PROVA AUTOMATIZADA DE TEOREMA EM LÓGICA PROPOSICIONAL

AUTOMATED THEOREM PROVING IN PROPOSITIONAL LOGIC

Frederico Martins Biber Sampaio[1]

Moisés Henrique Ramos Pereira[2] (Orientador)

Miriam Lourenço Maia[3] (Coorientadora)

Centro Universitário de Belo Horizonte, Belo Horizonte, MG

[1]fredmbs@gmail.com; [2] moiseshrp+unibh@gmail.com; [3] miriam.maia@prof.unibh.br

RESUMO: O artigo apresenta os fundamentos teóricos e a estrutura de um software de prova de teoremas em lógica proposicional elaborado com objetivos acadêmicos e didáticos, utilizando métodos de prova baseados em tableau semântico.
PALAVRAS-CHAVE: Lógica, lógica proposicional, inteligência artificial, prova de teorema, compilador, linguagem de domínio específico.

ABSTRACT: The paper presents the theoretical foundations and structure of a theorem proving software in propositional logic developed with academic and didactics objectives, using proof methods based on semantic tableau method.
KEYWORDS: Logic, propositional logic, artificial intelligence, automated theorem proving.

---

1 INTRODUÇÃO

A lógica é uma área de estudo ligada à filosofia que tem como objetivo geral a validação do raciocínio em termos de verdade ou falsidade. Lógica é um tema antigo com vários ramos, aplicações e formalizações. É fundamental em praticamente todas as ciências, incluindo a ciência da computação. Todos os computadores digitais modernos são fundamentados em lógica, em especial a lógica booleana. Em termos matemáticos, a lógica booleana é baseada na álgebra reduzindo o universo dos números ao conjunto {0, 1}. Com a simples observação que 1 pode ser tratado como verdade e 0 como falsidade, George Boole aproximou a matemática da lógica proposicional clássica, possibilitando o tratamento algébrico, por meio simbólico, e o cálculo proposicional.

No geral, a lógica proposicional é expressa em termos de verdade ou falsidade das proposições. As inferências relacionam as proposições, também resultando em verdade ou falsidade. As inferências básicas são “e”, “ou” e “não”. Uma fórmula proposicional é um conjunto de inferências e proposições. As proposições são os únicos elementos variáveis da lógica proposicional. Cada proposição pode ser interpretada em diferentes contextos como verdadeiro ou falso. Uma interpretação de uma fórmula é um conjunto específico de interpretação de suas proposições. Por exemplo, numa fórmula “A ou B”, A e B sendo proposições do tipo “A = faz calor” e “B = chove”, A e B só podem assumir valores verdadeiro (1) ou falso (0). Assim, no exemplo, todas as interpretações possíveis formam o conjunto {{1,1},{1,0},{0,1},{0,0}}. No presente artigo, o termo fórmula será usado no sentido de fórmula lógica que resulta em verdadeiro ou falso.

Considerando a lógica proposicional, um problema clássico é se uma fórmula possui uma ou mais interpretações que satisfaçam determinados objetivos. Sem formalismos, a satisfação de uma fórmula é se

seu resultado é verdadeiro em uma interpretação presumivelmente verdadeira. Por exemplo, caso se espere que todas as interpretações possíveis sejam verdadeiras (tautologia), toda e qualquer interpretação da fórmula deve resultar em verdade. Um caso mais específico é se uma fórmula satisfaz outra, ou se satisfaz a um conjunto de resultado esperado.

O problema da satisfação ("satisfatibilidade") de fórmulas lógicas, chamado de problemas SAT, é fundamental na ciência da computação. Por exemplo, o problema SAT de lógica booleana, tema do Teorema de Cook que criou as classificações de complexidade algorítmica adotadas atualmente, é considerado o primeiro problema NP-completo. Um problema é da classe NP (Não Polinomial) quando sua ordem de crescimento, ou complexidade, é maior que a polinomial. Por exemplo, sendo k uma constante e n o número de entradas de um algoritmo, uma complexidade da ordem $O(n!)$ ou $O(k^n)$ possuem crescimento maior que uma complexidade polinomial que, normalmente, é da ordem $O(n^k)$.

O tema de complexidade algorítmica foge ao escopo de presente trabalho, mas cabe uma introdução geral. Os computadores atuais são capazes de executar milhares de operações por segundo. Contudo, certos algoritmos exigem um aumento muito grande de operações a cada vez que se aumenta o número n de entradas (dados) que o computador deve tratar. Essa taxa de aumento é considerada como ordem de complexidade. Uma complexidade algorítmica muito elevada pode inviabilizar a solução de problemas, mesmo considerando poucas entradas. Apesar de ser um tema controverso, no geral, os computadores tratam com eficiência e proporcionalidade de tempo os problemas com complexidade polinomial (P).

Por isso, a classe de problemas NP-completos é especialmente interessante na computação, pois, mesmo que exista um algoritmo para sua solução, a execução prática exige um tempo computacional que inviabiliza a solução automática para a maioria dos problemas concretos. As pesquisas nessa área buscam técnicas para contornar essa limitação conceitual e reduzir a complexidade algorítmica desses tipos de problemas. Contudo, a redução de problemas NP para problemas P (NP=P) ainda não foi encontrada e, mesmo que ainda não exista prova formal, é consenso que tal redução é impossível.

Assim como outros problemas NP-completos, o problema SAT desperta enorme interesse em pesquisa. Especificamente, o problema SAT envolve várias aplicações práticas de grande interesse, tais como, apenas para citar alguns poucos exemplos, a simulação de raciocínio em agentes inteligentes, busca em bases de conhecimento, ferramenta de verificação e prova de teoremas matemáticos, verificação de circuitos lógicos, análise, otimização e validação de programas de computadores, etc.

O objetivo geral do trabalho é fornecer ferramentas e conceitos práticos para o estudo em lógica, em especial para alunos da disciplina de matemática discreta. O foco do trabalho é a comunidade acadêmica. O objetivo específico é elaborar um software que seja capaz de elaborar soluções para problemas SAT para atuar como ferramenta de ensino e pesquisa na área de lógica.

Para atingir os objetivos, o software é composto de dois módulos principais: (1) compilador para uma linguagem que expresse as fórmulas e o sistema lógico a ser tratado; (2) módulos de prova. O software trata de linguagem em lógica de predicado, especificamente a lógica de primeira ordem. Porém, o escopo dos algoritmos de prova é restrito à lógica proposicional.

Ao longo da elaboração do trabalho, vários temas e técnicas interessantes e relevantes foram sendo revelados. Sempre que possível, eles serão citados como estímulo para pesquisas futuras.

2 FUNDAMENTOS TEÓRICOS

2.1 SISTEMAS LÓGICOS

Não é escopo deste artigo a descrição das diversas provas de correção (“soundness”) e completude (“completeness”) dos algoritmos ou métodos lógicos adotados na elaboração do software, pois existe vasta bibliografia amplamente disponível e o conteúdo do artigo ficaria muito longo. Contudo, os conceitos de correção e completude são importantes para um software de prova de teoremas.

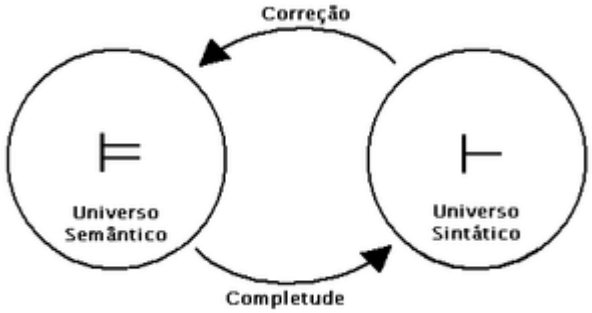

Figura 1 - Relação no cálculo lógico (WIKI-C)

No geral, o universo semântico de uma sentença ou fórmula lógica é obtido pela sua interpretação. Ou seja, o universo semântico está relacionado com as possíveis combinações de valores das variáveis do sistema e seu resultado. O resultado de um sistema lógico é sempre e apenas verdadeiro ou falso. Por outro lado, os valores que as variáveis podem assumir dependem da lógica adotada. Na lógica proposicional, o único elemento variável é a proposição e esses valores possíveis também são verdadeiro ou falso.

Já o universo sintático é obtido pelas derivações, ou seja, pela aplicação das inferências sobre os elementos simbólicos, sem considerar os resultados concretos. Por exemplo, independente de se saber qualquer valor de A ou B, a fórmula “não A e não B” pode ser inferida em “A ou B”.

O problema SAT está relacionado com ambos os universos. Contudo, os métodos de prova simbólicos, como as provas inferenciais sequenciais, a dedução natural ou tableau semântico são vinculados ao universo sintático dos sistemas lógicos. Já os métodos de interpretação ou busca de combinação de valores de variáveis, como a tabela verdade, estão vinculados ao universo semântico. Ambos os métodos, semânticos e sintáticos, são estudados em matemática discreta.

A questão relevante é: considerando-se um sistema lógico, qual é a melhor forma de verificação SAT? Além disso, para o mesmo sistema lógico em consideração, uma solução SAT por método sintático é equivalente a uma solução por método semântico?

Considere-se φ (fi) como uma fórmula lógica e Γ (GAMA) como uma lista de uma ou mais fórmulas separadas por vírgula.

Um sistema lógico é completo se $\Gamma \vDash \varphi \rightarrow \Gamma \vdash \varphi$ e é correto (provável) se $\Gamma \vdash \varphi \rightarrow \Gamma \vDash \varphi$. Essa relação é ilustrada na figura 1. Um sistema lógico pode ser generalizado para $\Gamma \vdash \Delta$ ou $\Gamma \vDash \Delta$, sendo Δ (DELTA) uma lista de fórmulas.

No caso da lógica de primeira ordem, o teorema da completude de Gödel, de 1929, prova que as formulas são completas e corretas. Apesar disso, a lógica de primeira ordem é considerada, em tese, *indecidível*.

> *Its undecidability, established by Church and Turing in the 1930’s, made it an ‘intrinsically’ interesting subject forever. If no decision procedure can be found, that is all decision procedures have to be partial, the problem area is in no danger of saturation: it is always possible to find ‘better’ methods (at least for certain purposes).*
>
> (D’AGOSTINO, et al., 1999, pág. 45)

Para a lógica proposicional, além de poder ser considerada com subconjunto da lógica de primeira ordem, existem diversas provas de sua completude e correção. Além disso, ela é considerada *decidível* desde o surgimento do algoritmo da tabela verdade no início de 1920.

Já para as lógicas de ordens superiores, os teoremas da incompletude de Gödel, ou teoremas da *indecidibilidade*, demonstram que a relação entre o universo semântico e o sintático não se mantém.

Daí, nas lógicas de primeira ordem, incluindo a lógica proposicional, uma prova por dedução equivale a uma demonstração por interpretação. Ou seja, um software que auxilie na demonstração SAT pode utilizar ambas as abordagens: prova simbólica ou métodos interpretativos. No contexto do presente artigo, o termo prova será usado no sentido de avaliação sintática de sistemas lógicos.

2.2 MÉTODOS DE PROVA

Considerando-se o objetivo de elaborar um software de prova que auxilie o meio acadêmico, é necessário considerar métodos simbólicos. Contudo, a elaboração de programas de prova por métodos simbólicos é uma tarefa complexa e propensa a erros, como foi comprovada pela prática do presente trabalho. Por isso, o software elaborado incluiu a tabela verdade para verificação das provas em lógica proposicional. Este artigo não fará maiores descrições da tabela-verdade, pois é um tema básico tratado no curso de matemática discreta.

Outro objetivo inicial, porém secundário, do software era elaborar provas mais legíveis e de menor tamanho possível. Ou seja, provas mais simples e diretas dentro do possível. Infelizmente, os métodos automatizados nem sempre atendem esse objetivo. Existem vários métodos de prova de sistema lógicos. Qual método seria o mais adequado para o objetivo do trabalho? Essa questão foi a mais complexa durante a pesquisa e fundamentação teórica para a elaboração do software.

O primeiro método considerado nas pesquisas foi o algoritmo DPLL (Davis-Putnam-Logemann-Loveland), amplamente citado em teses, artigos e usado em diversos softwares SAT. O DPLL é um método simbólico. Contudo, o DPLL trabalha com fórmulas na forma conjuntiva normal (CNF - *Conjunctive Normal Form*), que é muito eficiente para tratamento computacional, mas não atende aos objetivos de legibilidade pretendidos no presente trabalho. O objetivo não é elaborar um software focado no máximo desempenho e capaz de competir em concursos SAT.

Outra linha de pesquisa foram os algoritmos de dedução natural. Porém, os métodos automatizados de dedução natural dificilmente geram fórmulas reduzidas e diretas devido ao enorme espaço de busca originado das possibilidades combinatórias das diversas inferências. Por exemplo, uma das principais dificuldades em se reduzir o espaço de busca na dedução sequencial é a possibilidade das inferências acrescentarem novas fórmulas mais complexas que as anteriores na prova. Geralmente é necessário coletar parte das fórmulas já deduzidas para se inferir novas fórmulas até igualar ao resultado da conclusão esperada para a prova. No final das pesquisas do trabalho, uma heurística de prova “orientada a objetivos” (*goal-oriented*) se mostrou promissora para atingir o objetivo de simplicidade e legibilidade do resultado final da prova automática, em especial para as provas sequenciais adotadas na maioria dos cursos de matemática discreta.

A busca por técnicas e heurísticas para reduzir o tamanho das provas ou torná-las mais “legíveis” descortinou uma vastidão de material e abordagem nessa área de pesquisa que, apesar de parecer árida, possui um grande interesse acadêmico e muitas aplicações práticas relevantes.

2.3 TABLEAU SEMÂNTICO

Uma das técnicas de prova pesquisada foi o tableau semântico. O método é baseado na construção de uma árvore de prova (*proof-tree*). Apesar da estrutura da prova ser diferente das provas sequenciais, o resultado final é razoavelmente legível. Além disso, os algoritmos de tableau são relativamente diretos, pois

são baseados em inferências de simplificação ou eliminação. Ao não utilizar inferências aditivas, como normalmente ocorre em outras formas de dedução, a solução final é garantida para a lógica proposicional.

Outra vantagem do método é que pode ser aplicado a vários tipos diferentes de lógicas. Isso o iguala aos métodos dedutivos, como os sequenciais ou os naturais, mas é uma vantagem em relação a vários outros métodos SAT mais específicos, como o DPLL. Existem diversas teses demonstrando a aplicação de tableau a diversos tipos de lógica.

É natural que os diversos métodos de prova se intercalem e se integrem. Contudo, as pesquisas apontaram para uma grande capacidade de inter-relacionamento do método tableau com os vários outros métodos de prova.

Por fim, o método também conta com uma enorme variabilidade de técnicas e heurísticas de redução da árvore de prova. No presente trabalho, serão abordados os tableaux de Smullyan, o tableau com lemas e o tableau KE, todos apresentados em (D'AGOSTINO, et al., 1999), a principal referência adotada para elaborar o presente artigo.

2.3.1 FUNDAMENTOS DO MÉTODO TABLEAU

O método tableau é baseado na prova por contradição, ou seja, na busca de um contraexemplo (contramodelo). Considerando o sistema lógico, a prova ocorre se $\Gamma \vDash \varphi$, então $\Gamma \cup \{\neg\varphi\} \vDash \bot$. No caso, $\{\neg\varphi\}$ representa um contramodelo.

A construção da solução ocorre em forma de árvore. Considerando o sistema lógico $\Gamma \vdash \Delta$, na cabeça da árvore (*head*) são colocadas as hipóteses verdadeiras (Γ) e as falsas (Δ). Cada nó (*node*) da árvore representa uma fórmula, incluindo as hipóteses ou suas derivações por regras de inferência.

Como todo método de prova, tenta-se aplicar inferências sobre as fórmulas. Contudo só são aceitas inferências simples que reduzem as fórmulas. As regras de inferência mais gerais são:

$$\frac{A \wedge B}{\begin{array}{l}A\\B\end{array}} \qquad \frac{\neg(A \wedge B)}{\neg A \mid \neg B} \qquad \frac{A \vee B}{A \mid B} \qquad \frac{\neg(A \vee B)}{\begin{array}{l}\neg A\\\neg B\end{array}}$$

$$\frac{A \to B}{\neg A \mid B} \qquad \frac{\neg(A \to B)}{\begin{array}{l}A\\\neg B\end{array}} \qquad \frac{\neg\neg A}{A}$$

Figura 2 - Regras de inferência. Fonte - D'AGOSTINO, et al., 1999, pág. 57.

Cada aplicação das regras de inferência expande (aumentam) a árvore de prova. Por exemplo, as inferências de fórmulas disjuntivas (ou) criam um novo ramo (*branch*) na árvore. Cada ramo possui um nó folha (*leaf*). Independentemente da posição do nó que sofre a inferência, os novos nós inferidos ao longo do processo de prova só podem ser adicionados no final do seu respectivo ramo, se tornando o novo nó folha.

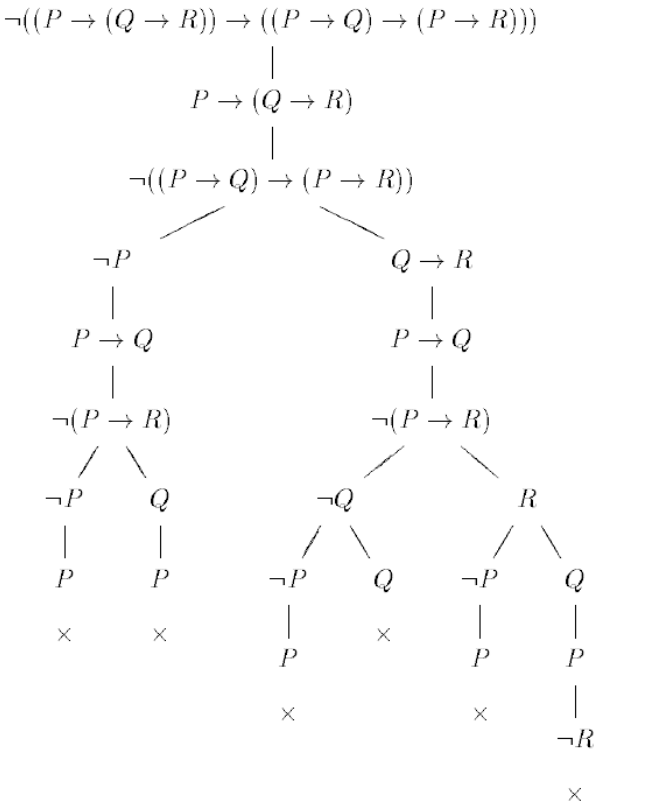

Figura 3 - Exemplo de uma árvore de prova. Fonte - D'AGOSTINO, et al., 1999, pág. 72.

Um mesmo nó pode sofrer inferência apenas uma vez em cada ramo. Considerando um processo de

execução linear (não paralelo), no qual os nós são gerados em um único ramo por vez, o algoritmo deve executar um *backtracking* (recuo) até o nó com ramificação imediatamente anterior e continuar a gerar os nós do novo ramo. As fórmulas dos nós antecedentes que ainda não foram inferidas no ramo corrente podem ser novamente inferidas. Note-se o caso do terceiro nó da figura 3 que expande os dois ramos laterais. Ou seja, o ramo é um dos principais pontos de controle na execução de um software de tableau.

Quando não for possível aplicar novas inferências em um ramo, ele se torna exaurido (esgotado) e é marcado como aberto. Caso a inclusão de um nó represente uma contradição, o ramo é marcado como fechado e não se aplica mais inferências nesse ramo. Uma contradição ocorre quando existem as fórmulas $\{\varphi, \neg\varphi\}$ no conjunto de nós do ramo. Essa verificação deve ser realizada a cada inclusão de nós (expansão do ramo).

No tableau clássico, como é o caso do tableau Smullyan, um ramo aberto indica que existe ao menos uma interpretação (modelo) que não satisfaz o sistema lógico. Ao contrário, um ramo fechado representa uma interpretação que satisfaz o sistema lógico.

O processo de prova termina se a árvore de prova se exaurir, ou seja, quando todos os ramos se tornam esgotados ou fechados. Independentemente do tipo de tableau, caso todos os ramos estejam fechados, o sistema lógico representa uma tautologia.

2.3.2 CLASSIFICAÇÃO DAS FÓRMULAS

Um importante conceito no método tableau é a classificação das fórmulas de cada nó. As fórmulas são classificadas como:

(α) alfa: fórmulas cuja inferência expande linearmente um ramo, por exemplo, as fórmulas com negação (*not*).

(β) beta: fórmulas cuja inferência expande a árvore por meio de uma nova ramificação.

(γ) gama: fórmulas de quantificadas universalmente.

(δ) delta: fórmulas de quantificadas existencialmente.

Literais ou atômicas são as fórmulas que não podem derivar outras fórmulas por nenhuma regra de inferência, por serem indivisíveis.

2.3.3 BREVE HISTÓRIA

Segundo Marcello D'Agostino (D'AGOSTINO, et al., 1999), o nome "tableau" (tabela e não árvore) foi proposto por Evert W. Beth, um dos principais idealizadores do método. Inicialmente, ele apresentou a técnica em forma de tabelas, cada uma com duas colunas, uma para as fórmulas verdadeiras e outra para as falsas. Ao longo do tempo, outros autores apresentaram a versão em forma de árvore.

2.3.4 TABLEAU DE SMULLYAN

A forma atual de representação dos tableaux foi proposta por Raymond Smullyan, em 1968 (D'AGOSTINO, et al., 1999). Além do formato de árvore, Smullyan representou os nós com sinais (V ou F). Conceitualmente, as regras de inferência são muito similares às regras gerais:

$$\frac{TA \wedge B}{\begin{array}{l}TA\\TB\end{array}} \qquad \frac{FA \wedge B}{FA \mid FB} \qquad \frac{TA \vee B}{TA \mid TB} \qquad \frac{FA \vee B}{\begin{array}{l}FA\\FB\end{array}}$$

$$\frac{TA \rightarrow B}{FA \mid TB} \qquad \frac{FA \rightarrow B}{\begin{array}{l}TA\\FB\end{array}} \qquad \frac{T\neg A}{FA} \qquad \frac{F\neg A}{TA}$$

Figura 4 - Inferência sinalizadas. Fonte - D'AGOSTINO, et al., 1999, pág. 57.

Na prática de elaboração do software para o presente trabalho, observou-se que os sinais dos nós facilitam a busca de fechamento e comparação de fórmulas na árvore. Por outro lado, as variáveis e os controles dos sinais tornam o código menos legível, aumentando a probabilidade de erros de programação.

Apesar da simplicidade estrutural, o tableau de Smullyan apresenta algumas limitações típicas. Um exemplo é a repetição, em ramos diferentes, de inferências já executadas e que não alteram o resultado final. Segue-se o exemplo da solução do sistema (A v B), (A v ¬B), (¬A v C), (¬A v ¬C) ⊨⊥:

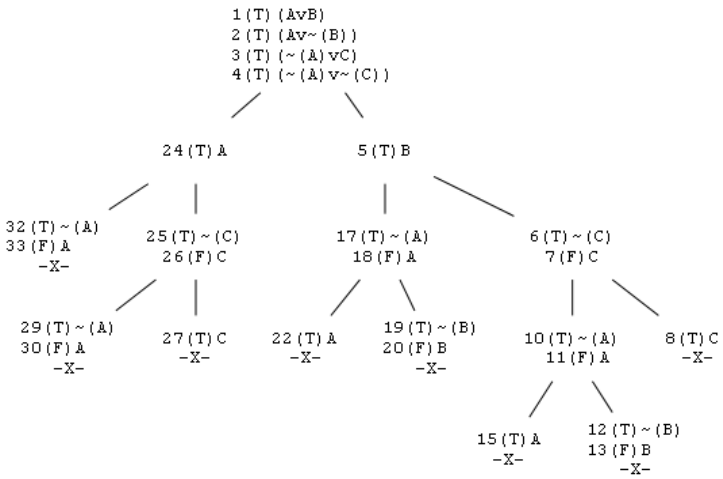

Figura 5 – Exemplo Tableau de Smullyan. Fonte - Próprio autor (gerado pelo software do trabalho).

2.3.5 TABLEAU COM LEMA

Uma técnica para tentar reduzir o tamanho da árvore de prova é o uso de lemas na construção do tableau (D'AGOSTINO, et al., 1999, pág. 91). A ideia geral do método é: quando ocorrer uma ramificação, é possível inserir um nó extra com uma fórmula oposta (negativa) à fórmula inserida no outro ramo. Esse nó poderá evitar a ocorrência de duplicação durante a expansão de nós. Note-se a figura 6 que apresenta uma abordagem operacional da técnica.

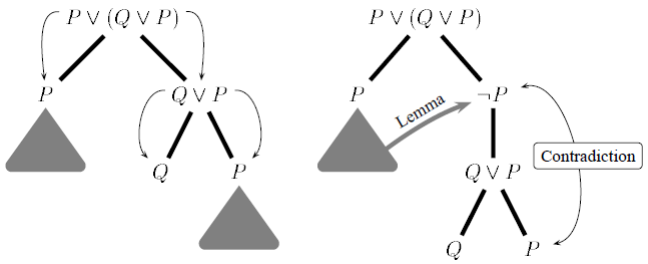

Figura 6 – Aplicação de *lemma* em tableau. Fonte - D'AGOSTINO, et al., 1999, pág. 619.

Considere-se o ramo *k* com um nó *n* que possui a fórmula θ (theta). Se o ramo *k* é fechado, então é possível afirmar que $\Gamma \vdash \neg\theta$. Nesse caso, ¬θ é um lema, ou seja, uma fórmula previamente considerada como verdadeira. Daí, ¬θ pode ser utilizado em qualquer outro ramo como uma fórmula válida. Contudo, inserção de nó com lema pode gerar efeito "indesejado" caso o ramo *k* não se feche. Segundo FUCHS (1998) e D'AGOSTINO (1999) essa inserção de fórmula não afeta o resultado final se a prova buscar verificar uma tautologia, ou teorema. Se a fórmula for uma tautologia, o ramo oposto ao lema certamente será fechado e o lema será válido. Se a fórmula não for uma tautologia e o ramo *k* for aberto, o sistema lógico não representará uma tautologia por causa do próprio ramo k aberto, independente do ramo do lema fechar ou não por causa de sua introdução.

Em muitos casos, a inserção de lema pode aumentar o tamanho da árvore de prova. Por exemplo, se não ocorrer nenhum nó com a fórmula P no ramo que o lema ¬P for adicionado, o lema é desnecessário. Por isso, o processo de geração do lema é fundamental para o sucesso da técnica. Porém, esse processo de geração não possui solução garantida (FUCHS, 1998, pág. 7).

Além disso, se a fórmula do lema não for atômica, é preciso cuidado ao se aplicar regras de inferência sobre o lema, pois, por definição, um lema é uma unidade lógica e deve ser conservada verdadeira (FUCHS, 1998, pág. 12). Ou seja, o lema e nenhuma das eventuais subfórmulas dele inferidas podem gerar ramificações.

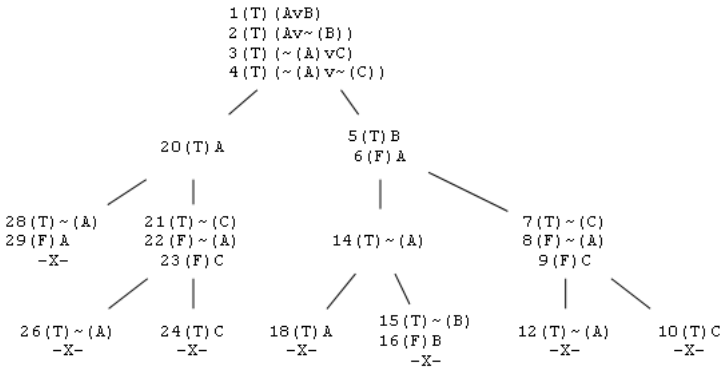

Figura 7 - Exemplo Tableau com lema. Fonte - Próprio autor (gerado pelo software do trabalho).

2.3.6 TABLEAU KE

Um conceito fundamental no estudo de provas de sistemas lógicos é o princípio do corte (*analytic cut*) de fórmulas. No geral, um corte ocorre quando uma inferência elimina fórmulas após analisar a existência de um das suas subfórmulas ao longo do caminho da prova. Exemplos de regras inferências com cortes são *modus ponens*, *modus tollens* e silogismo disjuntivo. Essas inferências analisam a existências de subfórmulas na prova para eliminar ou cortar fórmulas.

Gerhard Gentzen provou que qualquer método de prova se mantém completo e correto com ou sem as regras de inferências com cortes. Gentzen propôs um algoritmo para remover as inferências com cortes, introduzindo o conceito de eliminação de corte (*cut-elimination theorem*). Para D'AGOSTINO (1999), o trabalho de Gentzen estabeleceu o fundamento teórico para o método tableau clássico que não permite as regras de inferências com cortes.

Contudo, como demonstrado no algoritmo DPLL, os cortes possuem grande potencial para reduzir o tamanho das provas. Nessa linha de pensamento, D'AGOSTINO (1999) propõe uma técnica que inclui as regras de inferência com cortes na elaboração do tableau:

Disjunction Rules

$$\frac{TA \vee B \quad FA}{TB}\,ET\vee 1 \qquad \frac{TA \vee B \quad FB}{TA}\,ET\vee 2 \qquad \frac{FA \vee B}{\begin{array}{l}FA\\FB\end{array}}\,EF\vee$$

Conjunction Rules

$$\frac{FA \wedge B \quad TA}{FB}\,EF\wedge 1 \qquad \frac{FA \wedge B \quad TB}{FA}\,EF\wedge 2 \qquad \frac{TA \wedge B}{\begin{array}{l}TA\\TB\end{array}}\,ET\wedge$$

Implication Rules

$$\frac{TA \to B \quad TA}{TB}\,ET\to 1 \qquad \frac{TA \to B \quad FB}{FA}\,ET\to 2 \qquad \frac{FA \to B}{\begin{array}{l}TA\\FB\end{array}}\,EF\to$$

Negation Rules

$$\frac{T\neg A}{FA}\,ET\neg \qquad \frac{F\neg A}{TA}\,EF\neg$$

Principle of Bivalence

$$\frac{}{TA \mid FA}\,PB$$

Figura 8 - Regras de inferência do tableau KE. Fonte - D'AGOSTINO, et al., 1999, pág. 89.

O tableau KE modifica o fundamento básico do tableau clássico. Para que as provas sejam completas e corretas, nenhuma regra de inferência, incluindo as fórmulas tipo β, podem gerar novos ramos. A única forma de geração de ramos na árvore de prova KE é utilizar o princípio da bivalência (PB - *Principle of Bivalence*):

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma \vdash \Delta, A}{\Gamma \vdash \Delta}$$

Figura 9 – Princípio da bivalência (PB) Fonte - D'AGOSTINO, et al., 1999, pág. 89.

A regra de inferência PB só pode ser aplicada quando não for possível aplicar nenhuma das outras regras de inferência e ainda existirem nós β (disjunções) que não sofreram inferências. Um ramo KE se esgota quando não houver condições de aplicar a inferência PB. O caso de fechamento é o mesmo de qualquer

tableau, a existência, no mesmo ramo, de uma contradição.

“Allowing Cuts in an automated proof system is, in a sense, allowing the use of Lemmas” (D’AGOSTINO, et al., 1999, pág. 11). Em português, “permitir cortes em um sistema de prova automatizado é, de certo modo, permitir o uso de Lemas”.

O PB é uma técnica baseada no conceito de lema. Por isso, similar ao tableau com lemas, a escolha da fórmula do PB é o ponto fundamental no método KE. Porém, diferentemente do tableau lema, os nós originados da inferência PB podem sofrer ramificações sem alterar sua validade.

A referência principal adotada para o estudo do método KE (D’AGOSTINO, et al., 1999) destaca a importância da seleção adequada do PB. Porém, apresenta apenas uma heurística simples e intuitiva baseada em fórmulas atômicas com maior repetição. Contudo, nem sempre as fórmulas disponíveis para seleção do PB são atômicas. Em vários casos, a adoção de fórmulas não atômicas é a única alternativa para a conclusão do tableau KE.

É clara a relação direta entre o Tableau KE e o algoritmo DPLL, destacada em (D’AGOSTINO, et al., 1999). ABATE e GORÉ, no artigo sobre implementação de tableaux genéricos, sugere como método de seleção do PB as heurísticas MOMS (*Maximum number of Occurrence in dijunctions of Minimum Size*) e sua inversa a iMOMS, tipicamente utilizada no algoritmo DPLL.

O tableau KE sofre de limitações similares ao tableau com lema como, por exemplo, (1) a possibilidade de resultar em provas mais extensas, especialmente nos sistemas lógicos que não representam tautologias, e (2) dificuldade de selecionar a fórmula usada na ramificação (PB). Além disso, os algoritmos para a automação do tableau KE são bem mais complexos e demandam de muito mais controle do que o tableau clássico. Mesmo assim, o potencial de reduzir consideravelmente tanto o espaço de busca quanto o tamanho da solução do problema SAT, torna o tableau KE um relevante método de prova. Note-se que a solução do sistema (A v B), (A v ¬B), (¬A v C), (¬A v ¬C) $\models \perp$ pelo método do tableau KE é bem menor que as outras soluções apresentadas neste artigo:

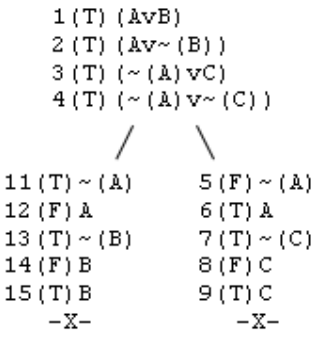

Figura 10 - Tableau KE. Fonte - Próprio autor (gerado pelo software do trabalho).

Contudo, também é importante notar que o sistema apresentado se encontra na forma conjuntiva normal (CNF) que é o caso típico de sucesso desse método.

Além de usar fórmulas em CNF, outra forma típica de aumentar a eficiência do tableau KE é usar operadores “E” (^) e “OU” (v) n-ários, na forma (X_1 ^ X_2 ^ X_3 ^... X_n) ou (X_1 v X_2 v X_3 v ... X_n). Esses operadores ampliam a capacidade de corte e facilitam a comparação entre as fórmulas, pois resolvem a questão da variação da sintaxe das fórmulas em função da comutatividade e associatividade desses operadores. Por outro lado, essas técnicas são conflitantes com o objetivo genérico de tornar as soluções das provas mais humanamente legíveis.

3 MODELAGEM DO SOFTWARE

O software elaborado para o trabalho foi codificado em Java™. Evitou-se, ao máximo, otimizações adiantadas. O design do software buscou priorizar a generalização e adaptabilidade do software, considerando em segundo plano a economia de recursos, principalmente de memória. Por isso, o

software utilizou recursos com reconhecido impacto de esforço computacional como, por exemplo, a reflexão da linguagem Java.

3.1 A LINGUAGEM E O COMPILADOR

Inicialmente, o objetivo era lidar com lógica de primeira ordem. Depois, restringiu-se à lógica proposicional. Contudo, na etapa inicial, antes da restrição do escopo, o primeiro problema resolvido foi a entrada de dados do software. Considerando os objetivos didáticos, a melhor forma de entrada de dados seria em forma de linguagem com sintaxe mais próxima da notação matemática usual. Apesar de existir uma sintaxe matemática para lógica de primeira ordem, o uso de símbolos, especialmente em alfabeto grego, dificulta a entrada de dados por teclado alfanumérico. Mesmo depois de várias pesquisas, também não se encontrou uma especificação padronizada adequada ao caso. Então, elaborou-se a seguinte gramática:

```
/* Símbolos Terminais */
lower        = [a-z]
upper        = [A-Z]
letter       = [a-zA-Z]
DIG          = [0-9]
IDENTIFIER   = {lower}({lower}|{dig}|"_")*
PREDICATE    = {upper}({upper}|{digit}|"_")*
AND          = "^" | "&" | "*" | "and"
OR           = "v" | "|" | "+" | "or"
NOT          = "~" | "!" | "-" | "not"
IMPLIES      = "->" | "=>"
REV_IMPLIES  = "<-" | "<="
EQUIVALENT   = "<->" | "<=>"
EXISTS       = "Ex" | "Exists"
FORALL       = "Fa" | "Forall"
EQUALITY     = "="
DERIVATION   = "|-"
ENTAILMENT   = "|="

/* Precedencia, da menor para a maior: */
Associação à esquerda: ENTAILMENT,DERIVATION,
EQUIVALENT,IMPLIES,REV_IMPLIES,OR,AND,NOT;
Não associativa: EQUALITY;

/* BNF  - gramática LALR(1)*/
Start  ::= LogicalSystem
System ::= FormulaList DERIVATION FormulaList
         | DERIVATION FormulaList
         | FormulaList DERIVATION
         | Formula
FormulaList   ::= FormulaList "," Formula
                |  Formula
Formula       ::=  AtomicFormula
                | NOT Formula
                | Connective
                | Quantifier
                | "("Formula")"
AtomicFormula ::= Predicate
                | Term EQUALITY Term
Predicate     ::= PREDICATE
                | PREDICATE"("TermList")"
Function      ::= IDENTIFIER"("TermList ")"
                | IDENTIFIER "()"
Variable      ::= IDENTIFIER
Term          ::= Function
                | Variable
TermList      ::= TermList "," Term
                | Term
Connective    ::= Formula AND Formula
                | Formula OR  Formula
                | Formula EQUIVALENT  Formula
                | Formula IMPLIES     Formula
                | Formula REV_IMPLIES Formula
Quantifier    ::= EXISTS":"Variable Formula
                | FORALL":"Variable Formula
```

Em termos sintáticos, toda proposição deve ser escrita em letras maiúsculas. Toda função deve ter parênteses, mesmo que não possua argumento, para distinguir das variáveis, pois ambas devem ser escritas em letras minúsculas. Como não existe o conceito de termos (variáveis ou funções) na lógica proposicional, as proposições dessa lógica não podem possuir argumentos ou parêntesis. Caso contrário, a fórmula com proposições com parêntesis será tratada como lógica de predicado.

O compilador utiliza o JFlex para a geração do analisador léxico e o CUP para a geração do analisador sintático.

3.2 O SISTEMA LÓGICO

Para a programação do sistema lógico, adotou-se uma hierarquia de classe similar à semântica da lógica de primeira ordem. Essas classes são geradas durante a compilação da linguagem adotada. Ou seja, elas atuam como árvore sintática abstrata (AST - Abstract Syntax Tree) para o compilador. O compilador retorna sempre um objeto da classe “LogicalSystem”. No entanto, apesar de atuar como AST, alguns métodos operacionais que são típicos da interpretação foram implementados nessas classes, como, por exemplo, a avaliação recursiva de modelos (interpretação) de

estruturas da lógica proposicional, o cálculo do tamanho da fórmula (|φ|).

Essas classes são utilizadas por todos os demais módulos do software. A figura 11 mostra um diagrama das classes do sistema lógico.

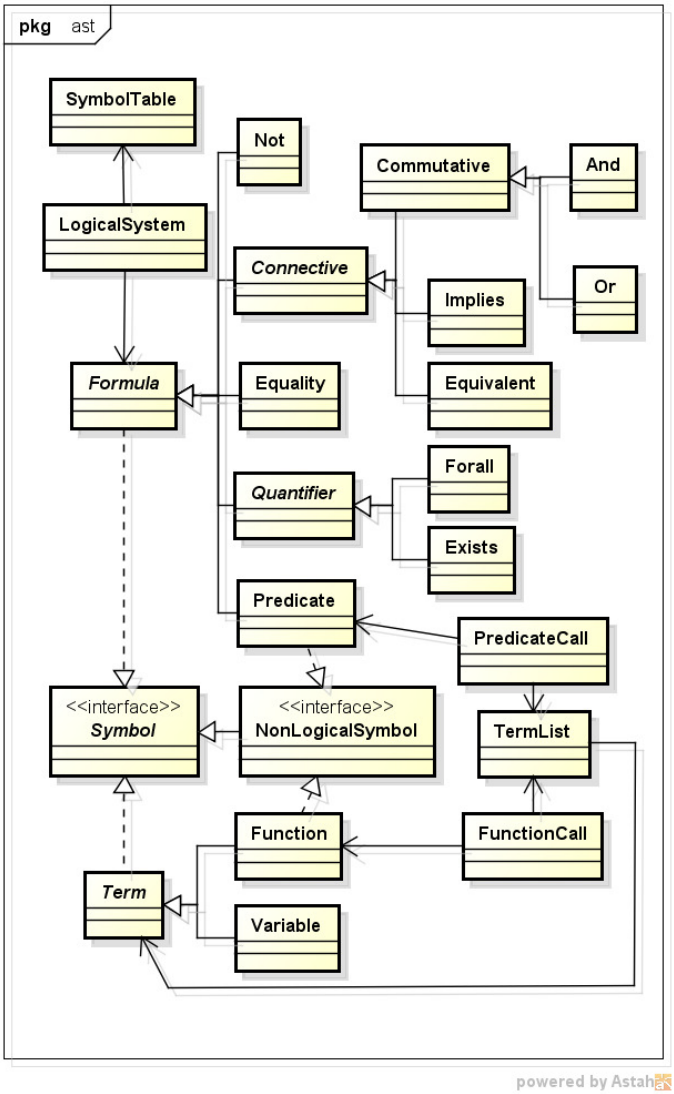

Figura 11 - Diagrama da AST da linguagem do sistema de lógica de premeria ordem. Fonte - Próprio autor.

Assim como nos demais módulos do software, existem vários detalhes de design e codificação que não fazem parte do escopo deste artigo. Contudo, um detalhe específico foi a adoção de uma classe abstrata "Commutative" para os conectivos "E" (*and*) e "OU" (*or*). Essa classe tenta ampliar a possibilidade de comparação entre subfórmulas, buscando igualdades comutativas da forma (A = B) ou (B = A). Porém, esse recurso não resolve a questão da associatividade. Um plano futuro é adotar conectivos "E" e "OU" n-ários, como apresentado nos fundamentos do tableau KE. Contudo, esse tipo de conectivo também diminui a legibilidade e o sentido didático do software de prova, por isso, o ideal é que seja um recurso opcional.

3.3 OS MÓDULOS DE PROVA

O módulo de prova contém as classes básicas e a interface geral para os motores (*engine*) de prova.

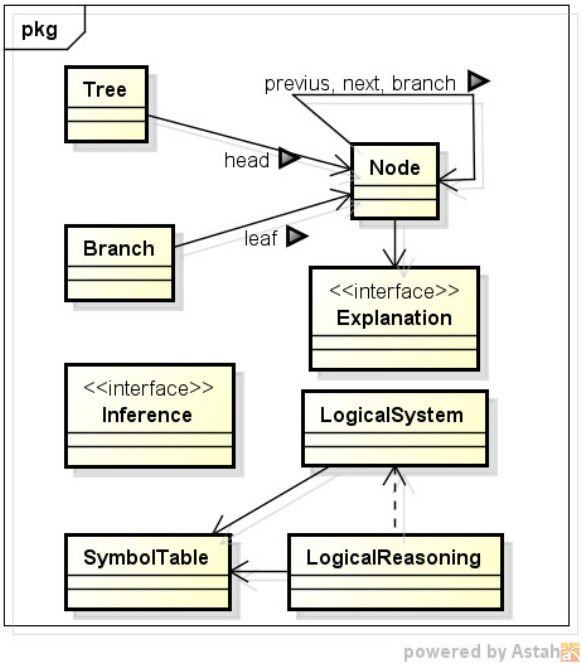

Figura 12 - Diagrama das classes de prova. Fonte - Próprio autor.

A árvore de prova é uma lista duplamente encadeada, sendo que um nó pode possuir um ramo (*branch*) e uma explicação (*Explanation*) opcionais. A explicação registra a origem do nó para informar com mais detalhes as regras de inferência aplicadas no processo de prova. A árvore não referencia os ramos, pois os ramos não são elementos estruturais, são elementos de controle usado pelos motores de inferência. A classe "LogicalReasoning" (raciocínio lógico) é a classe base para todos os métodos de prova implementados no software.

Uma classe utilitária relevante é a “DotTreeGenerator” que gera arquivos na linguagem DOT para elaborar imagens das árvores de prova no programa Graphiz. As diversas imagens dispostas neste artigo com exemplos de soluções geradas pelo software elaborado para o trabalho foram obtidas dessa forma.

3.3.1 TABELA-VERDADE

O módulo de tabela verdade é composto de uma classe principal e outras duas assessórias utilizadas para auxiliar na formatação da “impressão” da tabela.

Uma decisão de design relevante: a avaliação do sistema lógico não ocorre na tabela verdade, ela ocorre na estrutura das classes do sistema lógico, por meio de recursão. A vantagem dessa alternativa é o desempenho. Um plano futuro é desacoplar essa dependência e separar a interpretação da AST.

Uma curiosidade é que, como não existe uma estrutura que mantém os resultados da tabela verdade, o processo de “impressão” da tabela envolve a avaliação de cada interpretação possível da fórmula. Essa também é uma característica candidata a ser reavaliada em trabalhos futuros.

3.3.2 TABLEAU SEMÂNTICO

O módulo de tableau semântico fornece a base para a implementação das variantes do método. As principais decisões de design deste módulo são:

1. Controle dos ramos (*branchs*) como, por exemplo, coordenar o processo de inferência.
2. Generalização do processo de escolha e aplicação das regras de inferência (que caracterizam cada método tableau específico).
3. Seleção de nós que serão usados na inferência, incluindo o controle dos nós que foram ou ainda devem ser inferidos.
4. Classificação dos nós de acordo com as fórmulas ou o contexto da execução.

Um ponto fundamental de todo método tableau é o controle de ramos. Todos os textos que tratam sobre o método tableau destacam a necessidade da execução do “*backtracking*”, ou seja, volta ao ponto de ramificação imediatamente anterior. Contudo, observa-se que o processamento dos ramos é praticamente independente um do outro. Ou seja, é uma ótima oportunidade para adoção de algoritmos paralelos (*multithread*). Para isso, cada ramo deve possuir seus controles individuais e separados. A adoção de paralelismo é viável, pois praticamente não ocorre condição de corrida (concorrência para alteração de recursos) durante o processamento dos ramos. Além disso, mesmo não se adotando paralelismo, o controle separado facilita a codificação e os controles do programa de prova.

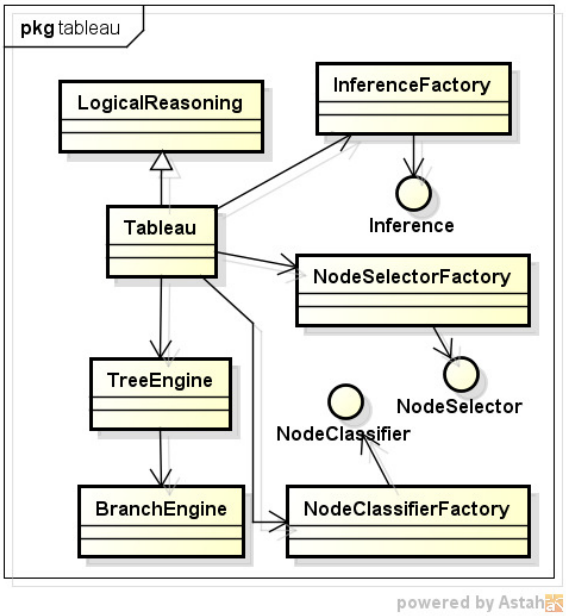

Figura 13 - Diagrama das classes de prova. Fonte - Próprio autor.

O controle de ramos é realizado pela classe “BranchEngine”. A tarefa da classe “TreeEngine” é controlar a execução dos diversos ramos.

É importante destacar que a versão do software elaborada para o presente trabalho não utiliza

programação paralela nos algoritmos, e essa também é uma oportunidade para trabalhos futuros.

Como os controles são separados, para possibilitar o processamento paralelo, cada "BranchEngine" constrói seus próprios objetos de regras de inferência, seleção e classificação de nós.

O objeto de classificação tem a responsabilidade de definir o tipo do nó (φ,β,γ,δ ou atômico) e sua prioridade no processo de seleção. Os tipos de nós γ e δ não são usados, pois são específicos para lógica de predicado. A classificação padrão atribui prioridade mais alta para os nós φ e mais baixa para os β.

Na classe padrão, a fórmula φ de maior prioridade é a da classe "Not". Como resultado, sempre que um nó com fórmula "Not" é adicionado, se não ocorrer o fechamento do ramo, o próximo adicionado normalmente possui a fórmula sem o "Not" e com sinal invertido. Esse tipo de aplicação de prioridade baseada apenas na classe da fórmula do nó acaba aumento o tamanho da maioria das provas.

A seleção de nós pode ou não usar as informações de classificação. Contudo, por motivos óbvios, nós atômicos nunca são selecionados para inferência. Diferentes classificações de prioridade causam resultados bem diversos no resultado final da árvore de prova. Ou seja, o estudo de técnicas de seleção de nós é um interessante ponto para eventuais de pesquisas futuras.

A implementação básica da classe de seleção de nós utiliza uma fila com prioridade (*PriorityQueue<Node>* no Java).

3.3.2.1 TABLEAU SIMPLES

Esse método de prova é baseado no tableau Smullyan, com acréscimo de uma regra de inferência para o conectivo de equivalência (↔). Para o método de prova simples, bastou implementar sua classe de inferência e sua fábrica (*factory*). Os objetos de seleção de nós e classificação usam a classe padrão.

4.3.2.2 TABLEAU COM LEMMA

Esse módulo também possui apenas sua classe de inferência e sua fábrica (*factory*). Os objetos de seleção de nós e classificação usam a classe padrão.

A geração de lemas adotou o método mais simples possível: para cada ramificação, adiciona-se um nó com o lema. Para escolha da fórmula lema, se a da direita ou da esquerda da ramificação, vários métodos foram testados como tamanho do texto da fórmula, número de elementos da fórmula e número de ocorrências da formula no ramo. Porém, no geral, os resultados não foram satisfatórios. Como já discutido, esse é um fator relevante, mas complexo e está relacionado com a seleção da ramificação PB do tableau KE.

3.3.2.3 TABLEAU KE

O módulo de tableau KE possui suas próprias classes de inferência, seleção e classificação de nós. Além disso, existe uma classe específica para a seleção dos nós PB quando não é possível mais aplicar as regras de inferência, mas ainda é possível ramificar a árvore. Essa seleção usa a heurística de selecionar a fórmula de menor tamanho que ocorra maior número de vezes.

3.4 MÓDULO PRINCIPAL

O programa recebe os seguintes parâmetros pela linha de comando:

```
ProofApp [opções] [arq. com a formula]
Opções da linha de comando:
  -s = Debug da análise sintática
  -l = Debug da análise léxica
  -t = Mostra a tabela de símbolos
  -n = Não realiza apenas a análise léxica
  -o = Arquivo de saída do grafo da solução
  -h = Help
```

```
java ProofApp -o teste.gv -t teste.txt
```

O comando acima informa que o arquivo com o sistema lógico a ser copilado é "teste.txt" e que os arquivos das árvores de prova, na linguagem "DOT",

devem terminar com “teste.gv”, ou seja, “simple_teste.gv”, “lemma_teste.gv” e “ke_teste.gv”.

4 RESULTADOS OBTIDOS

O objetivo de geração de prova legível e de menor tamanho possível levou ao estudo e adoção de três métodos de tableau. Além disso, ao menos no momento, esse objetivo é mais um ideal, que foi perseguido com muito esforço, do que um fator científico no qual a avaliação é fundamental. Além disso, legibilidade é um fator com características subjetivas que demandaria de outra pesquisa com amostragem de alunos e experimentos controlados para sua avaliação científica adequada. Esse também poderia ser considerado um tema para perspectivas futuras.

A questão relevante é: o software é capaz de realizar provas legíveis de sistemas em lógica proposicional? A resposta é sim. Muitas fórmulas foram testadas durante a elaboração do software e do artigo. O software funciona corretamente, ou seja, apresenta resultados esperados? A resposta é, até onde se sabe, sim. Todo software é propenso a erros, principalmente softwares em versões recentes. O módulo com maior propensão a erros é o do tableau KE, devido a sua complexidade conceitual e a falta de exemplos mais complexos de referência para os testes e comparação de resultados.

Até a data da elaboração deste artigo, o software contém cerca de 1680 linhas de código (cerca de 4300 no total), além de cerca de 790 linhas de código geradas pelo JFlex e CUP.

Foram observados os seguintes resultados empíricos:

1) O método Smullyan funcionou em todos os testes.

2) Como já foi mencionado na fundamentação, no geral, devido à escolha simples da fórmula lema, o tableau lema quase sempre gerou árvore de prova maior que o tableau simples. A principal causa de aumento foram os lemas irrelevantes. Além disso, mesmo que o sistema represente uma contradição, o tableau quase sempre apresenta ramos fechados. Esse resultado já seria esperado devido à limitação do método de tableau com lema (ver fundamentação).

3) Para os sistemas lógicos que representam tautologias, o tableau KE sempre apresentou prova de menor tamanho. No caso contrário, se o sistema lógico representa uma contradição, os números de nós do tableau KE sempre foram maiores que os demais métodos testados. Baseando-se na fundamentação teórica, esses resultados também eram previsíveis.

A observação desses resultados levanta a questão de controle de execução das provas. Caso se deseje provar uma tautologia, é possível incluir controles que verificam antecipadamente modelos (interpretações) falsas e encerre o processamento quando esses casos ocorrerem. Por exemplo, assim que um ramo se exaure e termina aberto, sabe-se antecipadamente que existe ao menos uma interpretação que prova que a fórmula não é uma tautologia. Esse tipo de controle é outro tema relevante para a eventual continuidade futura da linha de pesquisa do trabalho.

O software elaborado para o trabalho está disponível publicamente no GitHub (https://github.com/fredmbs/logic).

Segue-se um exemplo de execução de teste para uma tautologia. Observe que o tableau KE representa a menor árvore de prova:

```
Sistema:((Pv(Q^R))->((PvQ)^(PvR)))
==> Lógica Proposicional
==> Sistema lógico de prova simples (uma fórmula)
/-----------------------------------------------\
|                  Tabela de símbolos           |
|--------+------------+--------+-----------------|
| Escopo | Tipo       | Ocorr. | Lexema          |
|--------+------------+--------+-----------------|
|      0 | Predicate  |      2 | Q               |
|      0 | Predicate  |      3 | P               |
|      0 | Predicate  |      2 | R               |
\--------+------------+--------+-----------------/
Cópia (clone) do sistema lógico:
Sistema:((Pv(Q^R))->((PvQ)^(PvR)))
A fórmula representa uma tautologia.
/---------------------------------------\
|P|Q|R|((P v (Q ^ R)) -> ((P v Q) ^ (P v R)))|
|-+-+-+---------------------------------------|
|T|T|T|    T    T     T*     T    T    T     |
|T|T|F|    T    F     T*     T    T    T     |
|T|F|T|    T    F     T*     T    T    T     |
```

```
|T|F|F|    T    F     T*     T    T    T     |
|F|T|T|    T    T     T*     T    T    T     |
|F|T|F|    F    F     T*     T    F    F     |
|F|F|T|    F    F     T*     F    F    T     |
|F|F|F|    F    F     T*     F    F    F     |
\-------------------------------------------/
A fórmula representa uma tautologia.
Solução por Tabela Verdade (com shortCut)' em 0 ms
-------------------------------------
Segunda cópia do sistema lógico:
((Pv(Q^R))->((PvQ)^(PvR)))
A fórmula representa uma tautologia.
Solução por Tabela Verdade (sem shortCut)' em 0 ms
-------------------------------------
Tableau:
Solução por Tableau Semântico ' em 2 ms
Inference rules = Smullyan's Tableau
Node selector   = Priority Node Selector
     1 | (F) ((Pv(Q^R))->((PvQ)^(PvR)))  {H}
     2 | (T) (Pv(Q^R))          {1 F->:l}
     3 | (F) ((PvQ)^(PvR))      {1 F->:r}
     4 | | (F) (PvR)            {3 F^:r}
     5 | | (F) P                {4 Fv:l}
     6 | | (F) R                {4 Fv:r}
     7 | | | (T) (Q^R)          {2 Tv:r}
     8 | | | (T) Q              {7 T^:l}
     9 | | | (T) R              {7 T^:r}
    10 | | | -X-                {9,6 closure}
    11 | | (T) P                {2 Tv:l}
    12 | | -X-                  {11,5 closure}
    13 | (F) (PvQ)              {3 F^:l}
    14 | (F) P                  {13 Fv:l}
    15 | (F) Q                  {13 Fv:r}
    16 | | (T) (Q^R)            {2 Tv:r}
    17 | | (T) Q                {16 T^:l}
    18 | | -X-                  {17,15 closure}
    19 | (T) P                  {2 Tv:l}
    20 | -X-                    {19,14 closure}
Total de vértices = 17
TAUTOLOGY!
Salvando arquivo simple_teste.gv
-------------------------------------
Tableau:
Solução por Tableau Semântico ' em 0 ms
Inference rules = Lemma Tableau
Node selector   = Priority Node Selector
     1 | (F) ((Pv(Q^R))->((PvQ)^(PvR)))  {H}
     2 | (T) (Pv(Q^R))            {1 F->:l}
     3 | (F) ((PvQ)^(PvR))        {1 F->:r}
     4 | | (F) (PvR)              {3 F^:r}
     5 | | (F) P                  {4 Fv:l}
     6 | | (F) R                  {4 Fv:r}
     7 | | | (T) (Q^R)            {2 Tv:r}
     8 | | | (T) Q                {7 T^:l}
     9 | | | (T) R                {7 T^:r}
    10 | | | -X-                  {9,6 closure}
    11 | | (T) P                  {2 Tv:l}
    12 | | -X-                    {11,5 closure}
    13 | (F) (PvQ)                {3 F^:l}
    14 | (T) (PvR)                {lema}
    15 | (F) P                    {13 Fv:l}
    16 | (F) Q                    {13 Fv:r}
    17 | | (T) (Q^R)              {2 Tv:r}
    18 | | (T) Q                  {17 T^:l}
    19 | | -X-                    {18,16 closure}
    20 | (T) P                    {2 Tv:l}
    21 | -X-                      {20,15 closure}
Total de vertices = 18
TAUTOLOGY!
Salvando arquivo lemma_teste.gv
-------------------------------------
Tableau:
Solução por Tableau Semântico ' em 1 ms
Inference rules = KE Tableau
Node selector   = KE Tableau
     1 | (F) ((Pv(Q^R))->((PvQ)^(PvR)))  {H}
     2 | (T) (Pv(Q^R))            {1 F->:l}
     3 | (F) ((PvQ)^(PvR))        {1 F->:r}
     4 | | (F) (PvQ)              {PB-F}
     5 | | (F) P                  {4 Fv:l}
     6 | | (F) Q                  {4 Fv:r}
     7 | | (T) (Q^R)              {2,5 Tv-cut:l}
     8 | | (T) Q                  {7 T^:l}
     9 | | -X-                    {8,6 closure}
    10 | (T) (PvQ)                {PB-T}
    11 | (F) (PvR)                {3,10 F^-cut:l}
    12 | (F) P                    {11 Fv:l}
    13 | (F) R                    {11 Fv:r}
    14 | (T) Q                    {10,12 Tv-cut:l}
    15 | (T) (Q^R)                {2,12 Tv-cut:l}
    16 | (T) R                    {15 T^:r}
    17 | -X-                      {16,13 closure}
Total de vertices = 16
TAUTOLOGY!
Salvando arquivo ke_teste.gv
```

Segue-se um exemplo de execução de teste para uma contradição. Observe que o tableau KE representa a maior árvore de prova:

```
Sistema:((A^B)^(B->~(A)))
==> Lógica Proposicional
==> Sistema lógico de prova simples (uma fórmula)
/---------------------------------------------\
|                 Tabela de símbolos          |
|--------+------------+--------+--------------|
| Escopo | Tipo       | Ocorr. | Lexema       |
|--------+------------+--------+--------------|
|      0 | Predicate  |      2 | A            |
|      0 | Predicate  |      2 | B            |
\--------+------------+--------+--------------/
Cópia (clone) do sistema lógico:
Sistema:((A^B)^(B->~(A)))
/--------------------------\
|A|B|((A ^ B) ^ (B -> ~(A)))|
|-+-+------------------------|
|T|T|    T   F*    F  F     |
|T|F|    F   F*    T  F     |
|F|T|    F   F*    T  T     |
|F|F|    F   F*    T  T     |
\--------------------------/
Solução por Tabela Verdade (com shortCut)' em 0 ms
Segunda cópia do sistema lógico:
((A^B)^(B->~(A)))
Solução por Tabela Verdade (sem shortCut)' em 0 ms
-------------------------------------
Tableau:
Solução por Tableau Semântico ' em 1 ms
Inference rules = Smullyan's Tableau
Node selector   = Priority Node Selector
     1 | (F) ((A^B)^(B->~(A)))   {H}
     2 | | (F) (B->~(A))         {1 F^:r}
     3 | | (T) B                {2 F->:l}
     4 | | (F) ~(A)              {2 F->:r}
     5 | | (T) A                {4 F~}
     6 | | -O-                  {exhausted}
     7 | (F) (A^B)              {1 F^:l}
     8 | | (F) B                {7 F^:r}
     9 | | -O-                  {exhausted}
    10 | (F) A                  {7 F^:l}
    11 | -O-                    {exhausted}
Total de vertices = 9
Salvando arquivo simple_teste.gv
-------------------------------------
Tableau:
Solução por Tableau Semântico ' em 0 ms
Inference rules = Lemma Tableau
Node selector   = Priority Node Selector
     1 | (F) ((A^B)^(B->~(A)))   {H}
     2 | | (F) (B->~(A))         {1 F^:r}
     3 | | (T) B                {2 F->:l}
     4 | | (F) ~(A)              {2 F->:r}
     5 | | (T) A                {4 F~}
     6 | | -O-                  {exhausted}
     7 | (F) (A^B)              {1 F^:l}
     8 | (T) (B->~(A))          {lema}
     9 | | (F) B                {7 F^:r}
    10 | | -O-                  {exhausted}
    11 | (F) A                  {7 F^:l}
    12 | (T) B                  {lema}
    13 | -O-                    {exhausted}
Total de vertices = 11
Salvando arquivo lemma_teste.gv
-------------------------------------
Tableau:
Solução por Tableau Semântico ' em 0 ms
Inference rules = KE Tableau
Node selector   = KE Tableau
```

```
     1 | (F) ((A^B)^(B->~(A)))      {H}
     2 | | (F) (B->~(A))            {1 PB-F}
     3 | | (T) B                   {2 F->:l}
     4 | | (F) ~(A)                 {2 F->:r}
     5 | | (T) A                   {4 F~}
     6 | | -O-                     {exhausted}
     7 | (T) (B->~(A))             {1 PB-T}
     8 | | (F) ~(A)                 {7 PB-F}
     9 | | (T) A                   {8 F~}
    10 | | -O-                     {exhausted}
    11 | (T) ~(A)                  {7 PB-T}
    12 | (F) A                     {11 T~}
    13 | -O-                       {exhausted}
Total de vertices = 11
Salvando arquivo ke_teste.gv
```

Segue-se exemplo de compilação de um sistema aleatório de lógica de primeira ordem:

```
Sistema:

Ex:x(Ex:y(Q(x,h(y)))) |-
Fa:x(Ex:y((P(x,y)->Q(g(y,h(x))))))

==> Lógica de Predicado
==> Sistema lógico de derivação
/-----------------------------------------------\
|                 Tabela de símbolos            |
|--------+------------+--------+----------------|
| Escopo | Tipo       | Ocorr. | Lexema         |
|--------+------------+--------+----------------|
|      0 | Function   |      1 | g              |
|      0 | Predicate  |      2 | Q              |
|      0 | Predicate  |      1 | P              |
|      0 | Function   |      2 | h              |
|      0 | Variable   |      5 | y              |
|      0 | Variable   |      5 | x              |
\--------+------------+--------+----------------/
```

5 CONCLUSÃO E PERSPECTIVAS FUTURAS

O principal objetivo do trabalho foi elaborar o software e todo esforço foi priorizado nesse sentido. O trabalho de criação do software e de escrita do artigo trouxe muito desgaste, mas muita aprendizagem. Por isso, este objetivo, do ponto de vista de graduação, tem grande importância. Considerando que o POC (Projeto Orientado de Graduação) possui um tempo razoavelmente longo, é necessário destacar o fato que o projeto mudou radicalmente e o tempo total do trabalho foi de cerca de três meses, incluindo pesquisa, planejamento, programação, testes, correções, elaboração do artigo, etc.

No momento da escrita deste capítulo, a principal conclusão é que mudança de tema de pesquisa é uma atividade muito arriscada. O que no início se imaginava um objetivo simples, demonstrou envolver diversas áreas de conhecimento com várias técnicas específicas, que remontam uma longa história de pesquisa e evolução, que fundaram as bases da ciência da computação atual. A maior dificuldade encontrada foi acostumar, em pouco tempo, com a linguagem e termos utilizados por matemáticos e filósofos para descrever as técnicas e os fundamentos lógicos envolvidos. Do ponto de vista pessoal, essa aproximação mais íntima com o campo da lógica foi um dos principais benefícios de todo esforço aplicado na pesquisa e elaboração do presente trabalho.

Considerando o objetivo inicial de elaborar um software de prova em lógica proposicional, a conclusão parece ser positiva. A elaboração do software enfrentou mais problemas com os conceitos lógicos do que com o design da aplicação. Como todo software, principalmente em estágio inicial, o software tem muito a evoluir. O design adotado é flexível e modular, capaz de se adaptar a diferentes métodos de prova. A aplicação tem grande potencial de evoluir para uma ferramenta acadêmica útil em cursos de graduação, em especial na matéria de matemática discreta.

Os resultados e experimentos, mesmo sendo secundários em face dos objetivos iniciais, apresentaram questões relevantes, incluindo problemas difíceis de solucionar e que são temas de diversas linhas de pesquisa em computação, filosofia e matemática.

A existência de muitas perspectivas futuras é um reflexo direto da amplitude da área de pesquisa e do grande interesse acadêmico e cientifico do tema do trabalho. As perspectivas futuras incluem questões de natureza funcional, como a melhoria do software, e outras de natureza científica, como a eventual continuidade nessa linha de pesquisa. Só para citar alguns poucos exemplos de perspectivas futuras, fora as que já foram citados ao longo do trabalho:

1) Elaborar uma interface gráfica intuitiva e com resultado imediato, incluindo a possibilidade de gerar a árvore sintática de cada fórmula.
2) Criar um módulo de teste do software.
3) Ampliar a pesquisa para adaptar os algoritmos de solução para lógica de primeira ordem.
4) Pesquisar formas de geração de lema para ampliar a eficiência do tableau com lema e do tableau KE.
5) Ampliar a linha de pesquisa para aplicação do software para outras formas de lógica.
6) Melhorar o tratamento de erro do compilador, principalmente para apresentar informações mais detalhadas de eventuais erros sintáticos e semânticos.

REFERÊNCIAS

ABATE, Pietro; GORÉ, Rajeev. Theory and Pratice of a Generic Tableau Engine: The Tableau Workbench. The Australian National University and NICTA, Canberra, Australia. Disponível em: http://twb.rsise.anu.edu.au/files/tableaux07.pdf. Acessado em: 07/12/2012.

AHO, Alfred V.; LAM, Monica S.; SETHI, Ravi; ULLMAN, Jeffrey D.; *Compiladores: princípios, técnicas e ferramentas. 2ª edição*. Tradução: Daniel Vieira. São Paulo: Pearson Addison-Wesley, 2008. ISBN 978-85-88639-24-9.

D'AGOSTINO, et al., Marcello; et al. *Handbook of Tableau Methods.* Kluwer Academic Publishers. ISBN-13: 978-0792356271.

FUCHS, Mark. *Controlled Use of Clausal Lemmas in Connection Tableau Calculi.* Fakultät für Informatick, TU München, Germany, março, 1998. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.4650. Acessado em: 07/12/2012.

WIKI-C: *Completude (lógica)*. Disponível em: "http://pt.wikipedia.org/wiki/Completude_(l%C3%B3gica)". Acessado em: 07/12/2012.