo 6 é apresentada a conclusão.

Capítulo 1

Lógica Proposicional Clássica

No decorrer de todo o texto, a não ser que seja explicitamente dito o contrário, usaremos as convenções a seguir.

1.1 Sintaxe

Os *conectivos lógicos* são os quatro símbolos:

1. $\neg$: *negação*;
2. $\wedge$: *conjunção*;
3. $\vee$: *disjunção*;
4. $\rightarrow$: *implicação*.

O conectivo da negação é chamado de *conectivo unário*. Os três últimos são chamados de *conectivos binários*.

As *variáveis proposicionais* (ou *variáveis* ou *átomos*) são os símbolos pertencentes ao conjunto enumerável $\mathcal{P} = \{p_1, p_2, \ldots\}$. Usaremos letras minúsculas $(p, q, r, s, \ldots)$ para denotar elementos de $\mathcal{P}$.

Os símbolos (e) são chamados de *parênteses* e são usados para pontuação.

Uma *fórmula* é um elemento que pode ser definido recursivamente a partir do seguinte conjunto de regras:

F_0: Todo elemento p de $\mathcal{P}$ é uma fórmula;

F_1: Se A é uma fórmula, então $\neg A$ é uma fórmula;

F_2: Se A, B são fórmulas, então $(A \wedge B)$ é uma fórmula;

F_3: Se A, B são fórmulas, então $(A \vee B)$ é uma fórmula;

F_4: Se A, B são fórmulas, então $(A \to B)$ é uma fórmula.

Usaremos letras maiúsculas $(A, B, C, \ldots)$ para denotar fórmulas e letras maiúsculas gregas $(\Gamma, \Delta, \ldots)$ para denotar conjuntos de fórmulas. O conjunto de todas as fórmulas será denotado por $\mathcal{L}$.

Os parênteses poderão ser omitidos quando permitido pelo contexto.

As *subfórmulas imediatas* de uma fórmula A são dadas pelas regras (B e C são fórmulas):

I_0: Se $A \in \mathcal{P}$, então A não tem subfórmulas imediatas;

I_1: Se A é da forma $\neg B$, então B é a única subfórmula imediata de A;

I_2: Se A é da forma $(B \wedge C)$, então B e C são as únicas subfórmulas imediatas de A;

I_3: Se A é da forma $(B \vee C)$, então B e C são as únicas subfórmulas imediatas de A;

I_4: Se A é da forma $(B \to C)$, então B e C são as únicas subfórmulas imediatas de A.

As *subfórmulas* de uma fórmula A são dadas pelas regras:

S_1: Se X é uma subfórmula imediata de A ou $X = A$, então X é uma subfórmula de A;

S_2: Se X é uma subfórmula de Y e Y é uma subfórmula de A, então X é uma subfórmula de A.

Um *seqüente* é uma expressão da forma

$$A_1, A_2, \ldots, A_n \vdash B_1, B_2, \ldots, B_m$$

onde todo A_i e todo B_j são fórmulas. O seqüente exprime o fato de que, se todas as fórmulas $A_1, \ldots, A_n$ são verdadeiras, então ao menos uma das fórmulas dentre $B_1, \ldots, B_m$ é verdadeira.

A lista de fórmulas à esquerda do símbolo $\vdash$ é chamada de *antecedente* do seqüente, e a lista à direita é chamada de *conseqüente* do seqüente.

Para a lógica proposicional clássica, podemos considerar que as listas $A_1, A_2, \ldots, A_n$ e $B_1, B_2, \ldots, B_m$ são conjuntos de fórmulas.[1]

Uma *fórmula assinalada* é uma expressão da forma TA ou FA, onde $A \in \mathcal{L}$ e T e F são os *sinais* possíveis de uma fórmula assinalada, respectivamente *verdadeiro* e *falso*.

O *conjugado* de uma fórmula assinalada é o resultado da operação de inversão do sinal, ou seja, para uma fórmula A, TA é o conjugado de FA e vice-versa.

As *subfórmulas* de uma fórmula assinalada SA $(S = T, F)$ são todas as fórmulas da forma TB ou FB, onde B é subfórmula de A.

1.2 Semântica

Um *valor-verdade* é um elemento do conjunto $V = \{0, 1\}$. Os elementos 0 e 1 são chamados, respectivamente, de *falso* e *verdadeiro*.

Uma *valoração* é uma função $v : \Gamma \to V$. Deste modo, dizemos que $A \in \Gamma$ é *verdadeira* sob v se $v(A) = 1$, e *falsa* sob v caso contrário.

Uma valoração $v : \mathcal{L} \to V$ é uma *valoração booleana* se, para toda fórmula $A, B \in \mathcal{L}$:

B_1: $v(\neg A) = 1$ sse $v(A) = 0$;

B_2: $v(A \wedge B) = 1$ sse $v(A) = 1$ e $v(B) = 1$;

B_3: $v(A \vee B) = 0$ sse $v(A) = 0$ e $v(B) = 0$;

B_4: $v(A \to B) = 0$ sse $v(A) = 1$ e $v(B) = 0$.

[1] Isto não é verdade para outras lógicas, como a lógica relevante ou a lógica linear, onde o número de ocorrências das fórmulas deve ser levado em consideração. Nesses casos, as listas de fórmulas podem ser representadas por *multi-conjuntos*, ou seja, conjuntos onde cada elemento tem uma multiplicidade.

A valoração pode ser estendida para fórmulas assinaladas se acrescentarmos as seguintes condições:

B_5: $v(TA) = 1$ sse $v(A) = 1$;

B_6: $v(FA) = 1$ sse $v(A) = 0$.

Uma fórmula A é uma *tautologia* (denotamos $\models A$) se A é verdadeira sob todas as valorações booleanas.

Uma valoração v satisfaz um conjunto de fórmulas Γ se toda fórmula de Γ é verdadeira sob v.

Um conjunto de fórmulas Γ *implica logicamente* uma fórmula A (denotamos $\Gamma \models A$) se A é verdadeira sob toda valoração booleana que satisfaça Γ.

Considere a fórmula A. A *polaridade* de uma subfórmula B de A é uma relação binária $Pol : \mathcal{L} \times \mathcal{L} \to \{-, +\}$.

Denotamos $Pol(B, A) = +$ quando a polaridade de B é $+$ em A, e dizemos que B tem polaridade *positiva* em A. Denotamos $Pol(B, A) = -$ quando a polaridade de B é $-$ em A, e dizemos que B tem polaridade *negativa* em A[2].

Podemos definir a polaridade de todas as subfórmulas de uma fórmula A indutivamente a partir das seguintes regras (B, C e D são subfórmulas de A):

P_1: $Pol(A, A) = +$;

P_{2a}: Se $Pol(B, A) = +$ e $B = \neg C$ então $Pol(C, A) = -$;

P_{2b}: Se $Pol(B, A) = -$ e $B = \neg C$ então $Pol(C, A) = +$;

P_{3a}: Se $Pol(B, A) = +$ e $B = C \vee D$ então $Pol(C, A) = Pol(D, A) = +$;

P_{3b}: Se $Pol(B, A) = -$ e $B = C \vee D$ então $Pol(C, A) = Pol(D, A) = -$;

P_{4a}: Se $Pol(B, A) = +$ e $B = C \wedge D$ então $Pol(C, A) = Pol(D, A) = +$;

P_{4b}: Se $Pol(B, A) = -$ e $B = C \wedge D$ então $Pol(C, A) = Pol(D, A) = -$;

P_{5a}: Se $Pol(B, A) = +$ e $B = C \to D$ então $Pol(C, A) = -$ e $Pol(D, A) = +$;

[2]Por simplicidade, se for possível omitir o contexto da fórmula A, denotamos simplesmente B^+ quando a polaridade de B é positiva em A, e B^- quando a polaridade de B é negativa em A.

P_{5b}: Se $Pol(B, A) = -$ e $B = C \to D$ então $Pol(C, A) = +$ e $Pol(D, A) = -$.

A polaridade pode ser estendida a fórmulas assinaladas, se substituirmos a regra P_1 acima por:

P_{1a}: Se $A = TB$, então $Pol(B, A) = +$;

P_{1b}: Se $A = FB$, então $Pol(B, A) = -$.

Capítulo 2

Métodos de Tableaux

Neste capítulo apresentaremos dois métodos de resolução baseados em tableaux: o método de tableaux analíticos de Smullyan, e o método de tableaux KE de D'Agostino.

2.1 Tableaux Analíticos

O *tableau analítico* é um método de prova consolidado por Smullyan [Smu68] baseado em árvores de *fórmulas assinaladas*. A idéia básica do método é tentar provar o seqüente

$$A_1, A_2, \ldots, A_n \vdash B_1, B_2, \ldots, B_m$$

através da expansão da árvore associada.

O método consiste em, dado um *tableau inicial* para o seqüente, estendê-lo através das *regras de expansão*, até que se chegue num estado onde se possa decidir a validade do seqüente.

O tableau inicial para o seqüente acima é a árvore de um ramo onde cada nó é uma das fórmulas assinaladas TA_1, TA_2, ..., TA_n, FB_1, FB_2, ..., FB_m.

As regras de expansão do tableau estendem o tableau inicial. As regras de expansão de tableaux analíticos são apresentadas na Tabela 2.1.

Para cada regra de expansão, as fórmulas que aparecem acima da linha horizontal são as *premissas* da regra. As fórmulas que aparecem abaixo da linha horizontal são as *conclusões* da regra.

$\frac{T\neg A}{FA}\ (T\neg)$	$\frac{F\neg A}{TA}\ (F\neg)$
$\frac{T(A\wedge B)}{\begin{array}{l}TA\\TB\end{array}}\ (T\wedge)$	$\frac{F(A\wedge B)}{FA\|FB}\ (F\wedge)$
$\frac{T(A\vee B)}{TA\|TB}\ (T\vee)$	$\frac{F(A\vee B)}{\begin{array}{l}FA\\FB\end{array}}\ (F\vee)$
$\frac{T(A\to B)}{FA\|TB}\ (T\to)$	$\frac{F(A\to B)}{\begin{array}{l}TA\\FB\end{array}}\ (F\to)$

Tabela 2.1: Regras de expansão de tableaux analíticos

Sejam X, Y, Z fórmulas assinaladas. As regras da forma

$$\frac{X}{Y} \qquad \frac{X}{\begin{array}{l}Y\\Z\end{array}}$$

são denominadas *regras do tipo* α .

As regras da forma

$$\frac{X}{Y|Z}$$

são denominadas *regras do tipo* β .

Uma regra pode ser aplicada a um ramo do tableau se este ramo contiver a premissa da regra (um ramo do tableau é um caminho da raiz até uma folha).

A aplicação de uma regra do tipo α a um ramo do tableau consiste em adicionar as conclusões da regra a este ramo.

A aplicação de uma regra do tipo β a um ramo do tableau consiste em *bifurcar* o ramo e, a cada ramo resultante, adicionar a conclusão correspondente.

Se um ramo do tableau contiver duas fórmulas conjugadas (ou seja, se contiver TA e FA para alguma fórmula A), dizemos que o ramo está *fechado*.

Se todos os ramos do tableau estiverem fechados, dizemos que o tableau está fechado, e neste caso, $A_1, A_2, \ldots, A_n \vdash B$.

Se, por outro lado, o tableau não estiver fechado (estiver *aberto*), e não houver nenhuma regra aplicável, então $A_1, A_2, \ldots, A_n \nvdash B$.

Exemplo 2.1 O tableau da Figura 2.1 foi obtido a partir do tableau inicial (composto apenas da fórmula 1) através de aplicações sucessivas das regras de expansão. As fórmulas 2 e 3 foram obtidas de 1 através da regra $(F \rightarrow)$. As fórmulas 4 e 5 foram obtidas de 2 através da regra $(T\wedge)$. As fórmulas 6 e 7 foram obtidas de 3 através da regra $(F \rightarrow)$. As fórmulas 8 e 9 foram obtidas de 4 através da regra $(T \rightarrow)$. As fórmulas 10 e 11 foram obtidas de 5 através da regra $(T \rightarrow)$. O símbolo $\times$ indica que o ramo correspondente do tableau está fechado. Deste modo, o ramo que termina com a fórmula 8 está fechado, porque contém as fórmulas conjugadas 6 e 8. O ramo que termina com a fórmula 10 está fechado, porque contém as fórmulas conjugadas 9 e 10. O ramo que termina com a fórmula 11 está fechado, porque contém as fórmulas conjugadas 7 e 11. Logo, o tableau está fechado, e portanto,

$$\vdash ((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$$

□

Em [Smu68], Smullyan provou a correção e a completude do método de tableaux analíticos, ou seja, se $\vdash$ representa a relação de conseqüência estabelecida pelo método de tableaux analíticos, e $\models$ a relação de conseqüência lógica (semântica), então

$$A_1, A_2, \ldots, A_n \vdash B \text{ se e somente se } A_1, A_2, \ldots, A_n \models B$$

2.2 Tableaux KE

D'Agostino propôs em [D'A90] o método de Tableaux KE como uma melhoria do método de tableaux analíticos de Smullyan, no sentido computacional. Tableaux KE *p-simulam* tableaux analíticos, mas tableaux analíticos não p-simulam tableaux KE [1].

Os tableaux KE são semelhantes aos tableaux analíticos, pois também utilizam fórmulas assinaladas, apresentam regras de expansão com premissas e conclusões, e

[1] Em outras palavras, para toda prova em tableau analítico de tamanho n, existe uma prova em tableau KE com tamanho polinomial em n, mas existe pelo menos uma prova em tableau KE de tamanho n cuja prova correspondente em tableau analítico tem tamanho superpolinomial em n.

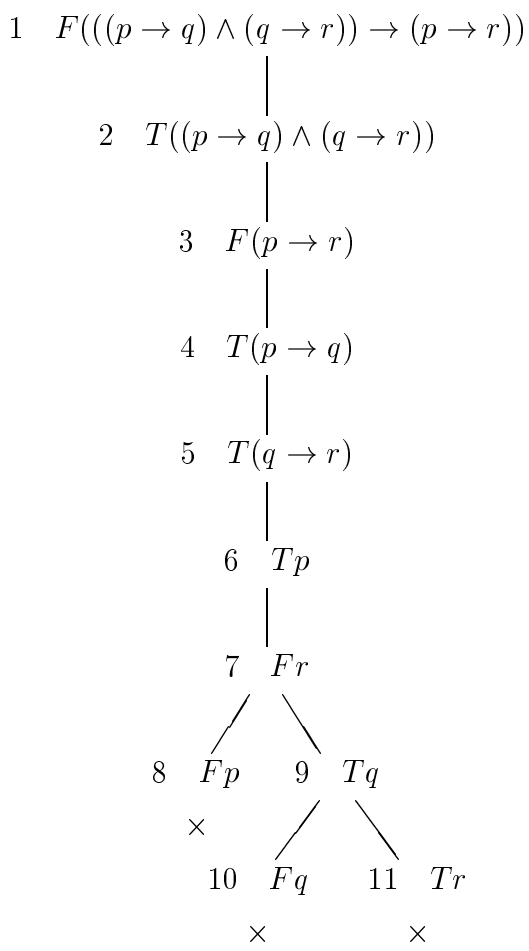


Figura 2.1: Tableau analítico para $\vdash ((p \to q) \wedge (q \to r)) \to (p \to r)$.

têm as mesmas regras para a construção do tableau inicial e para o fechamento do tableau.

As regras de expansão do tableau KE são apresentadas na Tabela 2.2.

$\frac{T\neg A}{FA}$	$(T\neg)$	$\frac{F\neg A}{TA}$	$(F\neg)$		
$\frac{T(A \wedge B)}{\begin{array}{c}TA\\TB\end{array}}$	$(T\wedge)$	$\frac{F(A \wedge B)}{TA \mid FB}$	$(F\wedge_1)$	$\frac{F(A \wedge B)}{TB \mid FA}$	$(F\wedge_2)$
$\frac{F(A \vee B)}{\begin{array}{c}FA\\FB\end{array}}$	$(F\vee)$	$\frac{T(A \vee B)}{FA \mid TB}$	$(T\vee_1)$	$\frac{T(A \vee B)}{FB \mid TA}$	$(T\vee_2)$
$\frac{F(A \to B)}{\begin{array}{c}TA\\FB\end{array}}$	$(F\to)$	$\frac{T(A \to B)}{TA \mid TB}$	$(T\to_1)$	$\frac{T(A \to B)}{FB \mid FA}$	$(T\to_2)$
$\frac{}{TA \mid FA}$	(PB)				

Tabela 2.2: Regras de expansão de tableaux KE

Os conectivos lógicos $\wedge, \vee$ e $\to$ têm três regras de expansão: uma com uma premissa e duas conclusões, e duas com duas premissas e uma conclusão. Nas regras com duas premissas, a primeira premissa é a *premissa primária* e a segunda é a *premissa secundária* [2]. O conectivo unário $\neg$ tem duas regras com uma premissa.

A regra (PB) permite que, a qualquer momento, o tableau seja bifurcado em ramos com as duas fórmulas conjugadas TA e FA. Isto corresponde ao *Princípio da Bivalência*, ou seja, o fato de uma fórmula poder assumir apenas um dos valores verdade: 0 (F) ou 1 (T). Note que esta é a única regra que permite a bifurcação de ramos do tableau.

O uso da regra (PB) no sistema KE deve se restringir a *aplicações analíticas*, ou seja, a fórmula A sobre a qual a regra se aplica deve servir como premissa secundária de alguma fórmula β não analisada. Isto garante a propriedade (também presente no tableau analítico) de que toda regra de expansão adiciona uma subfórmula de

[2] Por simplicidade, chamaremos as fórmulas que podem ser premissas de alguma regra com uma premissa de *fórmulas* α, e as fórmulas que podem ser premissas primárias de alguma regra com duas premissas de *fórmulas* β.

alguma fórmula mais complexa ocorrente no tableau.[3]

Exemplo 2.2 Vamos refazer o Exemplo 2.1 usando o método de tableaux KE. O tableau da Figura 2.2 foi obtido a partir do tableau inicial (composto apenas da fórmula 1) através de aplicações sucessivas das regras de expansão. As fórmulas 2 e 3 foram obtidas de 1 através da regra $(F \rightarrow)$. As fórmulas 4 e 5 foram obtidas de 2 através da regra $(T\wedge)$. As fórmulas 6 e 7 foram obtidas de 3 através da regra $(F \rightarrow)$. A fórmula 8 foi obtida usando-se a regra $(T \rightarrow_1)$ com a fórmula 4 como premissa primária e a fórmula 6 como premissa secundária. A fórmula 9 foi obtida usando-se a regra $(T \rightarrow_1)$ com a fórmula 5 como premissa primária e a fórmula 8 como premissa secundária. Neste ponto, o tableau está fechado, pois seu único ramo contém as fórmulas conjugadas 7 e 9. Portanto,

$$\vdash ((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$$

Note que, no Exemplo 2.1, foram geradas 11 fórmulas e houve duas bifurcações. Neste caso, foram geradas 9 fórmulas e o tableau fechou sem bifurcar nenhuma vez. □

Exemplo 2.3 Neste exemplo, queremos provar a inconsistência do conjunto $\{\neg p \rightarrow q, \neg p \rightarrow \neg q, p \rightarrow r, p \rightarrow \neg r\}$. Para isso construímos o tableau da Figura 2.3. Note que é impossível fechar o tableau sem o uso da regra (PB), pois a partir do tableau inicial não há nenhuma regra α ou β que possa ser utilizada.

[D'A90] apresenta a prova de correção e completude do método de tableaux KE.

[3] No sistema KE proposto por D'Agostino esta restrição da aplicação da regra (PB) está presente para garantir a propriedade da subfórmula e manter o caráter analítico do sistema de tableau. Num sistema mais genérico, ao invés de ser uma obrigação, esta restrição pode constituir apenas uma *heurística de bifurcação*. É correto e possível bifurcar o tableau sobre fórmulas que não geram premissas secundárias.

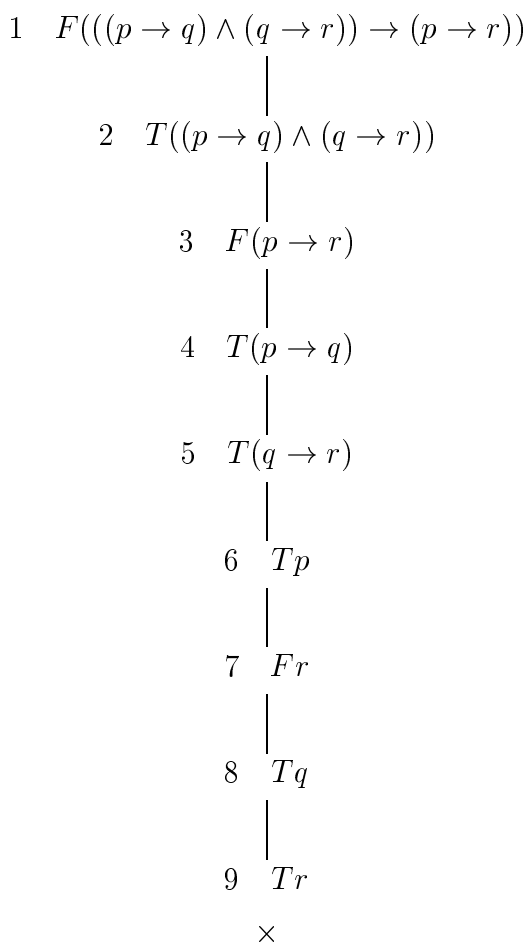


Figura 2.2: Tableau KE para $\vdash ((p \to q) \wedge (q \to r)) \to (p \to r)$.

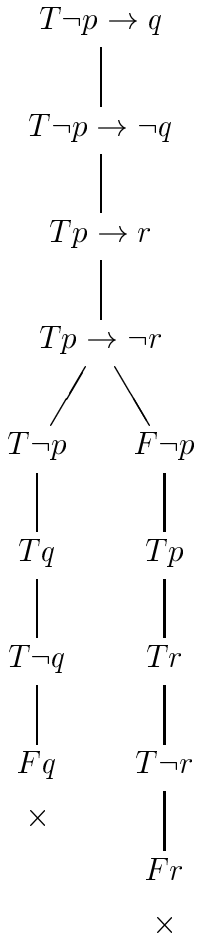


Figura 2.3: Tableau KE para verificar a inconsistência do conjunto de fórmulas $\{\neg p \to q, \neg p \to \neg q, p \to r, p \to \neg r\}$.

Capítulo 3

Raciocínio por Aproximações

Neste capítulo, apresentaremos o método de raciocínio por aproximações de Cadoli e Schaerf, e o método de tableaux KE para a lógica S_3 (ou tableaux KE-S_3).

3.1 O método de raciocínio por aproximações de Cadoli e Schaerf

Cadoli e Schaerf propuseram em [SC95] um método de *raciocínio por aproximações*. Esse método aproxima a lógica clássica por duas famílias de lógicas: a família S_1, que serve como um limitante superior, e a família S_3, que serve como um limitante inferior da lógica clássica. O método se restringe a fórmulas na forma clausal (ou seja, conjunções de disjunções de literais).

Ambas as famílias de lógicas S_1 e S_3 são baseadas em um *conjunto-contexto* S, que é um conjunto de átomos. Os elementos de S são os elementos que se comportam classicamente na lógica. Deste modo, as lógicas S_1 e S_3 se aproximam da lógica clássica conforme o conjunto S aumenta.

Uma valoração v em S_1 é definida como se segue:

- $v(p) \neq v(\neg p)$, se $p \in S$;
- $v(p) = v(\neg p) = 0$, se $p \notin S$.

Para um conjunto de fórmulas Γ e uma fórmula A, dizemos que Γ implica logicamente A em S_1 (e escrevemos $\Gamma \models^1_S A$) se toda valoração em S_1 que satisfaça Γ também satisfaz A.

Pode-se provar que $\models^1_S$ é completa e incorreta com relação a $\models$. Ou seja, para todo Γ e para todo A,

$$\Gamma \not\models^1_S A \Rightarrow \Gamma \not\models A$$

Uma valoração v em S_3 é definida como se segue:

- $v(p) \neq v(\neg p)$, se $p \in S$;
- $v(p) \neq v(\neg p)$ ou $v(p) = v(\neg p) = 1$, se $p \notin S$.

Para um conjunto de fórmulas Γ e uma fórmula A, dizemos que Γ implica logicamente A em S_3 (e escrevemos $\Gamma \models^3_S A$) se toda valoração em S_3 que satisfaça Γ também satisfaz A.

Pode-se provar que $\models^3_S$ é correta e incompleta com relação a $\models$. Ou seja, para todo Γ e para todo A,

$$\Gamma \models^3_S A \Rightarrow \Gamma \models A$$

Note que, se $S = \mathcal{P}$, então $v(p) \neq v(\neg p)$ para todo $p \in \mathcal{P}$, ou seja, tanto S_1 como S_3 coincidem com a lógica clássica.

O raciocínio por aproximações de Cadoli e Schaerf baseia-se em simplificações dos conjuntos de fórmulas, que utilizam as propriedades das lógicas S_1 e S_3 descritas acima.

Lema 3.1 (Cadoli & Schaerf [SC95]) *Seja simplifica-1*(Γ, S) *o resultado de apagar todos os literais do conjunto de fórmulas* Γ *que mencionam átomos fora do conjunto* S. *O conjunto* Γ *é* S_1*-satisfatível sse simplifica-1*(Γ, S) *é classicamente satisfatível.*

Teorema 3.2 (Cadoli & Schaerf [SC95]) *Seja* $A = A_S \vee A_{\overline{S}}$, *onde os átomos de* A_S *estão todos em* S *e nenhum dos átomos de* $A_{\overline{S}}$ *está em* S. *O conjunto* $\Gamma \models^1_S A$ *sse* $\Gamma \cup \{\neg A_S\}$ *não é* S_1*-satisfatível.* [1]

[1] Isto só pode ser feito porque A_S se comporta classicamente, e assim podemos calcular sua negação em forma clausal.

Lema 3.3 (Cadoli & Schaerf [SC95]) *Seja simplifica-3*(Γ, S) *o resultado de apagar todas as cláusulas de* Γ *que contêm pelo menos um átomo fora de* S*. O conjunto* Γ *é* S_3*-satisfatível sse simplifica-3*(Γ, S) *é classicamente satisfatível.*

Teorema 3.4 (Cadoli & Schaerf [SC95]) *Seja* A *tal que os átomos de* A *estejam todos em* S*. O conjunto* $\Gamma \models^3_S A$ *sse* $\Gamma \cup \{\neg A\}$ *não é* S_3*-satisfatível.*

Um algoritmo para decidir se $\Gamma \models A$ baseado no raciocínio por aproximações pode ser feito da seguinte maneira: começando com S igual ao conjunto dos átomos de A e passo a passo adicionando átomos a S, e efetuando as simplificações *simplifica-1* e *simplifica-3*, parando com o menor S tal que $\Gamma \not\models^1_S A$ ou $\Gamma \models^3_S A$.

Um tal algoritmo apresenta certas desvantagens:

- O método é restrito a fórmulas na forma clausal, devido às simplificações *simplifica-1* e *simplifica-3*;
- Não se tem uma heurística para determinar qual átomo será adicionado ao conjunto S a cada passo.

Exemplo 3.5 Seja $\Gamma = \{\neg p \vee q, \neg q \vee r\}$ e $A = \neg p \vee r$. Queremos decidir se $\Gamma \models A$.

1. Faça $S \leftarrow \{p, r\}$. Temos:
   - *simplifica-1*$(\Gamma \cup \neg A_S, S) = \{\neg p, r, p, \neg r\}$, e portanto $\Gamma \cup \neg A_S$ não é S_1-satisfatível, ou seja, $\Gamma \models^1_S A$.
   - *simplifica-3*$(\Gamma \cup \neg A, S) = \{p, \neg r\}$, e portanto $\Gamma \cup \neg A$ é S_3-satisfatível, ou seja, $\Gamma \not\models^3_S A$.
2. Faça $S \leftarrow S \cup \{q\}$. Temos:
   - *simplifica-1*$(\Gamma \cup \neg A_S, S) = \{\neg p \vee q, \neg q \vee r, p, \neg r\}$, e portanto $\Gamma \cup \neg A_S$ não é S_1-satisfatível, ou seja, $\Gamma \models^1_S A$.
   - *simplifica-3*$(\Gamma \cup \neg A, S) = \{\neg p \vee q, \neg q \vee r, p, \neg r\}$, e portanto $\Gamma \cup \neg A$ não é S_3-satisfatível, ou seja, $\Gamma \models^3_S A$, donde concluímos que $\Gamma \models A$. □

3.2 Tableaux KE-S_3

Finger e Wassermann propuseram em [FW01] o método de tableaux KE-S_3 como uma generalização do método de Cadoli e Schaerf para raciocínio por aproximações.

A fim de eliminar a restrição a cláusulas presente no método de Cadoli e Schaerf, eles apresentam uma semântica $S_{3.2}$ para o fragmento completo de S_3.[2]

A semântica $S_{3.2}$ se baseia em uma valoração proposicional:

Definição 3.6 (Finger & Wassermann [FW01]) *Uma $S_{3.2}$-valoração $v_S^{3.2}$ é uma função $v_S^{3.2} : \mathcal{L} \to \{0,1\}$ que estende uma valoração proposicional v_p (ou seja, $v_S^{3.2}(p) = v_p(p)$), satisfazendo as seguintes condições:*

(i)	$v_S^{3.2}(\alpha \wedge \beta) = 1$	$\Leftrightarrow$	$v_S^{3.2}(\alpha) = v_S^{3.2}(\beta) = 1$
(ii)	$v_S^{3.2}(\alpha \vee \beta) = 0$	$\Leftrightarrow$	$v_S^{3.2}(\alpha) = v_S^{3.2}(\beta) = 0$
(iii)	$v_S^{3.2}(\alpha \to \beta) = 0$	$\Leftrightarrow$	$v_S^{3.2}(\alpha) = 1$ *e* $v_S^{3.2}(\beta) = 0$
(iv)	$v_S^{3.2}(\neg\alpha) = 0$	$\Rightarrow$	$v_S^{3.2}(\alpha) = 1$
(v)	$v_S^{3.2}(\neg\alpha) = 1, \alpha \in S$	$\Rightarrow$	$v_S^{3.2}(\alpha) = 0$

As condições (iv) e (v) exprimem o fato de que, se $\alpha \in S$, então α deve se comportar classicamente, enquanto que se $\alpha \notin S$, então α pode se comportar classicamente ou paraconsistentemente, ou seja, α e $\neg\alpha$ podem ambos assumir o valor 1.

As regras de expansão do tableau KE-S_3 são apresentadas na Tabela 3.1.

Note que a antiga regra ($T\neg$) foi substituída por:

$$\frac{T\neg A}{FA} \quad \text{se } \textit{átomos(A)} \subseteq S$$

Esta regra só poderá ser aplicada a um ramo se este ramo contiver o antecedente da regra *e a condição átomos(A)* $\subseteq S$ *for satisfeita.*

A regra ($T\neg$) é uma restrição da regra clássica, que torna o sistema KE-S_3 subclássico. De fato, todo tableau que fecha em KE-S_3 também fecha na lógica clássica.

Observemos agora como uma heurística de aproximação é obtida no método de tableaux KE-S_3. O algoritmo é apresentado a seguir:

1. $S \leftarrow \emptyset$.

2. Transformar o seqüente de entrada em um tableau KE-S_3 inicial.

[2] O nome $S_{3.2}$ se deve ao fato de ser uma semântica de dois valores para S_3.

$\frac{T\neg A}{FA}$	se *átomos(A)* $\subseteq S$		$(T\neg)$	$\frac{F\neg A}{TA}$	$(F\neg)$
$\frac{T(A \wedge B)}{\begin{array}{l}TA\\TB\end{array}}$	$(T\wedge)$	$\frac{F(A \wedge B)}{\begin{array}{l}TA\\FB\end{array}}$	$(F\wedge_1)$	$\frac{F(A \wedge B)}{\begin{array}{l}TB\\FA\end{array}}$	$(F\wedge_2)$
$\frac{F(A \vee B)}{\begin{array}{l}FA\\FB\end{array}}$	$(F\vee)$	$\frac{T(A \vee B)}{\begin{array}{l}FA\\TB\end{array}}$	$(T\vee_1)$	$\frac{T(A \vee B)}{\begin{array}{l}FB\\TA\end{array}}$	$(T\vee_2)$
$\frac{F(A \to B)}{\begin{array}{l}TA\\FB\end{array}}$	$(F\to)$	$\frac{T(A \to B)}{\begin{array}{l}TA\\TB\end{array}}$	$(T\to_1)$	$\frac{T(A \to B)}{\begin{array}{l}FB\\FA\end{array}}$	$(T\to_2)$
$\frac{}{TA \mid FA}$	(PB)				

Tabela 3.1: Regras de expansão de tableaux KE-S_3

3. Expandir o tableau até que ele feche ou fique bloqueado devido à impossibilidade de aplicação de regras.[3]

4. Se o tableau estiver fechado, terminar com sucesso.

5. Se o tableau contiver um ramo que não pode ser expandido classicamente, terminar com falha.

6. Se o tableau estiver bloqueado devido à fórmula $T\neg\alpha$, fazer $S \leftarrow S \cup$ átomos(α) e voltar ao passo 3.

Ao adicionarmos átomos ao conjunto-contexto S, o que estamos fazendo é uma mudança da lógica com a qual estamos trabalhando. A cada passo de adição ao conjunto S estamos cada vez mais próximos da lógica clássica. A lógica clássica é atingida quando todos os átomos estão em S.

A construção do tableau nos diz quais os próximos átomos a serem adicionados a S, dando imediatamente uma heurística que diz qual o próximo átomo a ser incluído no conjunto-contexto S.

[3]Se o seqüente a ser provado é válido, então o tableau só pode ficar bloqueado devido à impossibilidade de aplicação da regra $(T\neg)$. Em outras palavras, o tableau só pode ficar bloqueado se, para a fórmula $T\neg\alpha$ sobre a qual a regra seria aplicada, tivermos *átomos*$(\alpha) \not\subseteq S$.

Exemplo 3.7 Vamos decidir se $\Gamma \vdash A$, onde Γ e A são os mesmos do exemplo 3.5. O tableau correspondente a essa prova é o da Figura 3.1. O tableau inicial é composto pelas três primeiras fórmulas, e o conjunto S é inicializado com vazio.

As fórmulas (4) e (5) são obtidas de (3) usando a regra $(F\vee)$. A fórmula (6) é obtida de (4) usando a regra $(F\neg)$. A fórmula (7) é obtida de (1) e (4) usando a regra $(T\vee_1)$. A fórmula (8) é obtida de (2) e (5) usando a regra $(T\vee_2)$. A fórmula (9) é obtida de (8) usando a regra $(T\neg)$ e fazendo $S \leftarrow S \cup \{q\}$.

Pela figura, vemos que o tableau fecha com $S = \{q\}$. □

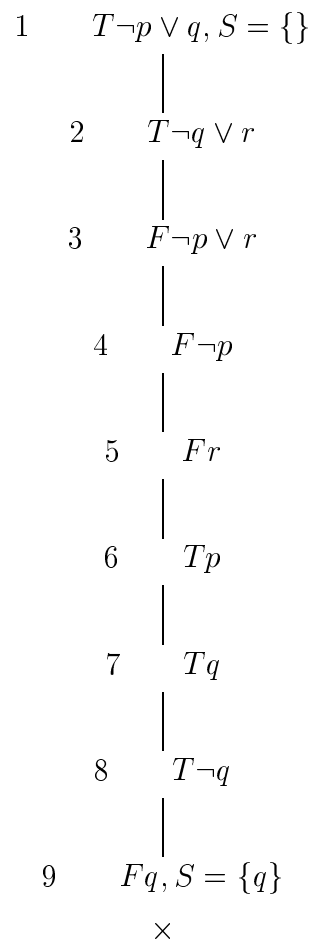


Figura 3.1: Tableau KE-S_3 para $\neg p \vee q, \neg q \vee r \vdash \neg p \vee r$.

Em [FW01] são provados os teoremas de correção e completude do método de tableaux KE-S_3 com relação à semântica $S_{3.2}$.

Capítulo 4

Implementação

4.1 Arcabouços orientados a objetos

Um *arcabouço orientado a objetos* (do inglês, *object-oriented framework*) é o projeto reutilizável de um sistema ou subsistema, implementado através de um conjunto de classes concretas e abstratas e suas colaborações [JF88, BJ94, FHLS98].

Em outras palavras, o arcabouço deve dar uma solução genérica para um conjunto de problemas similares em um determinado domínio de aplicação. Deste modo, ele não deve implementar as especificidades de cada problema, mas apenas as características comuns. Para permitir a implementação das funcionalidades específicas de cada problema, o arcabouço deve fornecer *ganchos* (do inglês *hooks*) onde se possa adicionar novos componentes (ou colaborações).

A Figura 4.1 mostra um esquema de um arcabouço. O retângulo A representa o arcabouço. $G1, G2, \ldots, Gn$ são os ganchos onde os novos componentes com as funcionalidades específicas $FE1, FE2, \ldots, FEn$ são adicionados.

Ao ato de adicionarmos funcionalidades específicas ao arcabouço damos o nome de *instanciação do arcabouço*.

No nível de programação, o esquema de arcabouço orientado a objetos pode ser implementado em uma linguagem de programação orientada a objetos (como C++, Java ou SmallTalk) através do uso de *classes abstratas* e *derivação*.

Uma classe abstrata é uma classe cujos métodos podem não ter implementação. Sob esse aspecto, pode-se considerar que a classe abstrata é incompleta.

O arcabouço pode então ser representado por uma ou mais classes abstratas. Os

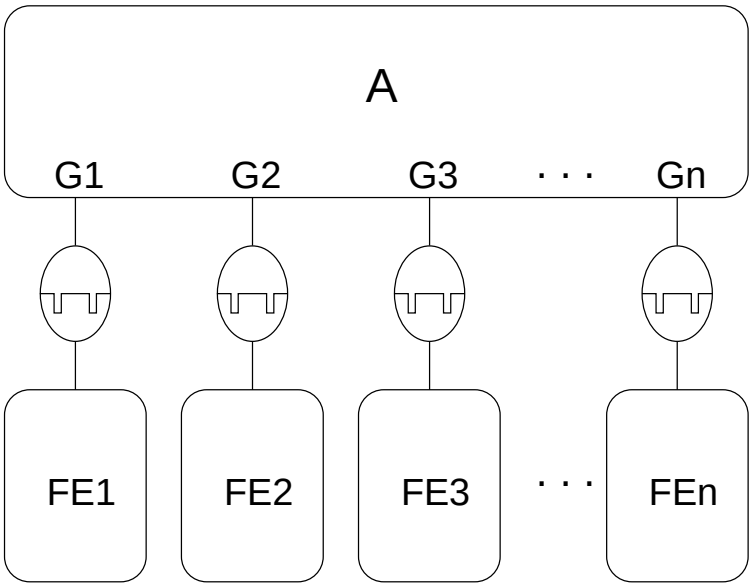


Figura 4.1: Um esquema de um arcabouço.

ganchos do arcabouço são os métodos sem implementação.

Uma *classe derivada* é uma classe que herda os membros e métodos de uma outra classe (chamada *classe primitiva*). Uma classe derivada de uma classe abstrata será abstrata se não der implementação para todos os métodos abstratos da classe primitiva, e concreta, caso contrário.

A instanciação do arcabouço, no nível da implementação, consiste em derivar as classes abstratas que constituem o arcabouço, e dar implementação a todos os ganchos de modo a completar todas as funcionalidades e transformar o arcabouço em uma aplicação completa.

A Figura 4.2 mostra um exemplo da implementação de um arcabouço. A classe A representa o arcabouço. Os métodos $FE1()$, $FE2()$, ..., $FEn()$ representam os ganchos do arcabouço. As implementações desses métodos na classe derivada D representam os componentes com novas funcionalidades específicas do arcabouço.

O uso do arcabouço orientado a objetos como modelo de desenvolvimento apresenta vantagens e desvantagens [FHLS98].

```
class A {                          class D : class A {
 abstract method FE1();             method FE1() {...}
 abstract method FE2();             method FE2() {...}
 abstract method FE3();             method FE3() {...}
 ...                                ...
 abstract method FEn();             method FEn() {...}
}                                  }
```

Figura 4.2: Implementação de um arcabouço.

Vantagens

- **Reuso de experiência.** Quando usamos um arcabouço previamente construído, estamos reutilizando a experiência dos desenvolvedores do arcabouço, porque estes já analisaram o domínio da aplicação para construir um projeto de qualidade.
- **Tempo reduzido de desenvolvimento.** Como o domínio de aplicação já foi analizado pelos desenvolvedores do arcabouço e todas as decisões estruturais já foram tomadas no projeto do arcabouço, os usuários de um arcabouço podem desenvolver aplicações em menos tempo. Além disso, o arcabouço pode oferecer componentes prontos para o uso na aplicação.
- **Qualidade melhorada.** Se o projeto do arcabouço foi bem pensado e executado com qualidade, então as aplicações derivadas deste arcabouço herdarão essa qualidade de projeto.
- **Custo reduzido de manutenção.** Se várias aplicações são derivadas de um único arcabouço, então essas aplicações terão um mesmo padrão de projeto e serão mantidas mais facilmente.

Desvantagens

- **Incompatibilidade com o arcabouço.** Se os requisitos da aplicação são incompatíveis com o projeto do arcabouço, pode ser difícil implementá-la a partir deste. O mais grave é que a incompatibilidade pode ser encontrada muito tarde, durante o desenvolvimento da aplicação. Uma boa documentação sobre as capacidades do arcabouço pode ajudar a solucionar este problema. Projetos de protótipos também podem ajudar a familiarizar os usuários com as funcionalidades do arcabouço.

- **Tempo de aprendizado.** Usar um arcabouço requer um certo aprendizado. Se o arcabouço for muito complexo, pode ser necessário muito tempo para se aprender a utilizá-lo.
- **Perda do controle no projeto.** Como o arcabouço geralmente tem a priori um projeto especificado e implementado, as aplicações derivadas precisam ser implementadas de acordo com este projeto.

4.2 O arcabouço para métodos de tableaux

Neste projeto, foi feita a implementação em linguagem de programação C++ dos três métodos de tableaux descritos nos capítulos anteriores: o método analítico de Smullyan, o método KE de D'Agostino e o método para raciocínio por aproximações KE-S_3 de Finger e Wassermann. Para isso, foi criado um *arcabouço para métodos de tableau*.

O desenvolvimento do arcabouço e das aplicações derivadas deste (ou seja, cada um dos métodos de tableaux) foi feito concorrentemente, o que apresenta algumas vantagens:

- Ao se analisar todas as aplicações, pode-se ter uma melhor idéia dos requisitos do arcabouço, visto que se conhecerá as funcionalidades comuns e as funcionalidades específicas de cada uma delas.
- Como as aplicações já são conhecidas durante o projeto do arcabouço, as incompatibilidades entre o arcabouço e as aplicações são imediatamente detectadas e evitadas.
- Pela mesma razão, o tempo de aprendizado do arcabouço é diminuído, e não há perda de controle no projeto das aplicações.

A Figura 4.3 mostra a estrutura básica do projeto. Temos o arcabouço genérico *Tableau*, que implementa a funcionalidade genérica de um método de tableau. As aplicações derivadas deste arcabouço são *AnalyticTableau* e KETableau, que implementam, respectivamente, o método analítico e o método KE.

A aplicação *KETableau* é especial, porque é também um arcabouço. A aplicação *KES3Tableau*, que implementa o método KE-S_3, instancia o arcabouço *KETableau*.

Associado a cada um dos métodos de tableau, temos um módulo de estratégia, que é responsável pela implementação da heurística de resolução do tableau correspondente. Cada um dos módulos `Strategy` é também um arcabouço. Isto significa que para cada método de tableau, podemos ter várias heurísticas de resolução diferentes.

No Apêndice A, o uso do arcabouço é explicado com detalhes. O intuito do apêndice é fornecer informações para facilitar o uso do arcabouço para o desenvolvimento de novas aplicações.

4.3 As implementações

Como pode ser visto no Apêndice A, o método principal do arcabouço para métodos de tableau é o método `Tableau::close()`[1]. É neste método que toda a inteligência está embutida, na forma de heurísticas de escolha da próxima regra a ser aplicada no tableau a cada passo. Vamos então discutir as heurísticas que foram utilizadas nos métodos de tableau.

4.3.1 As heurísticas

A ordem de aplicação das regras

Primeiramente, observemos os dois tableaux analíticos abaixo, para verificação da validade do seqüente

$$a_1 \vee b_1, a_1 \rightarrow a_2 \vee b_2, b_1 \rightarrow a_2 \vee b_2 \vdash a_2 \vee b_2$$

[1] Algumas práticas já utilizadas na adaptação KE* do sistema KE introduzido por Endriss [End99] são adotadas nas implementações:

- A busca por fórmulas assinaladas complementares para o fechamento do tableau é restrita a fórmulas literais;
- A busca por fórmulas a serem usadas como premissas secundárias de regras com duas premissas é restrita a fórmulas literais, com uma exceção: diretamente após cada aplicação da regra PB, a próxima aplicação (óbvia) de uma regra de duas premissas é feita sem restrição a literais. Por exemplo, se PB é aplicada sobre A subfórmula de $TA \vee B$, então o ramo esquerdo será iniciado com TA e o ramo direito será iniciado com FA e TB, seja A literal ou não.

Além disso, uma melhoria é introduzida fazendo com que as fórmulas β não sejam analisadas até que todas as fórmulas α sejam analisadas.

O sistema KE* com as restrições acima é correto e completo. Além disso, é mais eficiente, pois os passos que mais exigem tempo em todo o procedimento de prova são os passos onde se faz busca em uma lista de fórmulas — ou seja, o fechamento do tableau e a aplicação de regras com duas premissas —, e as restrições descritas acima reduzem o espaço de busca nessas listas de fórmulas.

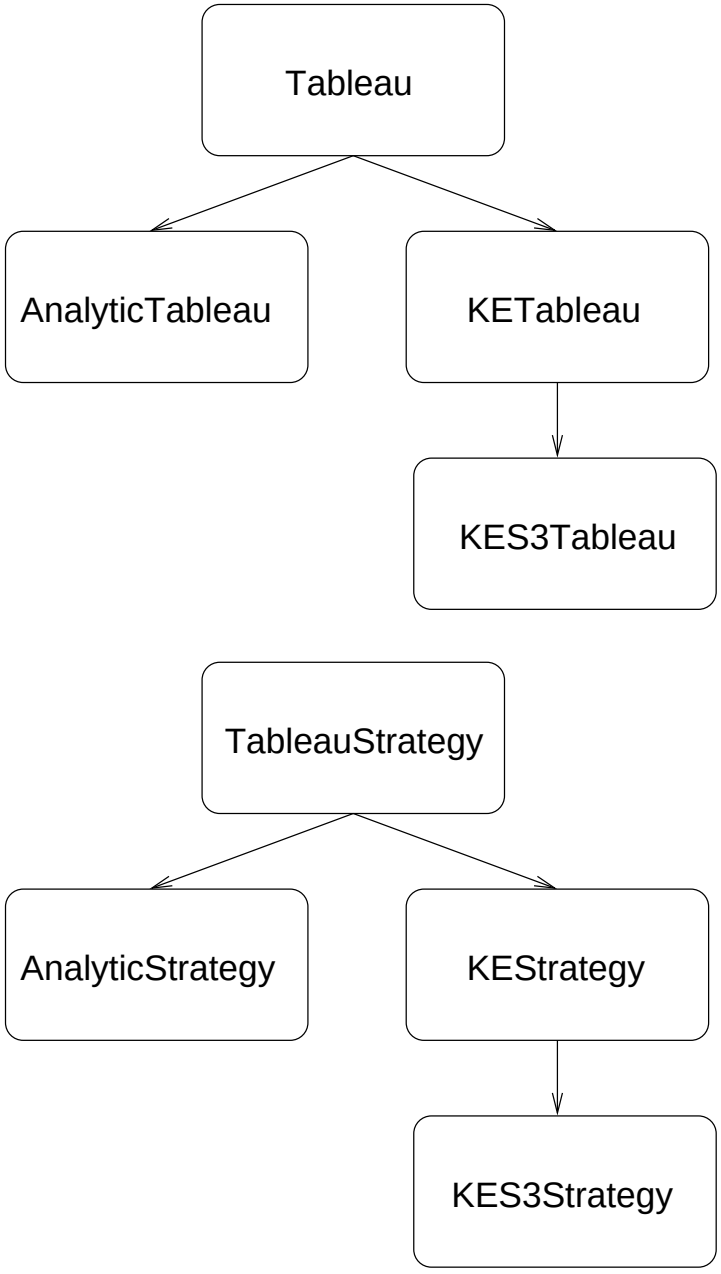


Figura 4.3: A estrutura básica do projeto.

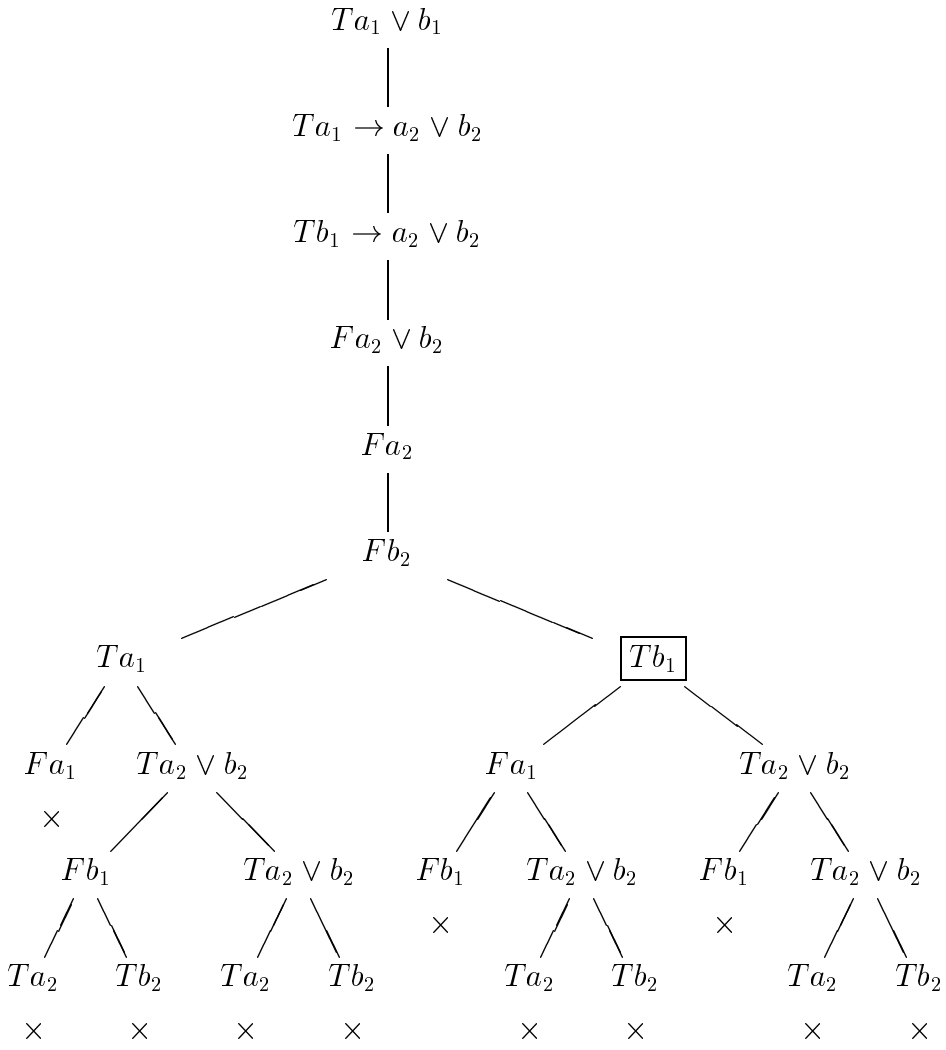


Figura 4.4: Tableau analítico para $a_1 \vee b_1, a_1 \to a_2 \vee b_2, b_1 \to a_2 \vee b_2 \vdash a_2 \vee b_2$.

No tableau da Figura 4.4, as regras são aplicadas na seguinte ordem de preferência:

D_1 primeira ocorrência de fórmula do tipo α ainda não aplicada;

D_2 primeira ocorrência de fórmula do tipo β ainda não aplicada.

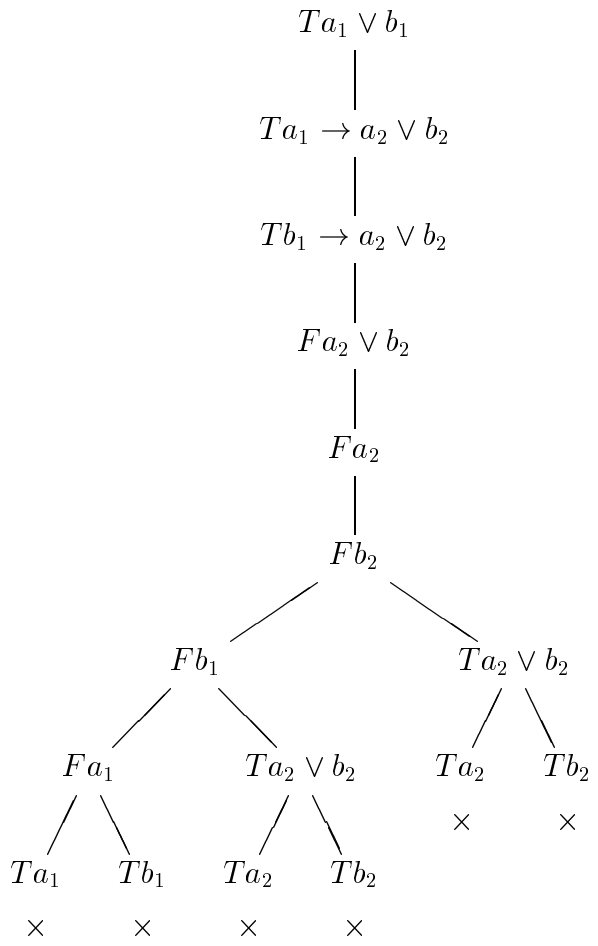


Figura 4.5: Tableau analítico para $a_1 \vee b_1, a_1 \to a_2 \vee b_2, b_1 \to a_2 \vee b_2 \vdash a_2 \vee b_2$.

No tableau da Figura 4.5, as regras são aplicadas na seguinte ordem de preferência:

U_1 primeira ocorrência de fórmula do tipo α ainda não aplicada;

U_2 última ocorrência de fórmula do tipo β ainda não aplicada.

No tableau da Figura 4.4, note que a aplicação da regra

$$\frac{Ta_1 \to a_2 \vee b_2}{Fa_1 | Ta_2 \vee b_2}$$

no ramo enraizado na fórmula destacada é redundante[2]. Neste ponto, a prova poderia ser encurtada se fosse aplicada a regra

$$\frac{Tb_1 \to a_2 \vee b_2}{Fb_1 | Ta_2 \vee b_2}$$

Agora, seja

$$\Gamma_n = \bigcup_{i=1}^{n} \{a_i \to a_{i+1} \vee b_{i+1}, b_i \to a_{i+1} \vee b_{i+1}\}$$

e considere o tableau analítico com as mesmas regras de preferência D_i e U_i para os seqüentes

$$a_1 \vee b_1, \Gamma_n \vdash a_{n+1} \vee b_{n+1}$$

Teremos como resultado a Tabela 4.1.[3]

	D_i		U_i	
n	# nós	# fórmulas	# nós	# fórmulas
1	21	26	11	16
2	125	132	19	26
3	789	798	27	36
4	4741	4752	35	46
5	27333	27346	43	56
6	153221	153236	51	66

Tabela 4.1: Número de nós gerados e número de fórmulas geradas na prova do seqüente $a_1 \vee b_1, \Gamma_n \vdash a_{n+1} \vee b_{n+1}$ pelo método de tableau analítico usando as regras de preferência D_i e U_i.

Neste exemplo, as diferenças nos números de nós e fórmulas gerados pelas duas heurísticas sugerem que uma escolha que leve em conta apenas a ordem em que as fórmulas aparecem no tableau não é uma escolha inteligente. Podemos facilmente "enganar" uma heurística deste tipo alterando a ordem das fórmulas do tableau. Portanto, outras propriedades das fórmulas precisam ser exploradas.[4]

[2] Em decorrência dessa redundância, ocorrem repetições de subtableaux. Repetições dessa natureza são responsáveis pelo caráter exponencial das provas de tableaux analítico.

[3] Os resultados desta tabela foram obtidos pela execução dos testes na implementação, utilizando as regras de preferência descritas.

[4] As duas heurísticas descritas a seguir, a heurística da valoração e a heurística da polaridade, são formas conhecidas de se atacar o problema de escolha da ordem de aplicação das regras do tableau, e portanto podem ser consideradas de domínio público.

Preferência através de valorações

No caso do método de tableaux KE, a aplicação de regras sobre fórmulas do tipo β não levam à bifurcação do tableau. Assim, as repetições de subtableaux poderão ser evitadas através de uma heurística sobre a regra PB, que é a única que leva à bifurcação do tableau.

No tableau da Figura 4.6, as regras são aplicadas na seguinte ordem de preferência:

K_1 primeira ocorrência de fórmula do tipo α ainda não aplicada;

K_2 primeira ocorrência de fórmula do tipo β ainda não aplicada, usando uma fórmula literal como secundária;

K_3 regra PB sobre subfórmula da primeira ocorrência de fórmula do tipo β ainda não aplicada.

Note que os ramos gerados após a segunda aplicação da regra PB não adicionam nenhuma informação nova ao tableau (ver as fórmulas destacadas), porque a regra foi aplicada sobre uma fórmula (a_1) sobre a qual a regra já havia sido aplicada neste ramo.

Semanticamente, como temos Fa_1 no ramo ($v(a_1) = 0$), e $Ta_1 \rightarrow a_2 \vee b_2$, então $v(a_1 \rightarrow a_2 \vee b_2) = 1$. Como queremos falsificar o tableau, ou seja, achar uma valoração que falsifique as fórmulas do tableau inicial, devemos olhar para as fórmulas A com $v(A) \neq 1$, onde v é a valoração parcial dada pelos literais do ramo corrente. Ou seja, ao fazermos essa restrição na aplicação da regra PB, estamos evitando aplicações redundantes como a do exemplo acima.

Dos comentários acima segue a

Heurística da valoração: aplicar PB sobre uma subfórmula (sobre a qual PB ainda não foi aplicada) de uma fórmula A do tipo β não analisada tal que $v(A) \neq 1$ segundo a valoração parcial dada pelas fórmulas literais do ramo corrente ($p \in \mathcal{P}$):

$$v(p) = \begin{cases} 1 & \text{se } Tp \in \text{ ramo} \\ 0 & \text{se } Fp \in \text{ ramo} \\ * & \text{caso contrário} \end{cases}$$

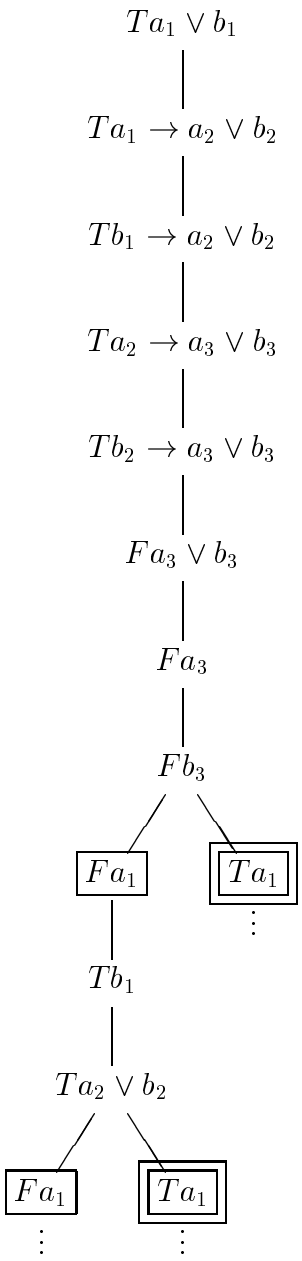


Figura 4.6: Tableau KE (incompleto) para $a_1 \vee b_1, \Gamma_2 \vdash a_3 \vee b_3$.

A Figura 4.7 mostra o tableau KE para o seqüente $a_1 \vee b_1, \Gamma_2 \vdash a_3 \vee b_3$ usando a heurística da valoração.

A regra PB no ramo destacado com caixa simples foi aplicada sobre a_1, subfórmula de $Ta_1 \vee b_1$, que é a primeira fórmula β não analisada com valor diferente de 1 segundo a valoração dada pelo ramo ($v(Ta_1 \vee b_1) = *$[5]).

A regra PB no ramo destacado com caixa dupla foi aplicada sobre b_1, subfórmula de $Tb_1 \rightarrow a_2 \vee b_2$, que é a primeira fórmula β não analisada com valor diferente de 1 segundo a valoração dada pelo ramo ($v(Tb_1 \rightarrow a_2 \vee b_2) = *$).

Preferência através da polaridade

Observe na Tabela 4.2 as regras com duas premissas do tableau KE anotadas com as polaridades.

$\dfrac{F(A^- \wedge B^-)}{\begin{array}{c}TA^+ \\ \hline FB^-\end{array}}$	$(F\wedge_1)$	$\dfrac{F(A^- \wedge B^-)}{\begin{array}{c}TB^+ \\ \hline FA^-\end{array}}$	$(F\wedge_2)$
$\dfrac{T(A^+ \vee B^+)}{\begin{array}{c}FA^- \\ \hline TB^+\end{array}}$	$(T\vee_1)$	$\dfrac{T(A^+ \vee B^+)}{\begin{array}{c}FB^- \\ \hline TA^+\end{array}}$	$(T\vee_2)$
$\dfrac{T(A^- \rightarrow B^+)}{\begin{array}{c}TA^+ \\ \hline TB^+\end{array}}$	$(T\rightarrow_1)$	$\dfrac{T(A^- \rightarrow B^+)}{\begin{array}{c}FB^- \\ \hline FA^-\end{array}}$	$(T\rightarrow_2)$

Tabela 4.2: Regras de expansão de tableaux KE com duas premissas anotadas com polaridades.

Considere, por exemplo, a regra $(F\wedge_1)$:

$$\dfrac{F(\mathbf{A}^- \wedge B^-)}{\begin{array}{c}T\mathbf{A}^+ \\ \hline FB^-\end{array}}$$

Note que a fórmula A ocorre na premissa secundária com polaridade positiva, enquanto que na premissa primária, A ocorre com polaridade negativa. Mais genericamente, em todas as regras de expansão com duas premissas, a fórmula da premissa secundária é uma subfórmula da premissa primária, e as duas ocorrências da fórmula têm polaridades opostas.

[5] Denotamos $*$=valor indefinido.

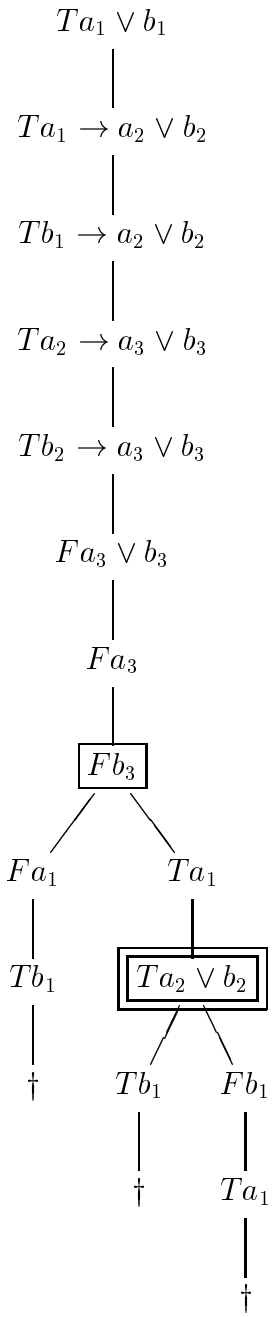


Figura 4.7: Tableau KE para $a_1 \vee b_1, \Gamma_2 \vdash a_3 \vee b_3$ usando a heurística da valoração.

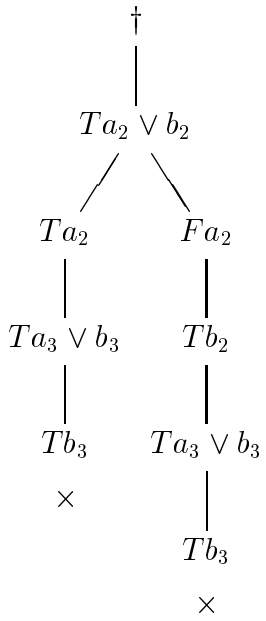


Figura 4.8: Continuação da Figura 4.7.

Deste modo, é vantajoso aplicar em primeiro lugar a regra PB sobre uma subfórmula de uma fórmula β ainda não analisada, que tenha uma fórmula literal existente no ramo como subfórmula de polaridade oposta, ou seja,

Heurística da polaridade: aplicar PB sobre uma subfórmula B de uma fórmula do tipo β A não analisada tal que, para alguma fórmula literal Sp do ramo ($S \in \{T, F\}$, $p \in \mathcal{P}$), A contém uma ocorrência de p e as ocorrências de p em A e em Sp têm polaridades opostas.

O tableau para o seqüente $a_1 \vee b_1, \Gamma_2 \vdash a_3 \vee b_3$ usando a heurística da polaridade é exibido na Figura 4.9.

A regra PB no ramo destacado com caixa simples foi aplicada sobre a_2, subfórmula da fórmula $Ta_2 \rightarrow \mathbf{a_3^+} \vee b_3$, que é a primeira fórmula β não analisada que contém um literal do ramo ($F\mathbf{a_3^-}$) com polaridade oposta.

A regra PB no ramo destacado com caixa dupla foi aplicada sobre a_1, subfórmula da fórmula $Ta_1 \rightarrow \mathbf{a_2^+} \vee b_2$, que é a primeira fórmula β não analisada que contém um literal do ramo ($F\mathbf{a_2^-}$) com polaridade oposta.

Note que este tableau é bem menor que o tableau da Figura 4.7, por que a ordem de escolha da heurística da polaridade é mais natural que a ordem de escolha da

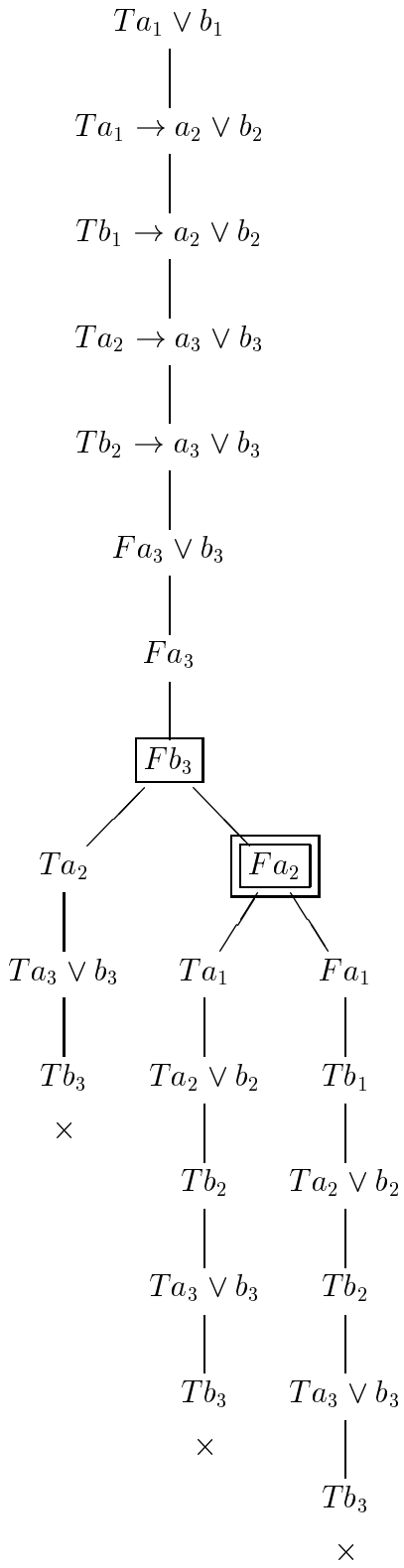


Figura 4.9: Tableau KE para $a_1 \vee b_1, \Gamma_2 \vdash a_3 \vee b_3$ usando a heurística da polaridade.

heurística da valoração.

4.3.2 O método KE-S_3

O método KE-S_3 difere do método KE apenas na manutenção do conjunto-contexto S. O conjunto contexto é inicializado no método virtual `preClose()`, é atualizado na aplicação da regra de uma premissa ($T\neg$), e é retornado para o nó-pai no método virtual `postClose()`.

Duas heurísticas foram implementadas para o método KE-S_3. Ambas utilizam o *critério da distância*[6] descrito a seguir para definir a ordem de seleção das fórmulas a serem analisadas.

O Critério da distância. Toda fórmula pode ser representada por uma árvore onde os nós internos são os conectivos lógicos e as folhas são os átomos. Mais genericamente, um seqüente

$$A_1, A_2, \ldots, A_n \vdash B_1, B_2, \ldots, B_m$$

Pode ser representado pela árvore da Figura 4.10.

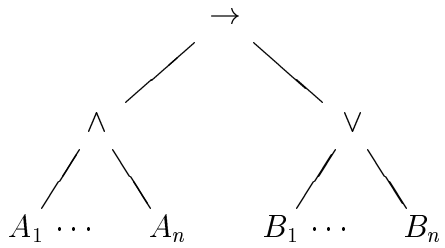


Figura 4.10: Árvore para o seqüente $A_1, A_2, \ldots, A_n \vdash B_1, B_2, \ldots, B_m$.

Nessa árvore, uma medida de interesse é a *distância* entre os átomos da fórmula.

Definimos a distância entre dois átomos a e b como a menor das distâncias entre ocorrências de a e ocorrências de b. Denotamos $d(a, b)$.[7]

[6] O critério da distância é original, mas guarda algumas semelhanças com o grafo de dependências de Kowalski [Kow79], no sentido em que busca a solução do problema a partir das conclusões (abordagem *top-down*).

[7] na prática, essa distância é computada montando-se a árvore do seqüente como descrito anteriormente, e calculando-se a distância entre os nós internos da árvore. Depois, calcula-se a distância entre os átomos p_1 e p_2 segundo a fórmula:

$$d(p_1, p_2) = min\{d(p_1, n_k) + d(n_k, n_l) + d(n_l, p_2)\}$$

onde n_k é um nó interno com pelo menos um filho igual a p_1 ($d(p_1, n_k) = 1$) e n_l é um nó interno com pelo menos um filho igual a p_2 ($d(p_2, n_l) = 1$).

Heuristicamente, podemos supor que um átomo estará próximo do local onde ele é necessário na prova. Em outras palavras, assumimos uma premissa de *relevância local*, uma vez que usaremos a noção de distância para determinar a relevância de um átomo para a prova.

Suponhamos que as arestas $(\wedge, A_i)$, $1 \leq i \leq n$, e $(\vee, B_j)$, $1 \leq j \leq m$ sejam assinaladas com peso ∞, e as demais arestas com peso 1. Seja b um átomo com ocorrência em alguma fórmula B_j e a um átomo com ocorrência em alguma fórmula A_i. Então o único modo de a distância entre b e a ser finita é existir um caminho que começa em b e termina em a que não passa pelas arestas de peso infinito. Ou seja, algum átomo p do caminho tem ocorrência tanto em uma fórmula A_i quanto em uma fórmula B_j.

A partir do parágrafo anterior, podemos fazer a seguinte observação: se fizermos uma busca a partir das conclusões (fórmulas B_j), através de caminhos de comprimento finito, então as fórmulas totalmente irrelevantes para a prova (fórmulas A_i cujos átomos não ocorrem em nenhum B_j) não serão atingidas pela busca.

Mais uma definição é necessária para que possamos aplicar o critério da distância às heurísticas do método KE-S_3.

Definimos a distância entre uma fórmula A e um conjunto de átomos B como

$$d(A, B) = min\{d(a, b)\}$$

onde a é átomo de A e $b \in B$.

A primeira heurística implementada para o método KE-S_3 é similar à heurística da polaridade para o método KE. Sua ordem de preferência é a seguinte:

SP_1 primeira ocorrência de fórmula do tipo α ainda não aplicada;

SP_2 primeira ocorrência de fórmula do tipo β ainda não aplicada, usando uma fórmula literal como secundária;

SP_3 dentre as fórmulas do tipo β ainda não aplicadas que satisfazem a Heurística da Polaridade, regra PB sobre subfórmula da ocorrência com o menor número de átomos não ocorrentes em S e com menor distância do conjunto de literais do ramo.

A escolha da fórmula com o menor número de átomos não ocorrentes em S procura conservar o conjunto contexto S com o menor tamanho possível.

A segunda heurística tem a seguinte ordem de preferência:

PB_1 primeira ocorrência de fórmula do tipo α ainda não aplicada **exceto** fórmula na forma $T\neg A$;

PB_2 primeira ocorrência de fórmula do tipo β ainda não aplicada, usando uma fórmula literal como secundária;

PB_3 dentre as fórmulas do tipo β ainda não aplicadas que satisfazem a Heurística da Polaridade, regra PB sobre subfórmula da ocorrência com o menor número de átomos não ocorrentes em S e com menor distância do conjunto de literais do ramo;

PB_4 primeira ocorrência de fórmula do tipo α na forma $T\neg A$.

Ou seja, a primeira heurística dá preferência a fórmulas do tipo α, sem distinção da regra $(T\neg)$, que adiciona elementos ao conjunto-contexto S. Já a segunda heurística adia as aplicações dessa regra até que nenhuma outra regra possa ser aplicada.

Exemplo 4.1 Vamos provar a validade do seqüente $a\vee c, e \to f, a \to b, c \to d \vdash b\vee d$ usando a primeira heurística SP_i.

A Figura 4.11 mostra a representação em árvore do seqüente.

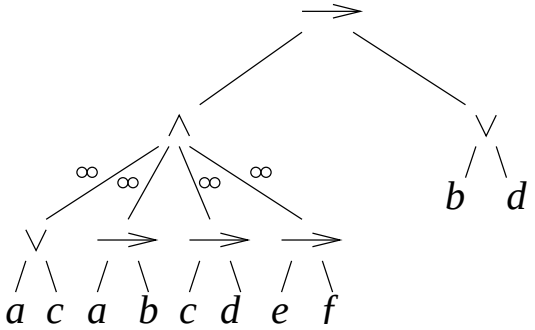


Figura 4.11: Representação em árvore do seqüente $a \vee c, e \to f, a \to b, c \to d \vdash b \vee d$.

A partir da árvore, calculamos as distâncias entre os átomos, que são exibidas na Tabela 4.3.

$d(\cdot,\cdot)$	a	b	c	d	e	f
a	0	2	2	4	∞	∞
b		0	4	2	∞	∞
c			0	2	∞	∞
d				0	∞	∞
e					0	2
f						0

Tabela 4.3: Tabela de distâncias entre os átomos do seqüente $a \vee c, e \rightarrow f, a \rightarrow b, c \rightarrow d \vdash b \vee d$.

O tableau para o seqüente é exibido na Figura 4.12.

As fórmulas 6 e 7 foram obtidas a partir da aplicação da regra $(F\vee)$ sobre a fórmula 5. Neste momento, não é possível aplicar nenhuma regra do tipo α e nenhuma regra do tipo β. É necessário aplicar a regra (PB) sobre alguma fórmula do tipo β ainda não analisada (as fórmulas candidatas são 1, 2, 3 e 4). A fórmula escolhida é a 3, porque dentre as que têm o menor número de átomos fora do conjunto-contexto $S = \emptyset$, 2, é a primeira cuja distância do conjunto de literais do ramo $\{Fb, Fd\}$ é mínima, 0. A fórmula 9 foi obtida a partir da aplicação da regra $(T \rightarrow_1)$ sobre 3 e 8, o que fecha o ramo esquerdo do tableau. A fórmula 11 foi obtida a partir da aplicação da regra $(T\vee_1)$ sobre 1 e 10. A fórmula 12 foi obtida a partir da aplicação da regra $(T \rightarrow_1)$ sobre 4 e 11, o que fecha o ramo direito do tableau.□

Exemplo 4.2 Vamos provar a validade do mesmo seqüente do exemplo anterior, mas com as fórmulas em forma clausal, $a \vee c, \neg e \vee f, \neg a \vee b, \neg c \vee d \vdash b \vee d$ usando a primeira heurística SP_i.

A Figura 4.13 mostra a representação em árvore do seqüente.

A partir da árvore, calculamos as distâncias entre os átomos, que são exibidas na Tabela 4.4.

$d(\cdot,\cdot)$	a	b	c	d	e	f
a	0	3	2	5	∞	∞
b		0	5	2	∞	∞
c			0	3	∞	∞
d				0	∞	∞
e					0	3
f						0

Tabela 4.4: Tabela de distâncias entre os átomos do seqüente $a \vee c, \neg e \vee f, \neg a \vee b, \neg c \vee d \vdash b \vee d$.

O tableau para o seqüente é exibido na Figura 4.14.

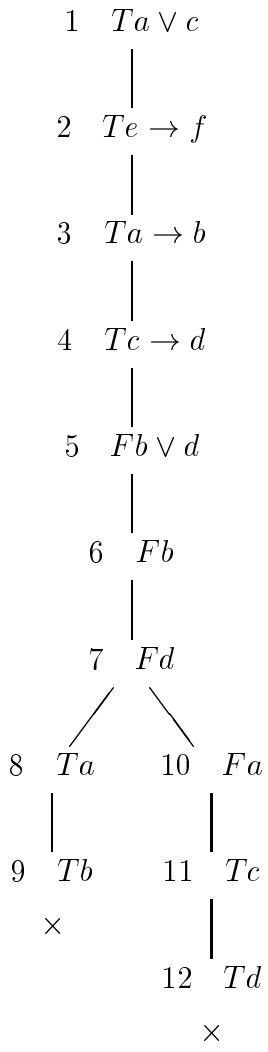


Figura 4.12: Tableau para o seqüente $a \vee c, e \to f, a \to b, c \to d \vdash b \vee d$ usando a heurística SP_i.

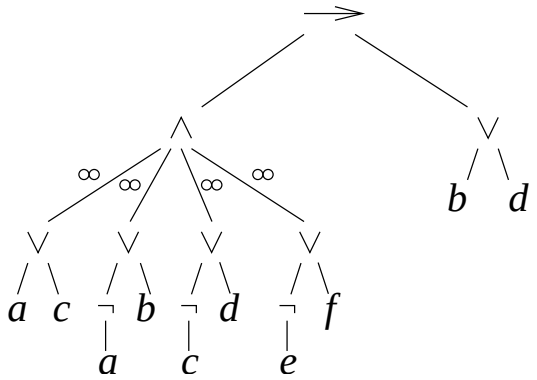


Figura 4.13: Representação em árvore do seqüente $a \vee c, \neg e \vee f, \neg a \vee b, \neg c \vee d \vdash b \vee d$.

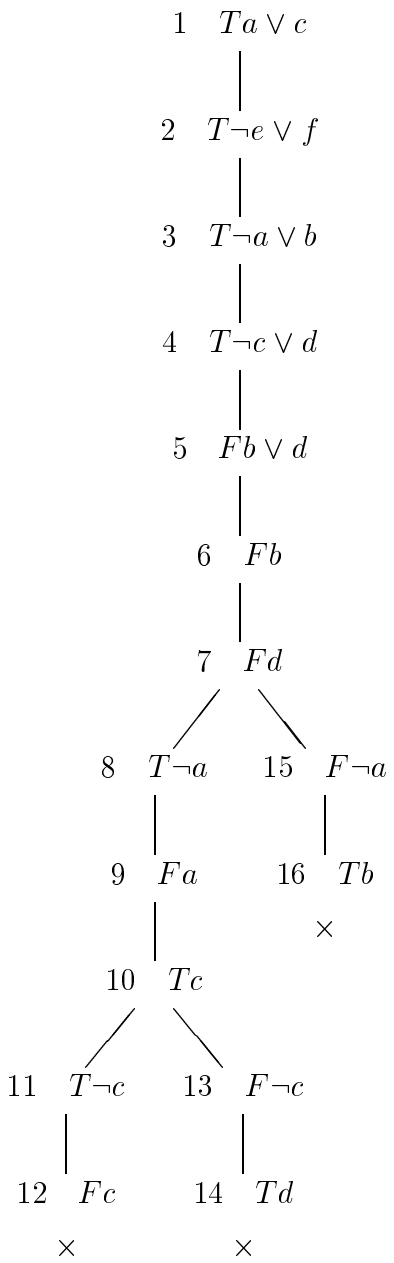


Figura 4.14: Tableau para o seqüente $a \vee c, \neg e \vee f, \neg a \vee b, \neg c \vee d \vdash b \vee d$ usando a heurística SP_i.

As fórmulas 6 e 7 foram obtidas a partir da aplicação da regra ($F\vee$) sobre a fórmula 5. Neste momento, não é possível aplicar nenhuma regra do tipo α e nenhuma regra do tipo β. É necessário aplicar a regra (PB) sobre alguma fórmula do tipo β ainda não analisada (as fórmulas candidatas são 1, 2, 3 e 4). A fórmula escolhida é a 3, porque dentre as que têm o menor número de átomos fora do conjunto-contexto $S = \emptyset$, 2, é a primeira cuja distância do conjunto de literais do ramo $\{Fb, Fd\}$ é mínima, 0. A fórmula 9 foi obtida a partir da aplicação da regra ($T\neg$) sobre 8, o que obriga $S = \{a\}$. A fórmula 10 foi obtida a partir da aplicação da regra ($T\vee_1$) sobre 1 e 9. Neste momento, não é possível aplicar nenhuma regra do tipo α e nenhuma regra do tipo β. É necessário aplicar a regra (PB) sobre alguma fórmula do tipo β ainda não analisada (as fórmulas candidatas são 2 e 4). A fórmula escolhida é a 4, porque dentre as que têm o menor número de átomos fora do conjunto-contexto $S = \emptyset$, 2, é a fórmula cuja distância do conjunto de literais do ramo $\{Fb, Fd, Fa, Tc\}$ é mínima, 0. A fórmula 12 foi obtida a partir da aplicação da regra ($T\neg$) sobre 11, o que obriga $S = \{a, c\}$ e fecha o ramo terminado em 12. A fórmula 14 foi obtida a partir da aplicação da regra ($T\vee_1$) sobre 4 e 13, o que fecha o ramo terminado em 14. A fórmula 16 foi obtida a partir da aplicação da regra ($T\vee_1$) sobre 3 e 15, o que fecha o ramo terminado em 16.

Note que os átomos e e f não entraram em S porque a fórmula 2, que é irrelevante para a prova, não foi analisada. □

Finalmente, apresentamos na figura 4.15 o mesmo exemplo utilizando a heurística PB_i.

No capítulo 5, os resultados obtidos pelas duas heurísticas são comparados.

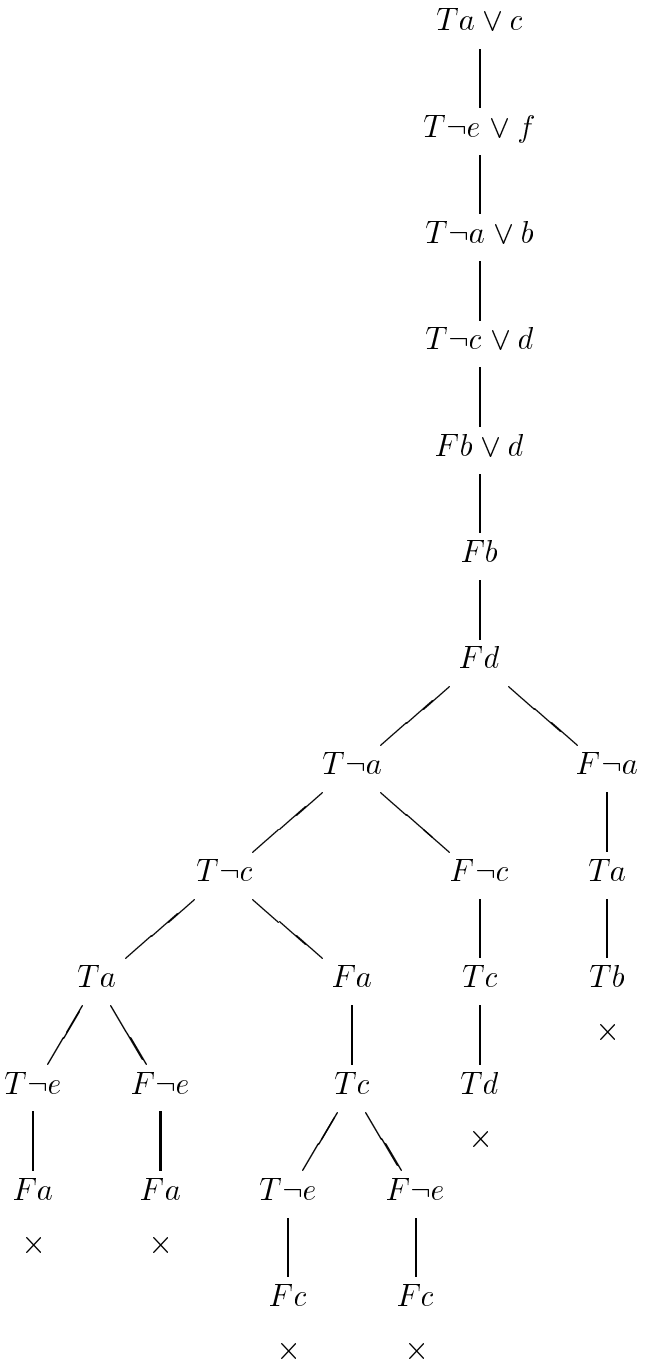


Figura 4.15: Tableau para o seqüente $a \vee c, \neg e \vee f, \neg a \vee b, \neg c \vee d \vdash b \vee d$ usando a heurística PB_i. O tableau fecha com $S = \{a, c\}$.

Capítulo 5

Resultados

5.1 Os casos de teste

Para fazer o teste comparativo dos métodos de tableaux implementados, consideramos os principais conjuntos de fórmulas que são conhecidamente difíceis. Os casos de teste são os seguintes:

1. `Gamma`: exemplo extraído de [CS00], p. 22.

   $$a_1 \vee b_1, \Gamma_n \vdash a_{n+1} \vee b_{n+1}$$

   onde

   $$\Gamma_n = \bigcup_{i=1}^{n} \{a_i \rightarrow a_{i+1} \vee b_{i+1}, b_i \rightarrow a_{i+1} \vee b_{i+1}\}$$

2. `H`: exemplo extraído de [D'A92].

   $$\vdash H_n$$

   onde H_n é dado por

   $$\begin{aligned} H_1 &= p_1 \vee \neg p_1 \\ H_2 &= (p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2) \vee (\neg p_1 \wedge p_2) \vee (\neg p_1 \wedge \neg p_2) \\ &\vdots \end{aligned}$$

3. `PHP` - *Pigeon Hole Principle* ou Princípio do Escaninho: exemplo extraído de [CS00], p. 29.

   $$A_n \vdash B_n$$

onde

$$A_n = \bigwedge_{i=0}^{n} \bigvee_{j=0}^{n-1} p_{ij}$$
$$B_n = \bigvee_{j=0}^{n-1} \bigvee_{i=0}^{n-1} \bigvee_{k=i+1}^{n} (p_{ij} \wedge p_{kj})$$

4. `Statman`: exemplo extraído de [Sta78].[1]

$$\bigwedge_{i=1}^{n} (A_i \vee B_i) \vdash c_n \vee d_n$$

onde

$$A_1 = c_1 \qquad A_{i+1} = F_i \rightarrow c_{i+1}$$
$$B_1 = d_1 \qquad B_{i+1} = F_i \rightarrow d_{i+1}$$

$$F_k = \bigwedge_{j=1}^{k} (c_j \vee d_j)$$

5.2 Resultado dos casos de teste

Todos os métodos de tableau foram testados com cada caso de teste. Para cada teste, foram medidos:

- o número de nós do tableau fechado;
- o número de fórmulas do tableau fechado;
- o tempo gasto em segundos para o fechamento do tableau.

[1] Statman [Sta78] mostrou que esta família de fórmulas tem provas polinomiais com corte, mas sem corte as provas são exponenciais (ver [CS00], p. 24).

5.2.1 Tableau analítico

A Tabela 5.1 mostra o resultado dos testes para o tableau analítico com as heurísticas D_i[2] e U_i[3].

		analytic			analytic+BU		
Caso	n	# nós	# fórmulas	Tempo (s)	# nós	# fórmulas	Tempo (s)
`Gamma`	1	21	27	0.002793	11	17	0.001436
	2	125	133	0.020373	19	27	0.003645
	3	789	799	0.155993	27	37	0.004496
	4	4741	4753	1.109542	35	47	0.006930
	5	27333	27347	7.483438	43	57	0.009229
	6	153221	153237	48.384375	51	67	0.012195
`H`	1	1	5	0.000152	1	5	0.000152
	2	17	29	0.002124	17	31	0.002201
	3	511	712	0.086488	511	838	0.091186
`PHP`	1	3	8	0.000439	3	8	0.000437
	2	107	119	0.017646	107	119	0.017904
	3	67051	67076	25.846443	65047	65072	24.697791
`Statman`	1	3	7	0.000408	3	7	0.000408
	2	15	28	0.002118	11	20	0.001512
	3	63	117	0.010052	59	89	0.009678
	4	239	438	0.043034	441	636	0.089275
	5	927	1703	0.179719	4103	5919	0.970773
	6	3647	6728	0.879744	45325	65802	12.873804

Tabela 5.1: Teste do método de tableaux analíticos.

Observando os dados da tabela, vemos que nos casos `H` e `Statman`, a heurística D_i tem um melhor desempenho, enquanto que nos casos `Gamma` e `PHP` a heurística U_i é a melhor.

Era de se esperar que nenhuma das duas heurísticas fosse melhor que a outra no caso geral, porque nenhuma delas usa um critério inteligente para a escolha da ordem de análise das fórmulas, apenas a ordem em que ocorrem no tableau.

No Apêndice B são exibidos os gráficos dos resultados dos testes.

[2] coluna "analytic"
[3] coluna "analytic+BU"

5.2.2 Tableau KE

A Tabela 5.2 mostra o resultado dos testes para o tableau analítico com a heurística K_i[4], a heurística da valoração [5] e a heurística da polaridade [6].

		KE			KE+V			KE+P		
Caso	n	#nós	#fmls	T (s)	#nós	#fmls	T (s)	#nós	#fmls	T (s)
Gamma	1	3	13	0.001299	3	13	0.001737	3	13	0.001403
	2	13	42	0.010141	7	23	0.006142	7	23	0.006337
	3	37	117	0.046355	11	33	0.015440	11	33	0.015199
	7	109	338	0.200616	15	43	0.030519	15	43	0.030138
	5	325	997	0.820640	19	53	0.052747	19	53	0.052012
	6	973	2970	3.564614	23	63	0.083877	23	63	0.082882
	7	2917	8885	13.231674	27	73	0.124285	27	73	0.151914
	8	8749	26626	49.277312	31	83	0.176152	31	83	0.175679
H	1	1	4	0.000111	1	4	0.000109	1	4	0.000108
	2	3	17	0.001450	5	17	0.001599	5	17	0.001490
	3	21	103	0.019906	13	55	0.009115	13	55	0.008841
	4	105	677	0.492401	29	179	0.047016	29	179	0.045761
	5	465	4777	10.325982	61	619	0.290332	61	619	0.227181
PHP	1	1	6	0.000199	1	6	0.000198	1	6	0.000197
	2	3	29	0.002681	3	28	0.002447	3	28	0.002449
	3	61	261	0.254678	11	110	0.053459	11	110	0.053243
	4	483	2219	8.619986	47	511	1.268295	47	511	1.378396
Statman	1	1	5	0.000165	1	5	0.000163	1	5	0.000162
	2	7	30	0.003533	3	18	0.001745	3	18	0.009173
	3	31	124	0.060553	13	58	0.019585	13	58	0.018665
	4	207	711	0.407214	47	178	0.134915	47	178	0.157982
	5	1079	3696	4.181898	145	508	0.763058	145	508	0.665001
	6	5391	19539	36.835890	403	1358	3.136678	403	1358	3.010920

Tabela 5.2: Teste do método de tableaux KE.

Observando os dados da tabela, vemos que a heurística da polaridade apresenta um desempenho melhor que o das outras duas heurísticas. Isto se deve ao fato de a regra de preferência usando a polaridade das fórmulas estar intimamente ligada à estrutura das regras de expansão do tableau KE.

A Tabela 5.3 mostra os resultados dos mesmos testes utilizando a heurística da distância em conjunção com as heurísticas da valoração e da polaridade.[7]

Podemos observar que a heurística da distância nada ajuda nestes casos de teste.

[4] coluna "KE"
[5] coluna "KE+V"
[6] coluna "KE+P"
[7] As heurísticas são modificadas da seguinte maneira: aplicar PB sobre uma subfórmula de uma fórmula A que satisfaça as condições da heurística da valoração/polaridade e que tenha distância mínima do conjunto de literais do ramo.

		KE+V			KE+P		
Caso	n	#nós	#fmls	T (s)	#nós	#fmls	T (s)
Gamma	1	3	13	0.001737	3	13	0.001761
	2	5	23	0.006490	5	23	0.029050
	3	7	35	0.016886	7	35	0.020094
	4	9	49	0.035752	9	49	0.047569
	5	11	65	0.096447	11	65	0.068034
	6	13	83	0.115460	13	83	0.133260
	7	15	103	0.188568	15	103	0.230932
	8	17	125	0.359210	17	125	0.351875
H	1	1	4	0.000114	1	4	0.000113
	2	3	17	0.001323	3	17	0.001324
	3	21	103	0.025815	7	55	0.009094
	4	105	677	0.616623	15	185	0.073449
	5	465	4777	10.661724	31	661	0.531249
PHP	1	1	6	0.000197	1	6	0.000196
	2	3	29	0.002991	3	29	0.003104
	3	61	261	0.358510	21	173	0.144977
	4	483	2219	11.360033	143	1289	5.716319
Statman	1	1	5	0.000158	1	5	0.000161
	2	3	18	0.001990	3	18	0.001936
	3	31	110	0.040688	31	105	0.041894
	4	183	620	0.521748	135	483	0.463096
	5	783	2688	4.190906	509	1860	3.561454
	6	3015	10948	27.988467	1779	6750	22.077161

Tabela 5.3: Teste do método de tableaux KE com heurística da distância.

No Apêndice B são exibidos os gráficos dos resultados dos testes.

5.2.3 Tableau KE-S_3

Para os testes do método de tableaux KE-S_3, vamos eliminar as conjunções e as implicações das fórmulas. Deste modo, forçamos o aparecimento de mais algumas negações nas fórmulas, tentando induzir o uso da regra $(T\neg)$.[8]

Os casos de teste ficam sendo, então, os seguintes:

1. `Gamma`

$$a_1 \vee b_1, \Gamma_n \vdash a_{n+1} \vee b_{n+1}$$

onde

$$\Gamma_n = \bigcup_{i=1}^{n} \{\neg a_i \vee (a_{i+1} \vee b_{i+1}), \neg b_i \vee (a_{i+1} \vee b_{i+1})\}$$

2. `H`

$$\vdash H_n$$

onde H_n é dado por

$$\begin{aligned} H_1 &= p_1 \vee \neg p_1 \\ H_2 &= \neg(p_1 \vee p_2) \vee \neg(p_1 \vee \neg p_2) \vee \neg(\neg p_1 \vee p_2) \vee \neg(\neg p_1 \vee \neg p_2) \\ &\vdots \end{aligned}$$

3. `PHP`

$$\vdash A_n \vee B_n$$

onde

$$\begin{aligned} A_n &= \neg \bigvee_{i=0}^{n} \neg \bigvee_{j=0}^{n-1} p_{ij} \\ B_n &= \bigvee_{j=0}^{n-1} \bigvee_{i=0}^{n-1} \bigvee_{k=i+1}^{n} \neg(\neg p_{ij} \vee \neg p_{kj}) \end{aligned}$$

[8] Note que esta transformação não é válida nas lógicas S_3, ou seja, estes casos de teste não necessariamente são logicamente equivalentes aos casos de teste da seção anterior na semântica de S_3.

4. `Statman`

$$\bigcup_{i=1}^{n}(A_i \vee B_i) \vdash c_n \vee d_n$$

onde

$$A_1 = c_1 \qquad A_{i+1} = \neg F_i \vee c_{i+1}$$
$$B_1 = d_1 \qquad B_{i+1} = \neg F_i \vee d_{i+1}$$

$$F_k = \neg \bigvee_{j=1}^{k} \neg(c_j \vee d_j)$$

A Tabela 5.4 mostra o resultado dos testes para o tableau KE-S_3 com a heurística SP_i[9] e a heurística PB_i[10], mas inicialmente sem a heurística da distância.

		KE-S_3				KE-S_3+PB			
Caso	n	#nós	#fmls	T (s)	$\|S\|$	#nós	#fmls	T (s)	$\|S\|$
Gamma	1	9	28	0.005383	2	11	31	0.007239	2
	2	19	60	0.022599	4	19	62	0.022715	4
	3	45	134	0.105748	6	53	160	0.109554	6
	4	109	300	0.261566	8	105	306	0.249372	8
	5	173	482	0.596367	10	189	550	0.670404	10
	6	429	1140	1.826961	12	363	1022	1.575956	12
	7	685	1862	3.722508	14	633	1790	3.489517	14
H	1	1	6	0.000191	1	1	6	0.000219	1
	2	3	19	0.001278	2	11	37	0.004655	2
	3	7	57	0.007294	3	41	171	0.047230	3
PHP	1	3	16	0.001061	2	3	20	0.001228	2
	2	51	192	0.066302	6	37	227	0.054892	6
	3	935	4228	5.839093	12	841	3995	6.882294	12
	4	22667	126694	530.427588	20	22177	97848	693.912202	20
Statman	1	1	5	0.000173	0	1	5	0.000174	0
	2	7	48	0.006805	2	7	44	0.009174	2
	3	37	219	0.075732	4	55	299	0.192884	4
	4	125	686	0.432528	6	407	2003	2.317840	6

Tabela 5.4: Teste do método de tableaux KE-S_3 sem a heurística da distância.

Observando os dados da tabela, vemos que para nenhum dos casos de teste existe diferença no tamanho do conjunto-contexto S no tableau final. Ou seja, o uso da heurística PB_i não melhora a eficácia do método. Além disso, para todos os casos, o uso da heurística SP_i torna o método mais eficiente.

Vejamos agora os resultados com a heurística da distância (Tabela 5.5).

[9] coluna "KE-S_3"
[10] coluna "KE-S_3+PB"

		KE-S_3				KE-S_3+PB			
Caso	n	#nós	#fmls	T (s)	$\|S\|$	#nós	#fmls	T (s)	$\|S\|$
Gamma	1	9	26	0.006016	2	11	30	0.008323	2
	2	23	72	0.035984	4	25	82	0.041576	4
	3	63	198	0.222385	6	67	218	0.180890	6
	4	169	529	0.857960	8	187	598	0.897771	8
	5	455	1427	3.404131	10	491	1561	3.643146	10
	6	1277	4016	14.220320	12	1323	4184	14.347676	12
	7	3703	11666	56.962195	14	3695	11671	56.162057	14
H	1	1	6	0.000196	1	1	6	0.000246	1
	2	5	21	0.001990	2	11	34	0.004747	2
	3	13	63	0.010944	3	43	157	0.036399	3
PHP	1	3	16	0.001093	2	3	20	0.001307	2
	2	13	75	0.020335	6	41	237	0.068699	6
	3	69	411	0.595718	12	907	4193	8.174117	12
	4	577	3644	20.874021	20	22993	100704	786.988453	20
Statman	1	1	5	0.000173	0	1	5	0.000173	0
	2	7	48	0.007948	2	7	49	0.012063	2
	3	39	231	0.095406	4	67	437	0.303217	4
	4	157	890	0.819562	6	777	5130	7.793823	6

Tabela 5.5: Teste do método de tableaux KE-S_3 com a heurística da distância.

Novamente, a heurística da distância não melhora em nenhum dos casos o desempenho do algoritmo.

A Tabela 5.6 mostra o resultado dos testes `Gamma` com fórmulas irrelevantes para o tableau KE-S_3 com a heurística SP_i e a heurística PB_i, e a Tabela 5.7 mostra os resultados das heurísticas em conjunção com a heurística da distância.

A família de testes é a seguinte:

$$c_1 \vee d_1, \Delta_n, a_1 \vee b_1, \Gamma_n \vdash a_{n+1} \vee b_{n+1}$$

onde

$$\Delta_n = \bigcup_{i=1}^{n} \{\neg c_i \vee (c_{i+1} \vee d_{i+1}), \neg d_i \vee (c_{i+1} \vee d_{i+1})\}$$

$$\Gamma_n = \bigcup_{i=1}^{n} \{\neg a_i \vee (a_{i+1} \vee b_{i+1}), \neg b_i \vee (a_{i+1} \vee b_{i+1})\}$$

Note que as fórmulas irrelevantes atrapalham a prova (comparar os tempos de execução com os tempos da Tabela 5.4), e quando usamos PB_i sem a heurística da

	KE-S_3				KE-S_3+PB			
n	#nós	#fmls	T (s)	$\|S\|$	#nós	#fmls	T (s)	$\|S\|$
1	19	56	0.028246	2	39	102	0.048521	4
2	45	132	0.131016	4	145	403	0.373984	8
3	73	214	0.292364	6	429	1184	1.923869	12
4	209	576	1.228155	8	1965	5310	14.347284	16
5	293	820	2.234904	10	5395	14539	50.660406	20
6	841	2268	8.295132	12	29309	78401	361.697747	24

Tabela 5.6: Teste do método de tableaux KE-S_3 com fórmulas irrelevantes, sem a heurística da distância.

	KE-S_3				KE-S_3+PB			
n	#nós	#fmls	T (s)	$\|S\|$	#nós	#fmls	T (s)	$\|S\|$
1	9	31	0.022876	2	15	48	0.062565	2
2	23	77	0.163566	4	55	152	0.332533	4
3	63	205	1.069044	6	137	372	1.650295	6
4	169	538	5.502665	8	313	868	7.334279	8
5	455	1438	27.214882	10	689	1979	30.051540	10
6	1277	4029	125.476909	12	1609	4782	124.422070	12

Tabela 5.7: Teste do método de tableaux KE-S_3 com fórmulas irrelevantes e com a heurística da distância.

distância, o método adiciona átomos irrelevantes ao conjunto-contexto S, enquanto que com o uso da distância, nenhuma das duas heurísticas adiciona átomos irrelevantes ao conjunto-contexto S. Além disso, em nenhum dos testes executados, a heurística PB_i, que adia as aplicações da regra $(T\neg)$, é melhor. Isso se deve ao fato de o adiamento dessas aplicações obrigar o adiamento da análise de outras fórmulas, o que pode gerar bifurcações do tableau que com a outra heurística não ocorreriam.

No Apêndice B são exibidos os gráficos dos resultados dos testes.

Capítulo 6

Conclusão

Vimos que a escolha da heurística que define a ordem de aplicação das regras de um método de tableaux é um passo crucial no desenvolvimento de um tal método. Observamos na prática o fato provado por D'Agostino [D'A90] que os tableaux KE são mais eficientes que os tableaux analíticos. Além disso, a heurística da polaridade para a escolha da fórmula sobre a qual se aplica a regra PB é a melhor heurística dentre as implementadas, pois aproveita a estrutura das regras de expansão. Finalmente, descobrimos uma heurística baseada na noção de relevância local que impede a análise de fórmulas totalmente irrelevantes para a prova. No entanto, essa heurística não apresenta nenhuma vantagem quando todos os átomos ocorrentes nas fórmulas são relevantes. Vale lembrar que a heurística pode ser enganada por outros testes. Por exemplo, pode-se forjar um teste com pelo menos uma fórmula que contenha um átomo relevante para a prova e um átomo irrelevante. Na prática o que se pode fazer é adaptar uma heurística de acordo com o conhecimento da aplicação onde o método é aplicado.

Pela análise dos resultados, observamos que o uso da heurística PB_i, que adia o aumento do conjunto-contexto S leva à pior performance, enquanto que o uso da heurística SP_i torna as provas do Tableau KE-S_3 iguais às provas do Tableau KE, a menos da informação adicional do conjunto-contexto S nos passos intermediários da resolução do tableau. Em outras palavras, o Tableau KE-S_3 não exibe um ganho de performance se comparado ao Tableau KE.

Em [FW02a], Finger e Wassermann analisam as perdas e ganhos do sistema KE-S_3 com relação ao sistema original S_3 de Cadoli e Schaerf, e concluíram que, apesar de o sistema KE-S_3 ser mais *expressivo* (admite mais teoremas que o sistema original

com um mesmo conjunto-contexto S), exibe menos *controle*. Então, eles propõem uma generalização KE-S_e de KE-S_3 que apresenta vários conjuntos-contexto, um para cada regra de expansão. Cada regra pode ter um custo diferente, e resta determinar esses custos para cada aplicação específica.

Existem trabalhos sendo feitos [FW02b] sobre a viabilidade de desenvolvimento de um sistema de tableaux para a lógica S_1, o que permitiria desenvolver um sistema de raciocínio por aproximações próximo ao de Cadoli e Schaerf totalmente baseado em sistemas de tableaux.

Apêndice A

Manual do Usuário do Arcabouço para Métodos de Tableau

A Figura A.1 mostra um esquema detalhado do arcabouço para méto-dos de tableau. As classes principais do arcabouço estão representadas na figura.

Nas seções a seguir será explicado o funcionamento de cada uma das classes apresentadas na figura.

A.1 Classe `Formula`

A classe `Formula` encapsula a fórmula da lógica proposicional. A declaração da classe `Formula` é mostrada na Figura A.2.

A.1.1 enum `opType`

A enumeração `opType` tem como elementos os tipos de operadores que podem ocorrer em uma fórmula.

O valor `AND` representa o operador *binário* de conjunção $\wedge$. O valor `ANDN` representa o operador *n-ário* de conjunção $\wedge$. O valor `OR` re-presenta o operador *binário* de disjunção $\vee$. O valor `ORN` representa o operador *n-ário* de disjunção $\vee$. O valor `IMPLIES` representa o ope-rador de implicação $\rightarrow$. O valor `NOT` representa o operador *unário* de negação $\neg$. O valor `ATOM` é um valor especial, indicando que a fórmula onde ele ocorre é atômica.

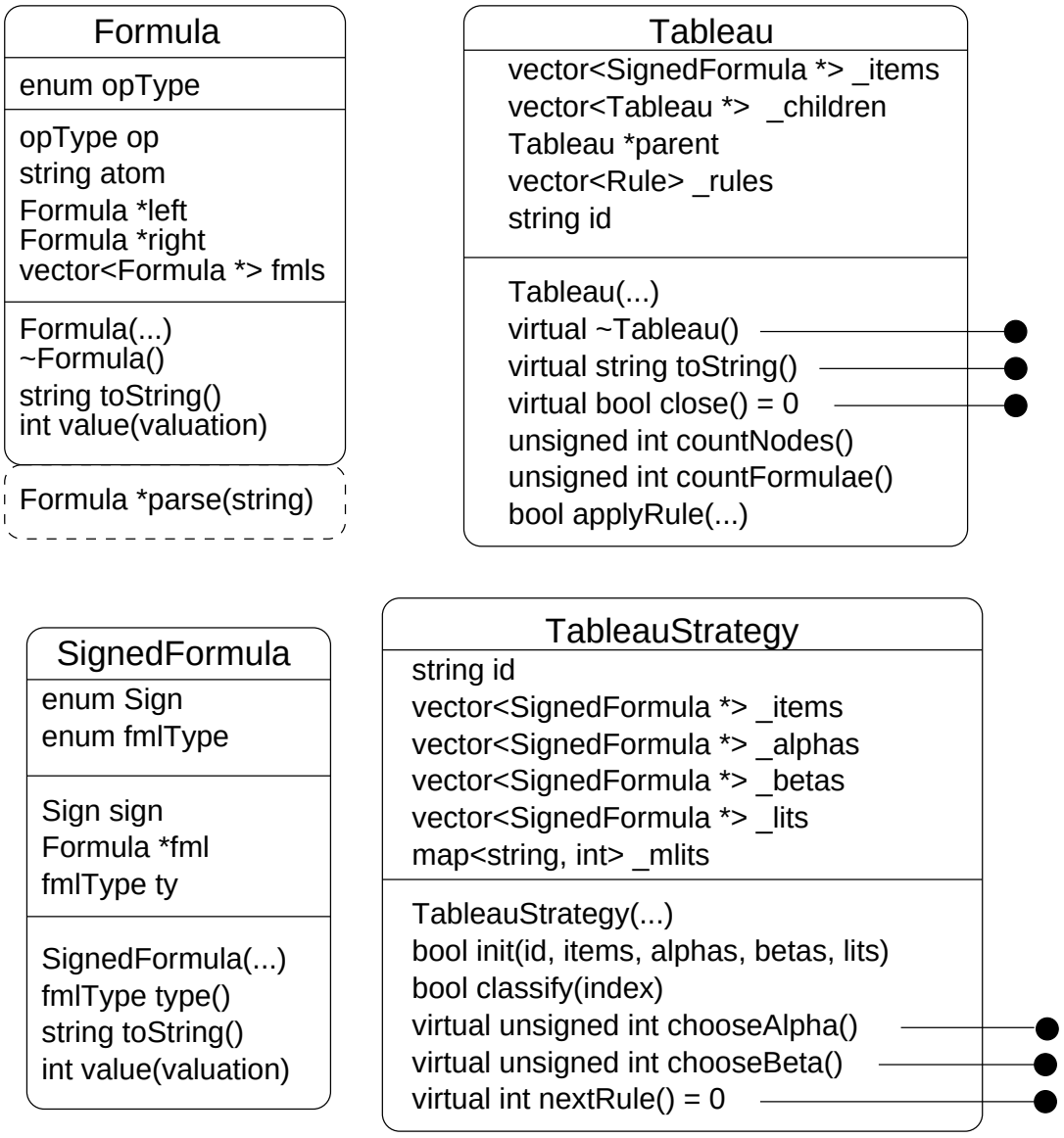


Figura A.1: Classes componentes do arcabouço para métodos de tableau.

```
class Formula {
 public:
  enum opType {AND, ANDN, OR, ORN, IMPLIES, NOT, ATOM;}

  Formula(opType t, vector<Formula *>& vfml);
  Formula(opType t, Formula *l, Formula *r);
  Formula(opType t, Formula *r);
  Formula(const string& a);
  Formula(const Formula& rhs);
  ~ Formula();

  string toString() const;
  int value(map<string, int>& valuation) const;
  int polarity(const string& atom) const;

  opType op;
  string atom;
  Formula *left;
  Formula *right;
  vector<Formula *> fmls;
};

Formula *parse(const string& s);
```

Figura A.2: Declaração da classe `Formula`.

A.1.2 Construtores e destrutor

O construtor

```
Formula(opType t, vector<Formula *>& vfml)
```

é o construtor para fórmulas de conjunção e disjunção n-ária (portanto, apenas para $t = ANDN$ ou $t = ORN$). Armazena os operandos no vetor membro `fmls`. Para uma fórmula construída a partir deste construtor, os membros `atom`, `left` e `right` não têm significado e podem ser ignorados.

O construtor

```
Formula(opType t, Formula *l, Formula *r)
```

é o construtor para fórmulas de conjunção, disjunção e implicação binárias (portanto, apenas para $t = AND$, $t = OR$ ou $t = IMPLIES$). Armazena os operandos nos ponteiros membro `left` e `right`. Para uma fórmula construída a partir deste construtor, os membros `atom` e `fmls` não têm significado e podem ser ignorados.

O construtor

```
Formula(opType t, Formula *r)
```

é o construtor para fórmulas de negação (portanto, apenas para $t = NOT$). Armazena o operando no ponteiro membro `right`. Para uma fórmula construída a partir deste construtor, os membros `atom`, `left` e `fmls` não têm significado e podem ser ignorados.

O construtor

```
Formula(const string& a)
```

é o construtor para fórmulas atômicas (portanto, apenas para $t = ATOM$). Armazena o átomo na string membro `atom`. Para uma fórmula construída a partir deste construtor, os membros `left`, `right` e `fmls` não têm significado e podem ser ignorados.

O destrutor da classe é responsável pela destrução (liberação de memória) dos objetos `Formula` correspondentes às subfórmulas.

A.1.3 string toString() const

O método `toString()` retorna a representação em caracteres da fórmula, com todos os parênteses para evitar ambigüidades.

A.1.4 int value(map<string, int>& valuation) const

Dada uma valoração para os átomos `valuation`, o método `value()` retorna:

- -1, se o valor da fórmula é indefinido segundo a valoração `valuation`;
- 0, se o valor da fórmula é falso segundo a valoração;
- 1, se o valor da fórmula é verdadeiro segundo a valoração.

A.1.5 int polarity(const string& atom) const

Dado um átomo `atom`, o método `polarity()` retorna a polaridade da(s) ocorrência(s) de `atom` dentro da fórmula. Retorna:

- -1, se não houver nenhuma ocorrência de `atom` na fórmula;
- 0, se há apenas ocorrências de polaridade negativa de `atom` na fórmula;
- 1, se há apenas ocorrências de polaridade positiva de `atom` na fórmula;
- 2, se há ocorrências de polaridade positiva e negativa.

A.1.6 Formula *parse(const string& s)

Esta função auxiliar faz o *parsing* de uma string que representa uma fórmula. A fórmula, e todas as suas subfórmulas exceto os átomos, devem estar entre parênteses. A função retorna um apontador para uma área recém-alocada onde está o objeto `Formula` representado pela string, ou um apontador nulo se a string não for uma fórmula válida.

A.2 Classe `SignedFormula`

A classe `SignedFormula` encapsula a fórmula assinalada, usada em todos os métodos de tableau do projeto. A declaração da classe `SignedFormula` é mostrada na Figura A.3.

```
class SignedFormula {
 public:
  enum Sign {S_F, S_T;}
  enum fmlType {ALPHA, BETA, LITERAL;}

  SignedFormula(Sign s, Formula *fml);

  fmlType type() const;
  string toString() const;
  int value(map<string, int>& valuation) const;
  int polarity(const string& atom) const;

  Sign sign;
  Formula *formula;
  fmlType ty;
};
```

Figura A.3: Declaração da classe `SignedFormula`.

A.2.1 enum `Sign`

A enumeração `Sign` tem como elementos os dois rótulos possíveis para uma fórmula assinalada: F (valor `S_F`) e T (valor `S_T`).

A.2.2 enum `fmlType`

A enumeração `fmlType` tem como elementos os tipos de fórmulas assinaladas: fórmulas α (valor `ALPHA`), β (valor `BETA`) e fórmulas literais (valor `LITERAL`).

A.2.3 Construtor

O construtor

```
SignedFormula(Sign s, Formula *fml);
```

constrói uma fórmula assinalada com rótulo `s` (armazenado no campo `sign`). A fórmula proposicional é armazenada no campo `formula`.

A.2.4 `fmlType type()` const

Este método retorna o tipo da fórmula assinalada (α, β ou literal).

A.2.5 string `toString()` const

Este método retorna a representação em caracteres da fórmula assinalada.

A.2.6 int `value(map<string, int>& valuation)` const

Este método é similar ao método `Formula::value()`, e incorpora a semântica do rótulo da fórmula já discutida anteriormente.

A.2.7 int `polarity(const string& atom)` const

Este método é similar ao método `Formula::polarity()`, e incorpora a semântica do rótulo da fórmula já discutida anteriormente.

A.3 Classe `Tableau`

A classe `Tableau` encapsula um método de tableau. A declaração da classe `Tableau` é mostrada na Figura A.4.

A.3.1 Construtores e destrutor

O construtor

```
Tableau(const string& id, SignedFormula *fml,
Tableau *parent = NULL)
```

constrói um objeto tableau a partir de uma única fórmula `fml`. É necessário fornecer um identificador para o tableau (`id`), e se o tableau não for raiz (se for um subtableau de um outro tableau), deve-se fornecer um apontador para o tableau-

```
class Tableau {
 public:
  Tableau(const string& id, SignedFormula *fml,
    Tableau *parent = NULL);
  Tableau(const string& id, const vector<SignedFormula *>& fmls),
    Tableau *parent = NULL);
  virtual ~ Tableau();

  virtual void setStrategy(TableauStrategy *strategy);
  virtual string toString(int level=0) const;
  virtual bool close() = 0;
  unsigned int countNodes();
  unsigned int countFormulae();

 protected:
  bool applyRule(unsigned int index, const vector<SignedFormula *>& in,
    const vector<SignedFormula *>& out);
  virtual void createChild(const string& id, SignedFormula *fml) = 0;

  vector<SignedFormula *> _items;
  vector<Tableau *> _children;
  Tableau *_parent;
  vector<Rule> _rules;
  string _id;

 private:
  TableauStrategy *_strategy;
};
```

Figura A.4: Declaração da classe `Tableau`.

pai (`parent`). Este construtor é normalmente utilizado para a construção de subtableaux. Vide a subseção A.3.8.

O construtor

```
Tableau(const string& id,
const vector<SignedFormula *>& fmls),
Tableau *parent = NULL)
```

constrói um objeto tableau a partir de um vetor de fórmulas (`fmls`). Este construtor é normalmente utilizado para a construção do tableau-raiz.

O destrutor deve ser responsável pela dealocação dos objetos alocados dinamicamente pela classe.

A.3.2 virtual void `setStrategy(TableauStrategy *strategy)`

Este método inicializa o membro `_strategy`, que é um apontador para o objeto que contém a estratégia do método de tableau. A classe `TableauStrategy` é discutida em detalhes na seção A.4.

A.3.3 virtual string `toString(int level=0)` const

Este método retorna a representação em caracteres do tableau. Como padrão, essa representação é a concatenação da representação de todas as fórmulas do tableau (armazenadas no vetor `_items`). Este método é virtual porque cada implementação de tableau pode dar uma representação em caracteres diferente (como é o caso da classe `KES3Tableau`, que inclui os conjuntos-contexto S na representação).

A.3.4 virtual bool `close() = 0`

Este é um método virtual puro, ou seja, não tem implementação na classe `Tableau`. Cada implementação de tableau deve dar a implementação mais conveniente.

A regra geral para a implementação deste método é:

- no construtor, chamar o método `setStrategy()`, e depois o método `_strategy->init()` para inicializar o objeto;
- no método close():

- chamar o método `_strategy->classify()` para classificar as fórmulas ainda não classificadas do tableau;
- chamar o método `_strategy->nextRule()` para descobrir a próxima fórmula a ser analisada;
- tratar o valor de retorno de `_strategy->nextRule()` apropriadamente, com os métodos `_strategy->chooseAlpha()` e `_strategy->chooseBeta()`.
- repetir o processo até que se feche o tableau, ou que não haja mais fórmulas a serem analisadas.

A.3.5 unsigned int `countNodes()`

Retorna o número total de nós gerados a partir deste tableau. Este método é normalmente chamado a partir do tableau-raiz.

A.3.6 unsigned int `countFormulae()`

Retorna o número total de fórmulas geradas a partir deste tableau. Este método é normalmente chamado a partir do tableau-raiz.

A.3.7 bool `applyRule(unsigned int index, const vector<SignedFormula *>& in, const vector<SignedFormula *>& out)`

As regras de tableau são armazenadas no vetor `_rules`, que é um vetor de apontadores para função do seguinte tipo:

typedef bool `(*Rule)(const vector<SignedFormula *>&, vector<SignedFormula *>&)`

ou seja, uma regra é uma função que recebe dois vetores de fórmulas assinaladas e retorna um **bool**. O primeiro parâmetro é o vetor com as premissas da regra. As premissas devem ser fornecidas em ordem: o primeiro elemento deve ser a premissa primária, e o segundo, a premissa secundária (se houver). O segundo parâmetro é o vetor com as conclusões da regra. A função retorna **true** se a aplicação da regra é bem-sucedida, e `false`, caso contrário.

O vetor de regras `_rules` deve ser inicializado na construção do tableau.

O método `applyRule()` recebe o índice da regra a ser aplicada no vetor `_rules`, e as premissas e conclusões da regra, e procede à aplicação desta regra.

A.3.8 virtual void `createChild(const string& id, SignedFormula *fml) = 0`

Este método é responsável pela criação de subtableaux. Recebe o identificador do novo tableau, e a nova fórmula a ser incluída nele. É geralmente utilizado na bifurcação do tableau.

Note que este é um método virtual puro, sendo cada classe derivada responsável pela sua implementação.

A.4 Classe `TableauStrategy`

A classe `TableauStrategy` encapsula uma estratégia para um método de tableau. A declaração da classe `TableauStrategy` é mostrada na Figura A.5.

A.4.1 bool `init(const string& tableau_id, vector<SignedFormula *> *items, vector<SignedFormula *> *alphas, vector<SignedFormula *> *betas, vector<SignedFormula *> *lits)`

Este método faz a inicialização dos membros `_items`, `_alphas`, `_betas` e `_lits`. É geralmente chamado no construtor do tableau, e recebe como parâmetros os membros correspondentes do tableau.

Além disso, inicializa o membro `_mlits`, que faz a manutenção dos literais do tableau. Para um átomo `x`,

- se o tableau tem uma fórmula assinalada Fx, então `_mlits[x] = 1`;
- se o tableau tem uma fórmula assinalada Tx, então `_mlits[x] = 2`;
- se o tableau tem Fx e Tx, então `_mlits[x] = 3`.

Retorna `true` se o tableau está fechado, ou seja, se para algum átomo `x`, `_mlits[x] = 3`.

```
class TableauStrategy {
 public:
  TableauStrategy();
  virtual ~ TableauStrategy();

  bool init(const string& tableau_id,
    vector<SignedFormula *> *items,
    vector<SignedFormula *> *alphas,
    vector<SignedFormula *> *betas,
    vector<SignedFormula *> *lits);
  bool classify(unsigned int& index);
  virtual unsigned int chooseAlpha();
  virtual unsigned int chooseBeta();
  virtual int nextRule() = 0;

 protected:
  vector<SignedFormula *> *_items;
  vector<SignedFormula *> *_alphas;
  vector<SignedFormula *> *_betas;
  vector<SignedFormula *> *_lits;

  map<string, int> _mlits;
};
```

Figura A.5: Declaração da classe `TableauStrategy`.

A.4.2 bool `classify(unsigned int& index)`

Este método faz a classificação das fórmulas no vetor `_items`. Coloca as fórmulas do tipo α no vetor `_alphas`, as fórmulas do tipo β no vetor `_betas`, e os literais no vetor `_lits`. Retorna **true** se o tableau estiver fechado.

O parâmetro `index` indica a partir de que posição do vetor `_items` a classificação será feita. Isto evita que chamadas consecutivas deste método reclassifiquem fórmulas que já foram classificadas.

A.4.3 virtual unsigned int `chooseAlpha()`

Retorna o índice no vetor `_alphas` da próxima fórmula α a ser escolhida. Este método é virtual porque cada estratégia pode ter uma maneira diferente de escolher a próxima fórmula.

A.4.4 virtual unsigned int `chooseBeta()`

Retorna o índice no vetor `_betas` da próxima fórmula β a ser escolhida. Este método é virtual porque cada estratégia pode ter uma maneira diferente de escolher a próxima fórmula. [1]

A.4.5 virtual int `nextRule() = 0`

Este método indica qual a próxima regra a ser aplicada. O padrão para o valor de retorno é:

- -1, se não houver nenhuma regra a ser aplicada;
- 0, se for uma regra com uma premissa (aplicável sobre fórmula α);
- 1, se for uma regra com duas premissas (aplicável sobre fórmula β).

Valores de retorno maiores que 1 podem ser usados nas classes derivadas (como, por exemplo, na classe `KEStrategy`, que utiliza o valor 2 para aplicação do Princípio da Bivalência). [2]

[1] Na classe `KEStrategy` e derivadas, além do método `chooseBeta()`, existe o método `chooseLit()`, que retorna o índice no vetor `_lits` do literal que será usado como premissa secundária da regra.

[2] Na classe `KEStrategy`, existe o método `choosePB()`, que retorna uma fórmula sobre a qual será aplicado o Princípio da Bivalência.

É um método virtual puro. Cada classe derivada deve dar a sua implementação.

Apêndice B

Gráficos

Neste apêndice são exibidos os gráficos correspondentes a cada um dos testes realizados.

Para cada um dos testes temos dois gráficos: o primeiro mostra o número de nós gerados e o segundo mostra o número de fórmulas geradas.

Os gráficos são:

1. Gamma (nodes): na abscissa, o número do caso de teste Gamma; na ordenada, o número de nós gerados para o teste;
2. Gamma (formulae): na abscissa, o número do caso de teste Gamma; na ordenada, o número de fórmulas geradas para o teste;
3. H (nodes): na abscissa, o número do caso de teste H; na ordenada, o número de nós gerados para o teste;
4. H (formulae): na abscissa, o número do caso de teste H; na ordenada, o número de fórmulas geradas para o teste;
5. PHP (nodes): na abscissa, o número do caso de teste PHP; na ordenada, o número de nós gerados para o teste;
6. PHP (formulae): na abscissa, o número do caso de teste PHP; na ordenada, o número de fórmulas geradas para o teste;
7. Statman (nodes): na abscissa, o número do caso de teste Statman; na ordenada, o número de nós gerados para o teste;

8. Statman (formulae): na abscissa, o número do caso de teste Statman; na ordenada, o número de fórmulas geradas para o teste;

9. Gamma clausal (nodes): na abscissa, o número do caso de teste Gamma na forma clausal; na ordenada, o número de nós gerados para o teste;

10. Gamma clausal (formulae): na abscissa, o número do caso de teste Gamma na forma clausal; na ordenada, o número de fórmulas geradas para o teste;

11. H clausal (nodes): na abscissa, o número do caso de teste H na forma clausal; na ordenada, o número de nós gerados para o teste;

12. H clausal (formulae): na abscissa, o número do caso de teste H na forma clausal; na ordenada, o número de fórmulas geradas para o teste;

13. PHP clausal (nodes): na abscissa, o número do caso de teste PHP na forma clausal; na ordenada, o número de nós gerados para o teste;

14. PHP clausal (formulae): na abscissa, o número do caso de teste PHP na forma clausal; na ordenada, o número de fórmulas geradas para o teste;

15. Statman clausal (nodes): na abscissa, o número do caso de teste Statman na forma clausal; na ordenada, o número de nós gerados para o teste;

16. Statman clausal (formulae): na abscissa, o número do caso de teste Statman na forma clausal; na ordenada, o número de fórmulas geradas para o teste;

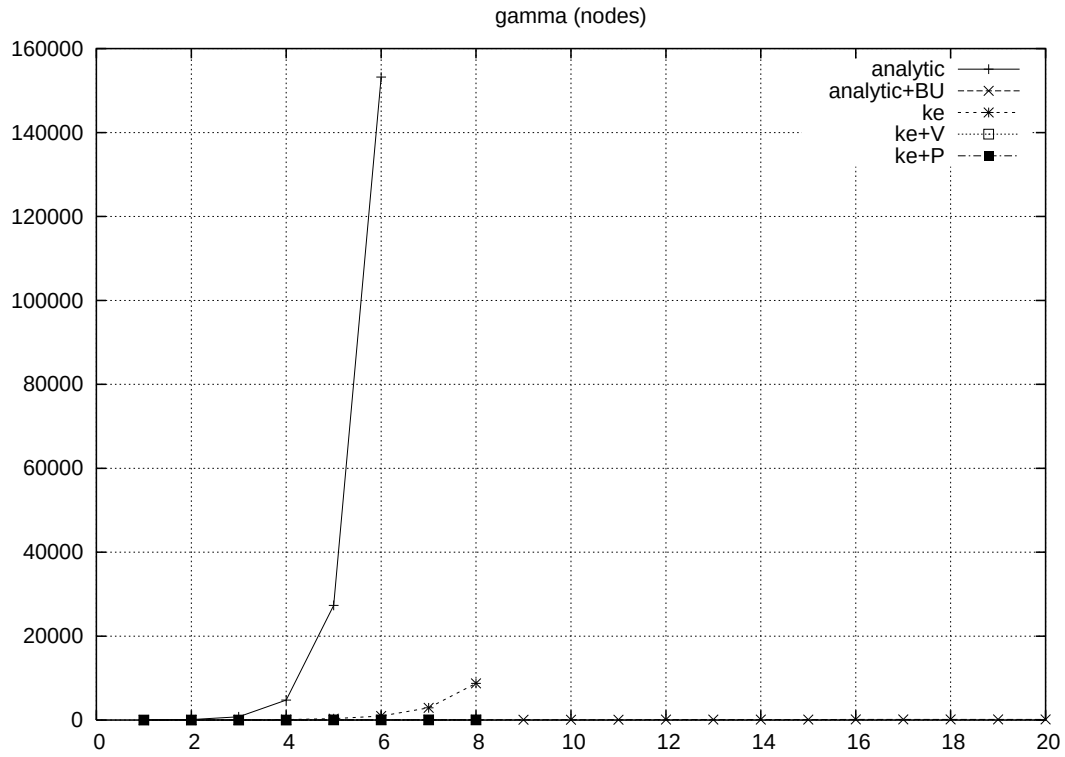
17. Gamma clausal w/ irrelevant formulae (nodes): na abscissa, o número do caso de teste Gamma na forma clausal com fórmulas irrelevantes; na ordenada, o número de nós gerados para o teste;

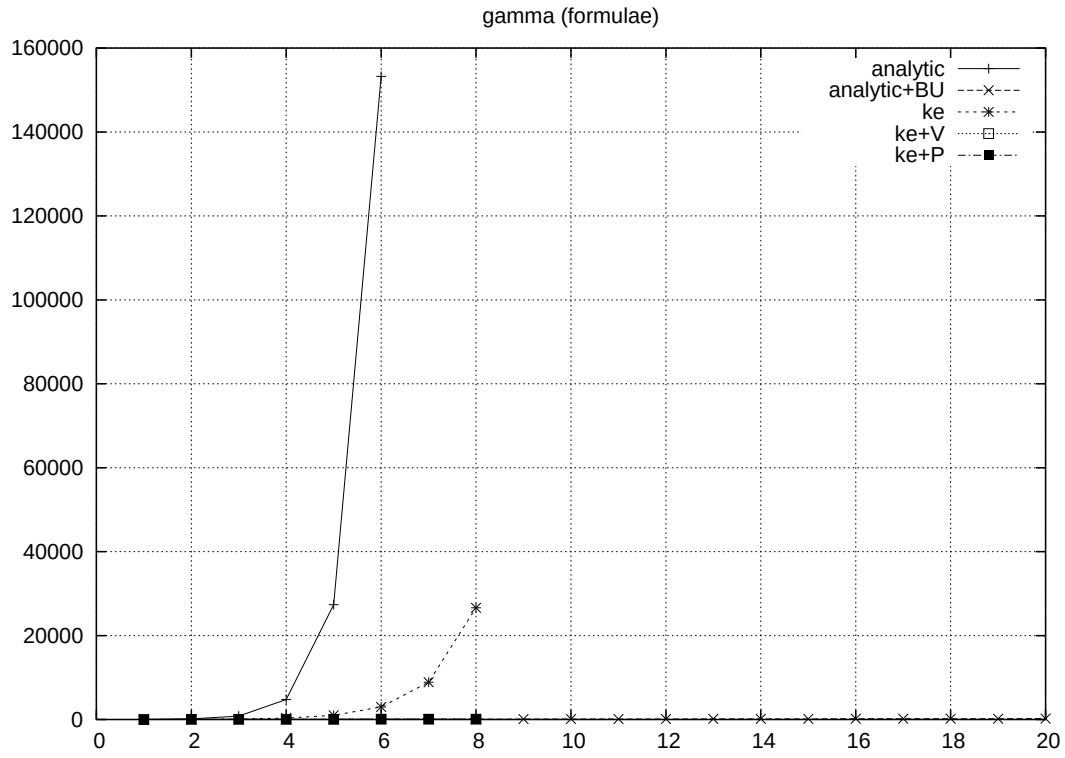
18. Gamma clausal w/ irrelevant formulae (formulae): na abscissa, o número do caso de teste Gamma na forma clausal com fórmulas irrelevantes; na ordenada, o número de fórmulas geradas para o teste.

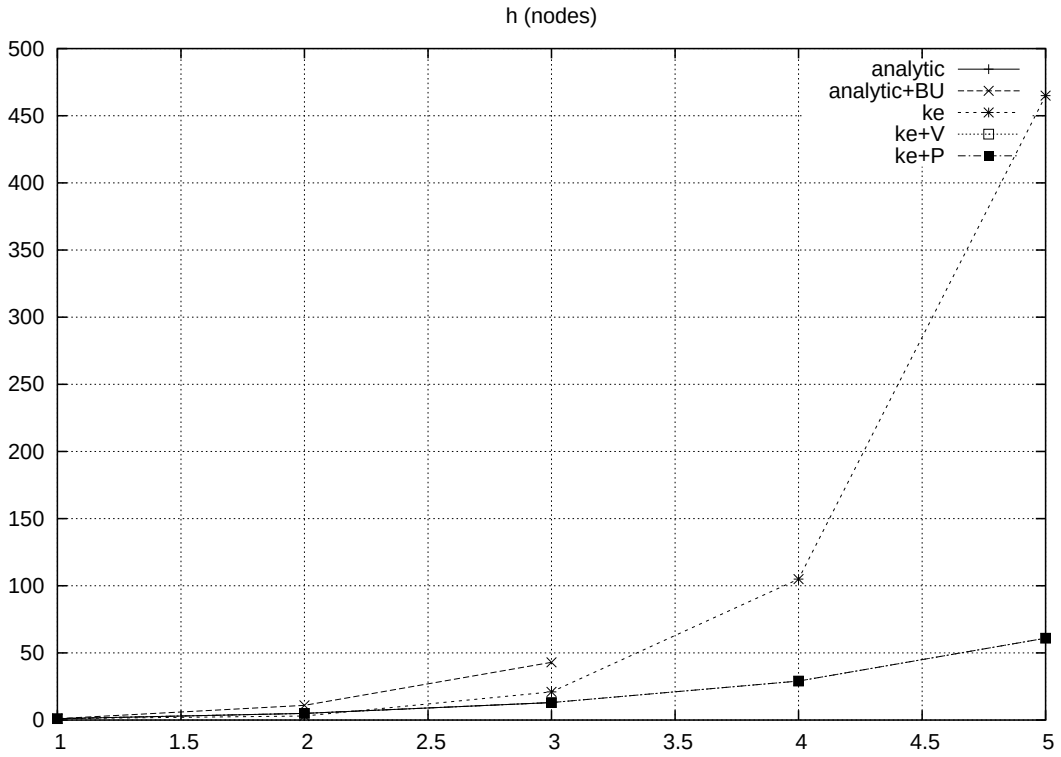
19. Gamma clausal w/ irrelevant formulae w/ distance (nodes): na abscissa, o número do caso de teste Gamma na forma clausal com fórmulas irrelevantes; na ordenada, o número de nós gerados para o teste com a heurística da distância;

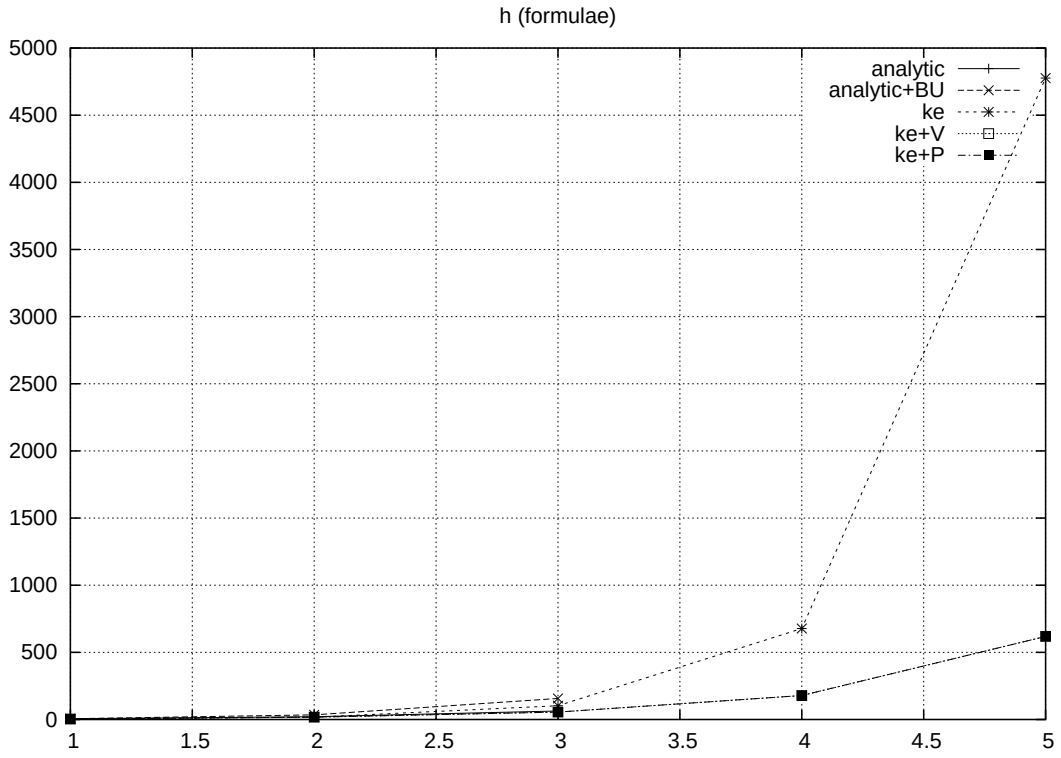
20. Gamma clausal w/ irrelevant formulae w/ distance (formulae): na abscissa, o número do caso de teste Gamma na forma clausal com fórmulas irrelevantes;

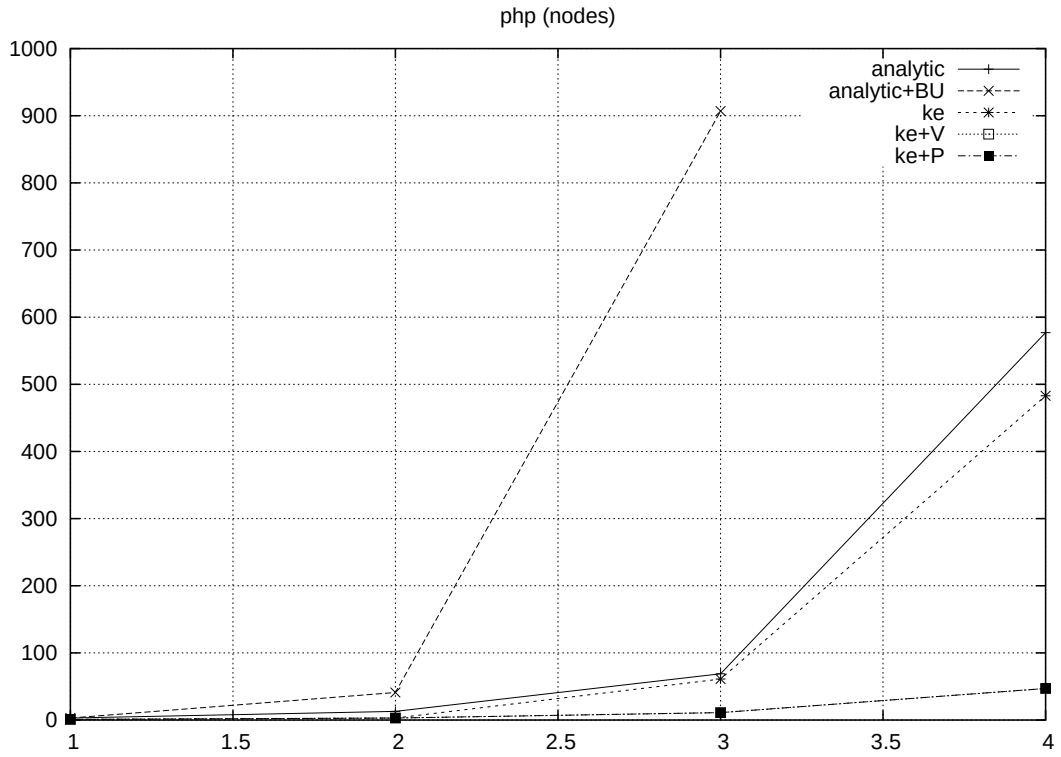
na ordenada, o número de fórmulas geradas para o teste com a heurística da distância.

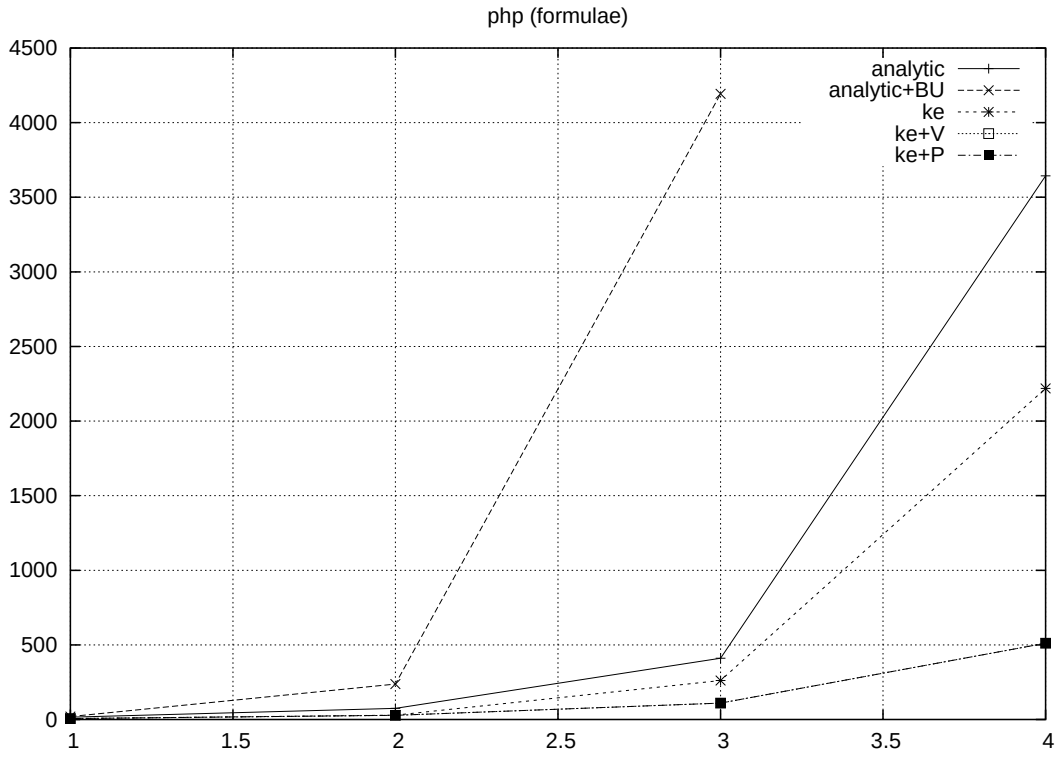

gamma (nodes)
160000
140000
120000
100000
80000
60000
40000
20000
0
0
2
4
6
8
10
12
14
16
18
20
analytic
analytic+BU
ke
ke+V
ke+P

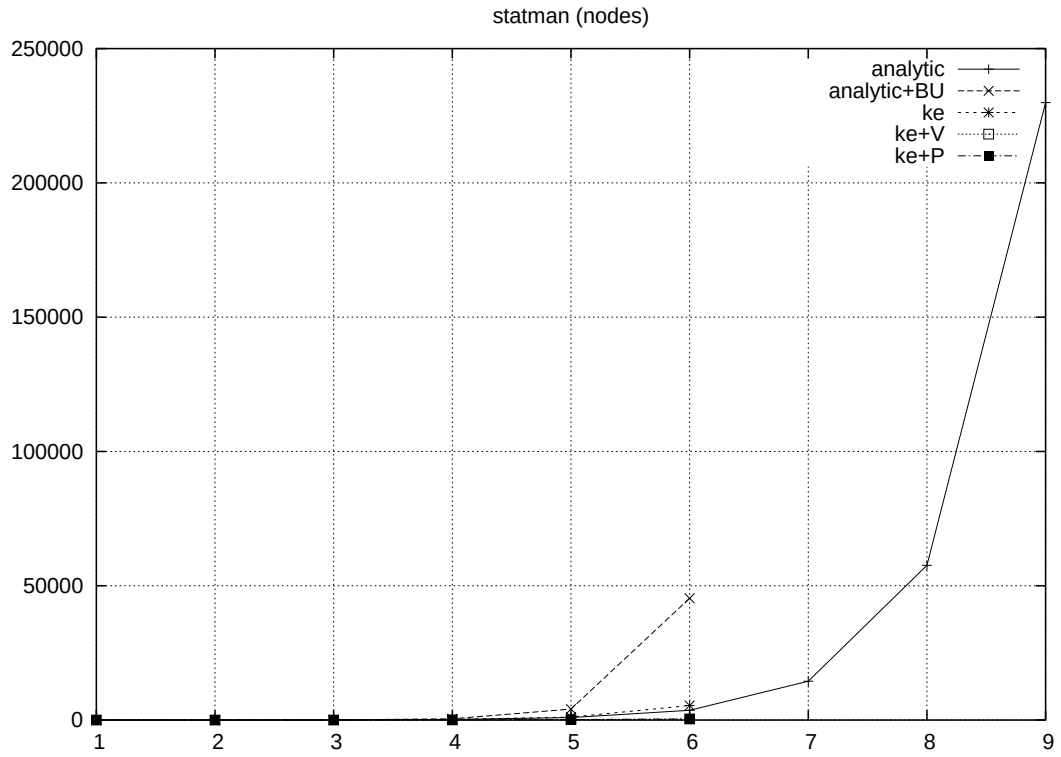

gamma (formulae)
160000
140000
120000
100000
80000
60000
40000
20000
0
0
2
4
6
8
10
12
14
16
18
20
analytic
analytic+BU
ke
ke+V
ke+P

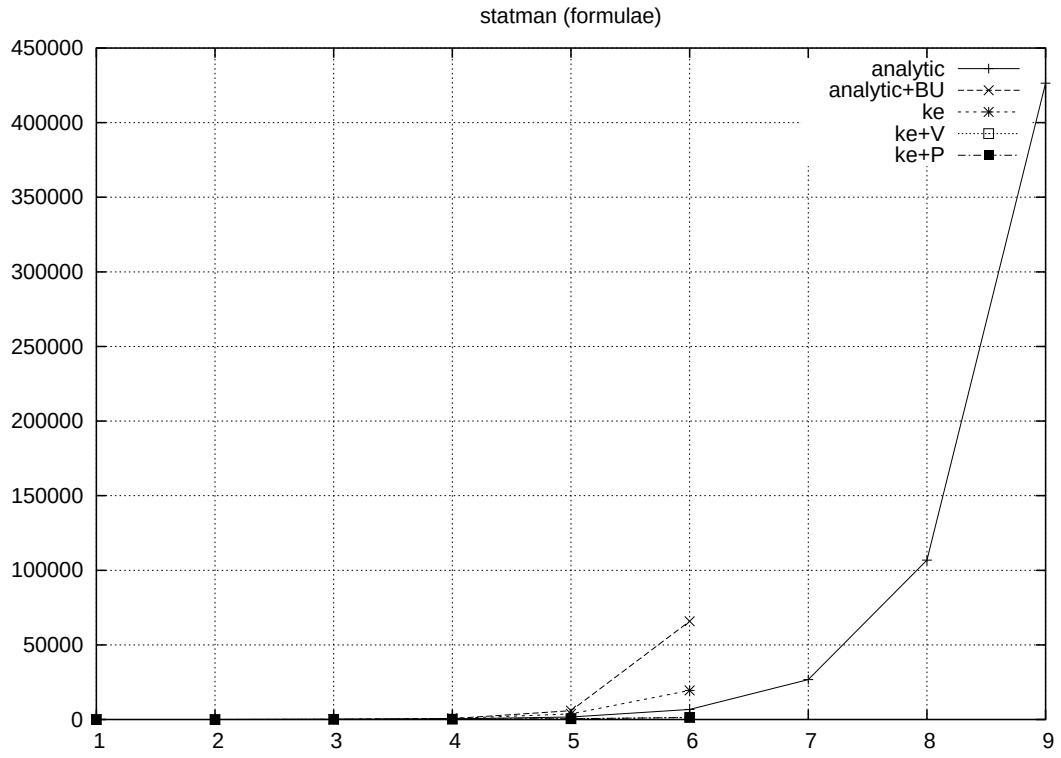

h (nodes)
analytic
analytic+BU
ke
ke+V
ke+P
500
450
400
350
300
250
200
150
100
50
0
1
1.5
2
2.5
3
3.5
4
4.5
5

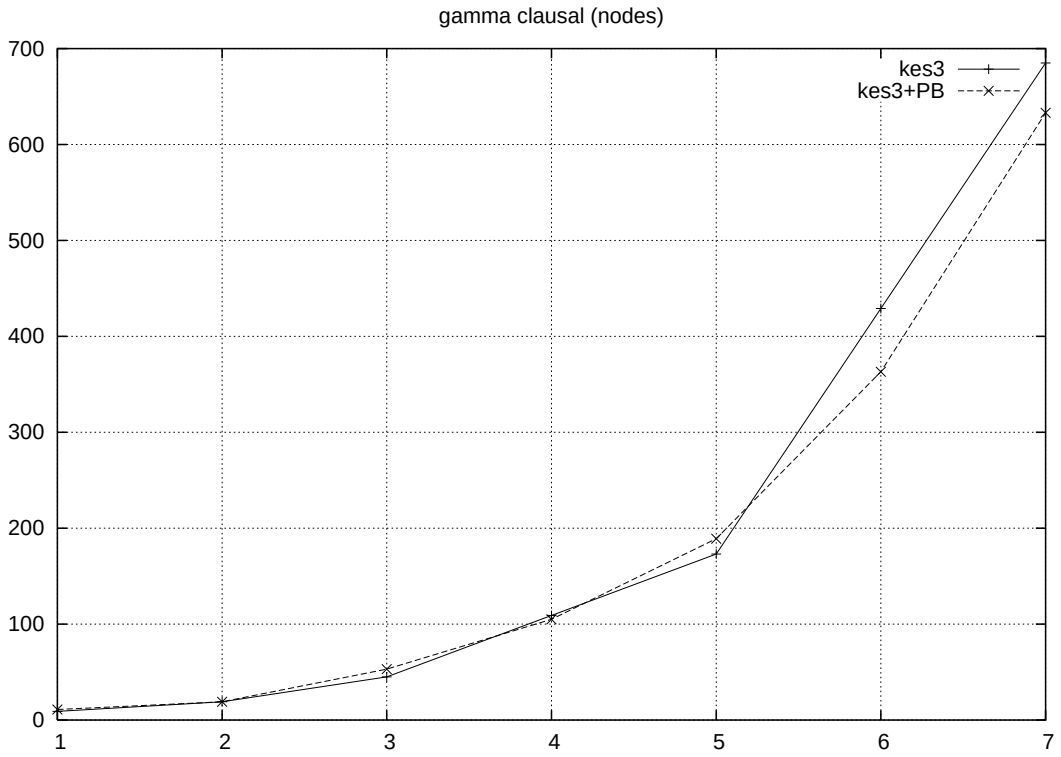

h (formulae)
analytic
analytic+BU
ke
ke+V
ke+P
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
1.5
2
2.5
3
3.5
4
4.5
5

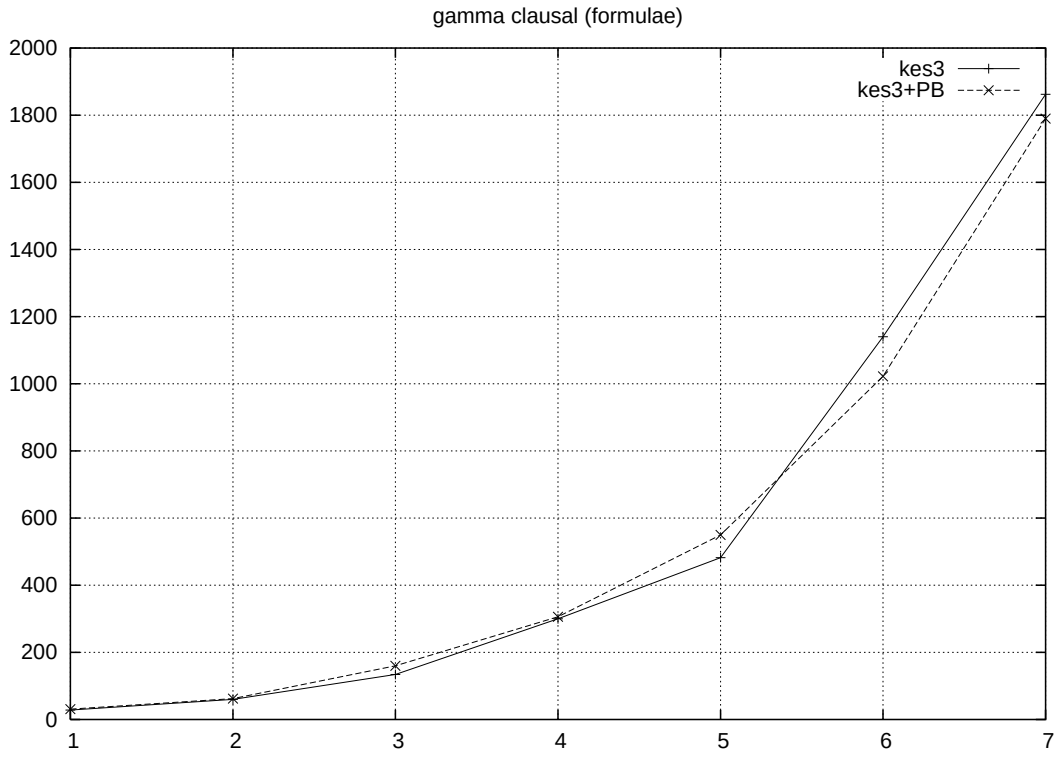

php (nodes)
analytic
analytic+BU
ke
ke+V
ke+P
1000
900
800
700
600
500
400
300
200
100
0
1
1.5
2
2.5
3
3.5
4

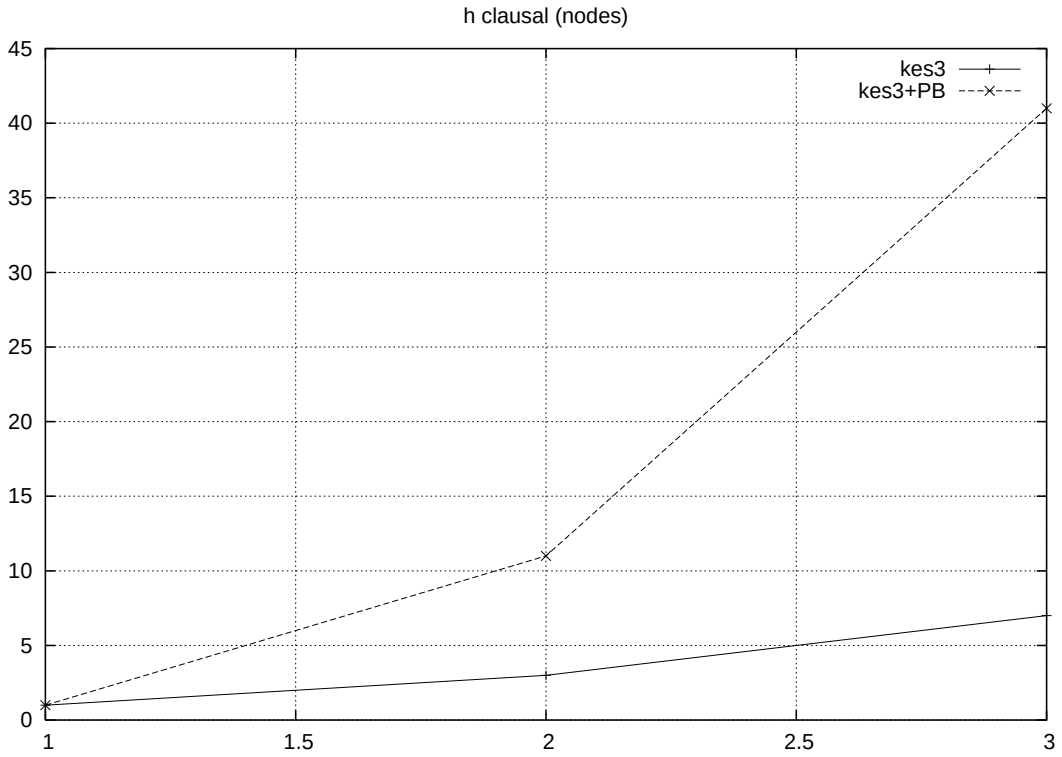

php (formulae)
analytic
analytic+BU
ke
ke+V
ke+P
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
1.5
2
2.5
3
3.5
4

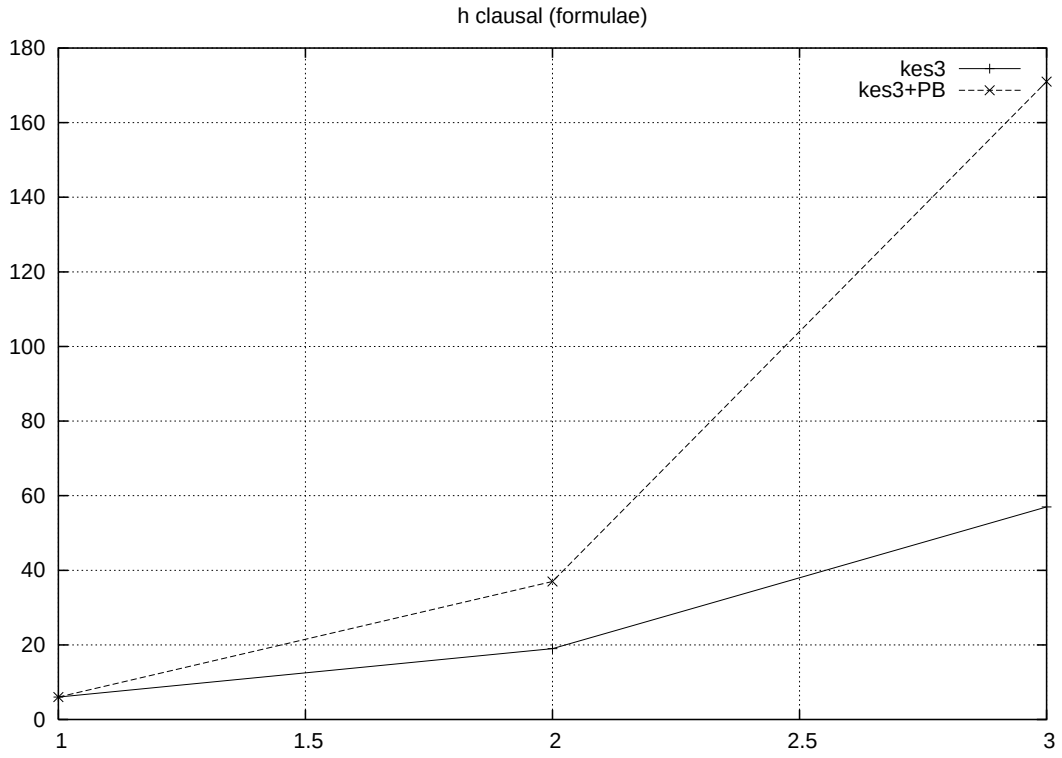

statman (nodes)
250000
200000
150000
100000
50000
0
1
2
3
4
5
6
7
8
9
analytic
analytic+BU
ke
ke+V
ke+P

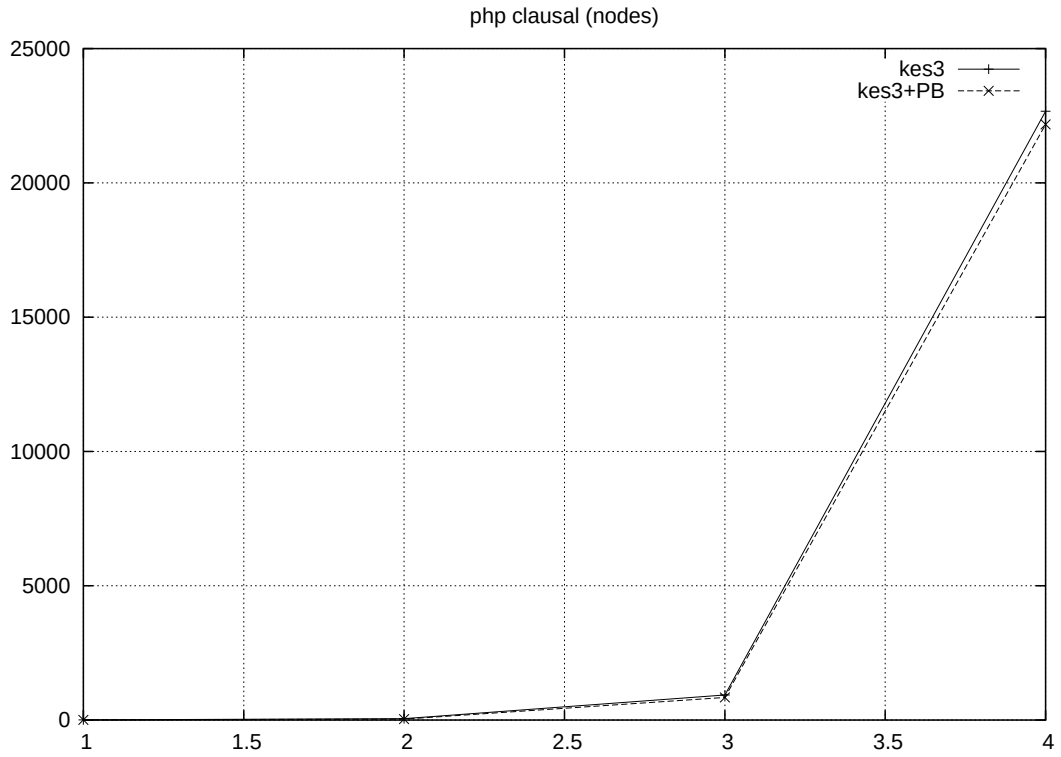

statman (formulae)
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
1
2
3
4
5
6
7
8
9
analytic
analytic+BU
ke
ke+V
ke+P

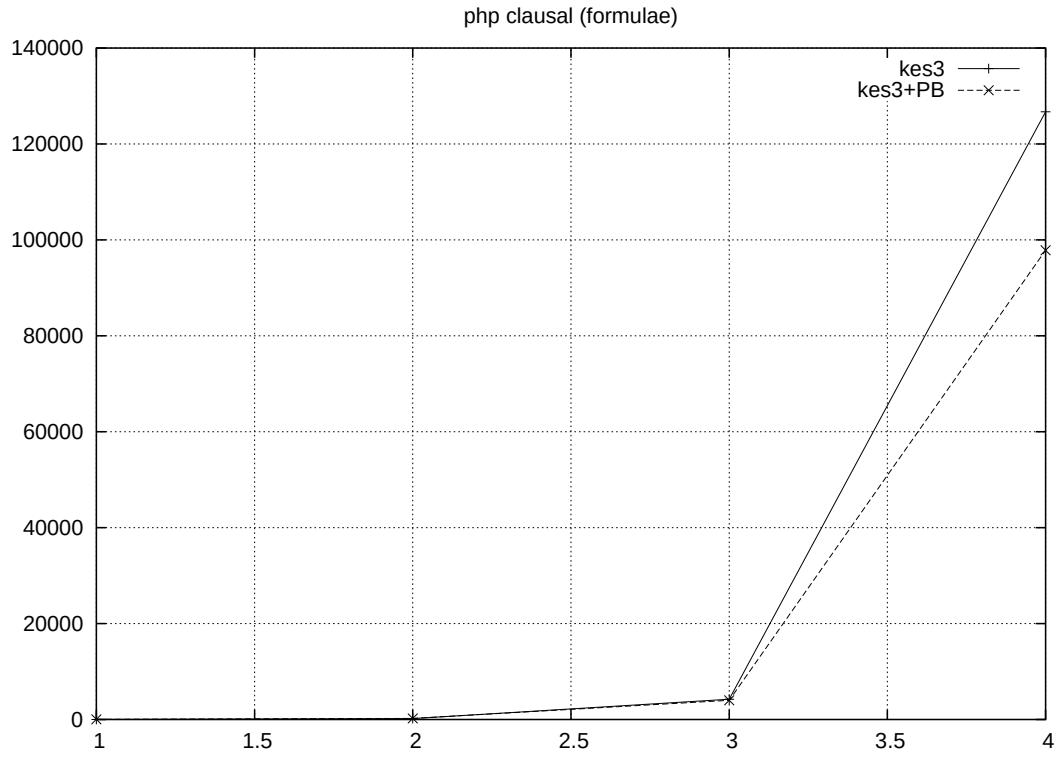

gamma clausal (nodes)
kes3
kes3+PB
700
600
500
400
300
200
100
0
1
2
3
4
5
6
7

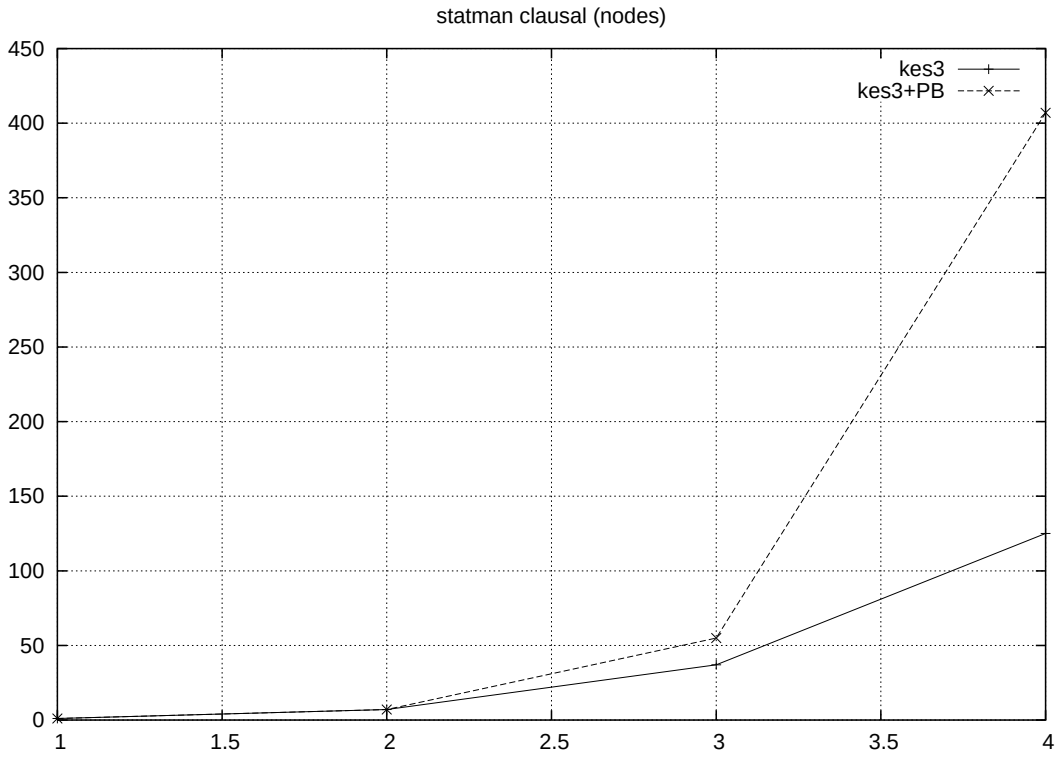

gamma clausal (formulae)
kes3
kes3+PB
2000
1800
1600
1400
1200
1000
800
600
400
200
0
1
2
3
4
5
6
7

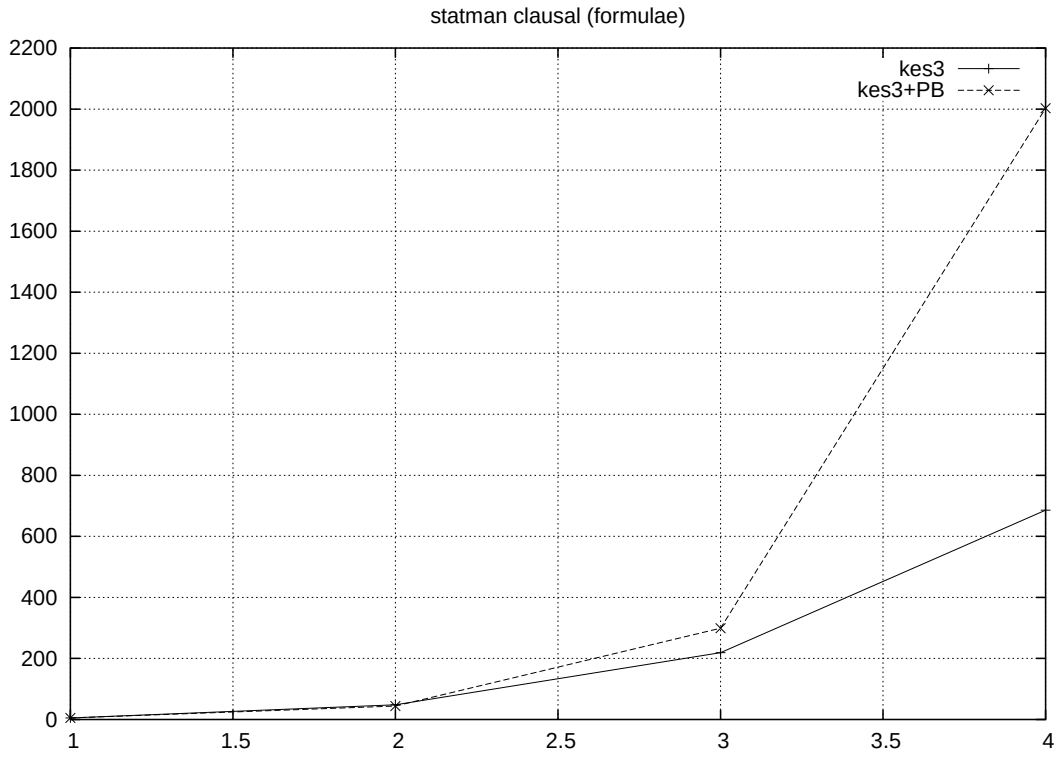

h clausal (nodes)
kes3
kes3+PB
45
40
35
30
25
20
15
10
5
0
1
1.5
2
2.5
3

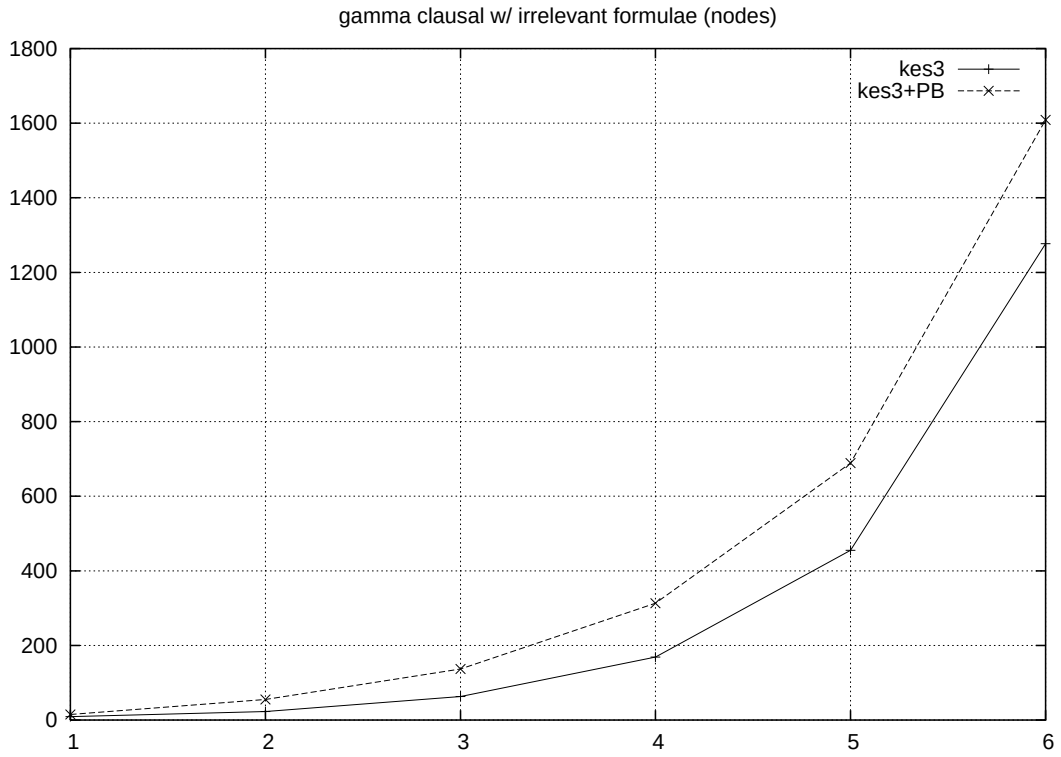

h clausal (formulae)
kes3
kes3+PB
180
160
140
120
100
80
60
40
20
0
1
1.5
2
2.5
3

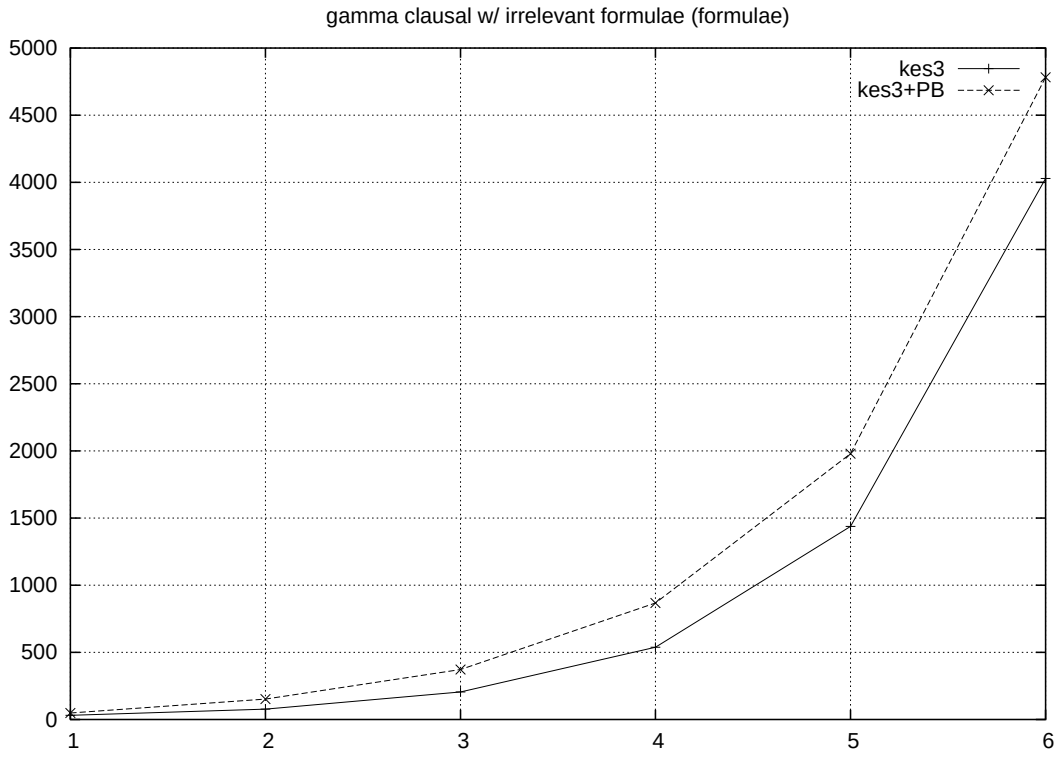

php clausal (nodes)
kes3
kes3+PB
25000
20000
15000
10000
5000
0
1
1.5
2
2.5
3
3.5
4

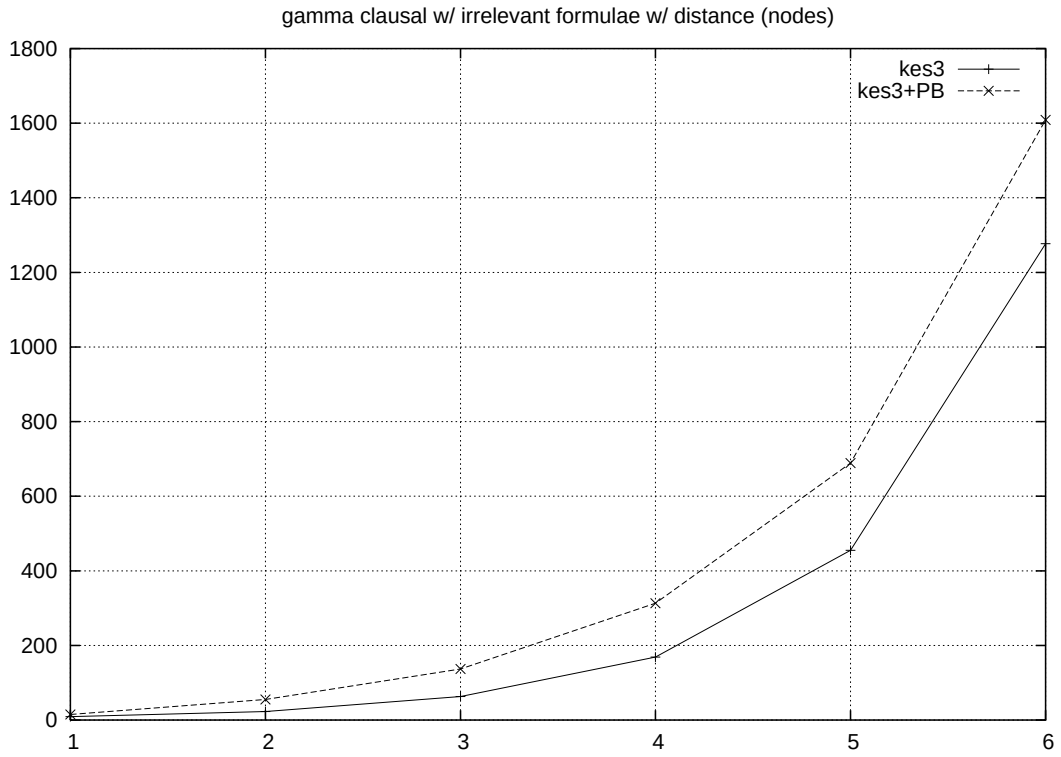

php clausal (formulae)
kes3
kes3+PB
140000
120000
100000
80000
60000
40000
20000
0
1
1.5
2
2.5
3
3.5
4

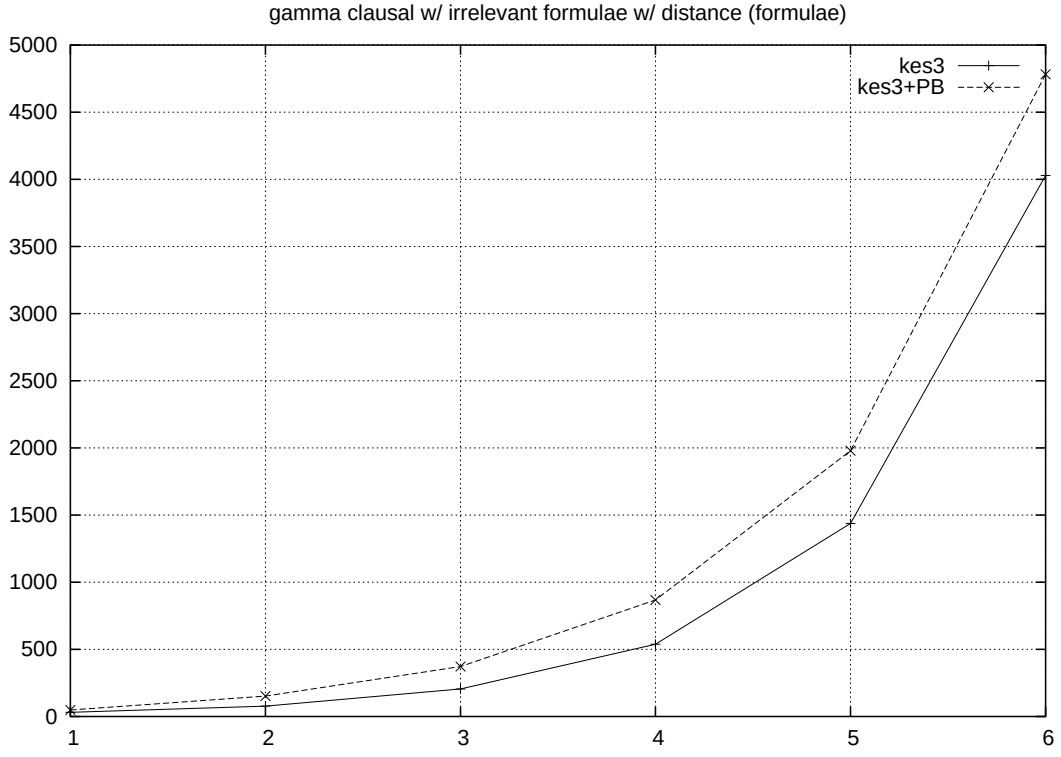

statman clausal (nodes)
kes3
kes3+PB
450
400
350
300
250
200
150
100
50
0
1
1.5
2
2.5
3
3.5
4


statman clausal (formulae)
kes3
kes3+PB
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
1
1.5
2
2.5
3
3.5
4


gamma clausal w/ irrelevant formulae (nodes)
kes3
kes3+PB
1800
1600
1400
1200
1000
800
600
400
200
0
1
2
3
4
5
6


gamma clausal w/ irrelevant formulae (formulae)
kes3
kes3+PB
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
2
3
4
5
6


gamma clausal w/ irrelevant formulae w/ distance (nodes)
kes3
kes3+PB
1800
1600
1400
1200
1000
800
600
400
200
0
1
2
3
4
5
6


gamma clausal w/ irrelevant formulae w/ distance (formulae)
kes3
kes3+PB
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
2
3
4
5
6

Referências Bibliográficas

[BJ94] K. Beck and R. Johnson. Patterns generate architectures. In *Proceedings of ECOOP 94*, pages 139–149, 1994.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, McGraw-Hill Book Company, 1990.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 1971.

[CS00] Alessandra Carbone and Stephen Semmes. *A Graphic Apology for Symmetry and Implicitness*. Oxford Mathematical Monographs. Oxford Science Publications, 2000.

[D'A90] Marcello D'Agostino. Investigations into the complexity of some propositioal calculi. Master's thesis, Oxford University Computing Laboratory, Programming Research Group, 1990.

[D'A92] Marcello D'Agostino. Are tableaux an improvement on truth-tables? — cut-free proofs and bivalence. *Journal of Logic, Language and Information*, 1:235–252, 1992.

[End99] Ulrich Endriss. A time efficient ke based theorem prover. In N. V. Murray, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, Tableaux'99, Proceedings*, volume 1617 of *LNAI*, pages 313–318. Springer-Verlag, June 1999.

[FHLS98] Garry Froehlich, H. James Hoover, Ling Liu, and Paul Sorenson. Using object-oriented frameworks. In *CRC Handbook of Object Technology*. CRC Press, 1998.

[FW01] Marcelo Finger and Renata Wassermann. Tableaux for approximate reasoning. In *Proceedings of the IJCAI Workshop on Inconsistency in Data and Knowledge*, Seattle, Washington, USA, August 2001.

[FW02a] Marcelo Finger and Renata Wassermann. Expressivity and control in limited reasoning. In *Proceedings of ECAI 2002*, 2002.

[FW02b] Marcelo Finger and Renata Wassermann. Logics for approximate reasoning: Approximating classical logic "from above". In *Proceedings of SBIA 2002*, 2002.

[JF88] R. Johnson and B. Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, 1(2):22–35, 1988.

[Kow79] Robert Kowalski. *Logic for Problem Solving.* North-Holland, 1979.

[SC95] Marco Schaerf and Marco Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74(2):249–310, 1995.

[Smu68] Raymond M. Smullyan. *First-Order Logic.* Springer-Verlag, 1968.

[Sta78] Richard Statman. Bounds for proof-search and speed-up in predicate calculus. *Annals of Mathematical Logic*, 15:225–287, 1978.