ISSN: 1984-3151

PROVA AUTOMATIZADA DE TEOREMA EM LÓGICA

AUTOMATED THEOREM PROVING IN PROPOSITIONAL LOGIC

Frederico Martins Biber Sampaio[1]; Moisés Henrique Ramos Pereira[2] (Orientador); Miriam Lourenço Maia[3] (Coorientadora)

1 Bacharel em Ciência da Computação, UniBH, 2012. Aluno de Mestrado do Programa de Pós-Graduação em Ciência da Computação - UFMG. Belo Horizonte, MG. fredmbs@gmail.com.

2 Mestre em Modelagem Matemática e Computacional, CEFET-MG, 2012. Professor do Centro Universitário de Belo Horizonte - UniBH. Belo Horizonte, MG. moiseshrp+unibh@gmail.com.

3 Mestre em Ciência da Computação, UFMG, 1976. Professora do Centro Universitário de Belo Horizonte - UniBH. Belo Horizonte, MG. miriam.maia@prof.unibh.br.

Recebido em: XX/XX/XXXX - Aprovado em: XX/XX/XXXX - Disponibilizado em: XX/XX/XXXX

RESUMO: Este trabalho aborda o desenvolvimento de um sistema para prova automatizada de teoremas em lógica proposicional. O artigo apresenta os fundamentos teóricos gerais, questões operacionais e a estrutura de um software de prova de teoremas, elaborado com propósitos acadêmicos e didáticos, utilizando métodos de prova baseados em três tipos de tableaux semânticos: tableau de Smullyan, tableau com Lema e tableau KE. Experimentos foram realizados para verificar a correção dos resultados das provas, utilizando fórmulas geradas automaticamente.
PALAVRAS-CHAVE: Lógica proposicional, tableau semântico, prova de teorema automatizada.

ABSTRACT: This work describes the development of an automated theorem proving system of propositional logic. The paper presents the theoretical foundations, operational issues and structure of a theorem proving software, developed with academic and didactic purposes, using proof methods based on three semantics tableaux: Smullyan tableau, Lema tableau e KE tableau. Experiments were performed to verify the correctness of the results of the proofs, using automatically generating formulas.
KEYWORDS: Propositional logic, semantic tableau, automated theorem proof.

---

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 tema fundamental em praticamente todas as ciências, incluindo a ciência da computação. Todos os computadores digitais modernos são baseados em lógica, em especial a lógica booleana. Em termos matemáticos, a lógica booleana é uma variação da álgebra, reduzindo o universo dos números ao

conjunto {0, 1} e as operação de conjunção (“e”), disjunção (“ou”) e negação (“não”). 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. Operações lógicas relacionam as proposições, também resultando em verdade ou falsidade. As operações básicas são “e” (^), “ou” (v) e “não” (¬), mas existem outras operações como a “implicação” (→) e a “equivalência" (↔). A operação de negação é especial, pois é a única que não é um conectivo binário.

Uma fórmula proposicional pode ser composta desde apenas uma única proposição até um conjunto infinito de operações lógicas sobre diversas 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.

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. Isoladamente, uma fórmula é satisfazível se possui uma única interpretação verdadeira.

O problema da satisfação (“satisfatibilidade” ou “satisfatoriedade”) 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.

O tema de complexidade algorítmica foge ao escopo de presente trabalho, mas cabe uma introdução geral. 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, complexidades da ordem $O(n!)$, $O(k^n)$ ou $O(n^n)$ possuem crescimento maior que uma complexidade polinomial que, normalmente, é da ordem $O(n^k)$.

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 computacional de problemas em tempo hábil, 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 é especialmente interessante na computação. Mesmo que exista um algoritmo para sua solução, a execução prática pode

exigir 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. As pesquisas em inteligência artificial buscam reduzir o "espaço de busca", tornando a execução prática mais viável. Existem pesquisas que buscam novas formas de solução, novos algoritmos, que possam 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 relacionados com relevantes aplicações práticas, o problema SAT desperta enorme interesse em pesquisa. Especificamente, o problema SAT envolve, 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 fórmulas para atuar como ferramenta de ensino e pesquisa na área de lógica.

Para atingir os objetivos, o software é composto de três módulos principais: (1) compilador para uma linguagem que expresse as fórmulas e o sistema lógico a ser tratado; (2) módulos de prova; (3) módulo de teste. 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.

A segunda seção deste artigo apresenta os fundamentos teóricos que subsidiaram a elaboração do software, incluindo sistemas lógicos, métodos de prova, os fundamentos dos tableaux semânticos, o tableau de Smullyan, o tableau com lema e o tableau KE. A terceira seção apresenta o design do software elaborado. A quarta seção apresenta rapidamente os resultados obtidos pelo software, em especial a verificação prática de correção das provas. Na quinta seção são apresentadas algumas conclusões e perspectivas futuras do trabalho.

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.

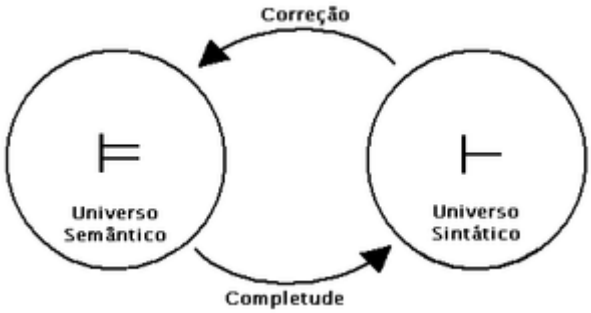

Figura 1 - Relação no cálculo lógico (WIKIPEDIA)

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 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 seus 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". Existem várias regras de inferências lógicas.

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 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 fórmulas 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, p.45)

Para a lógica proposicional, além de poder ser considerada como subconjunto da lógica de primeira ordem, ou lógica de ordem zero, existem diversas provas de sua completude e correção. Ela também 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. Contudo, como a lógica de primeira ordem pode adotar conjuntos infinitos, as demonstrações por interpretação podem ser limitadas.

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 importante 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 ou com cortes, 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 sintático de prova, tenta-se aplicar inferências sobre 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, p.57.

Cada aplicação das regras de inferência expande (aumentam) a árvore de prova adicionando novos nós. As inferências de fórmulas negativas ("não") e conjuntivas ("e") aumentam linearmente um ramo da árvore. As inferências de fórmulas disjuntivas ("ou") criam ramos (*branchs*) na árvore. Cada ramo possui um nó folha (*leaf*). Independentemente da posição do nó que sofre a inferência, os novos nós inferidos só podem ser adicionados no final do seu respectivo ramo, se tornando o novo nó folha.

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 desse 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.

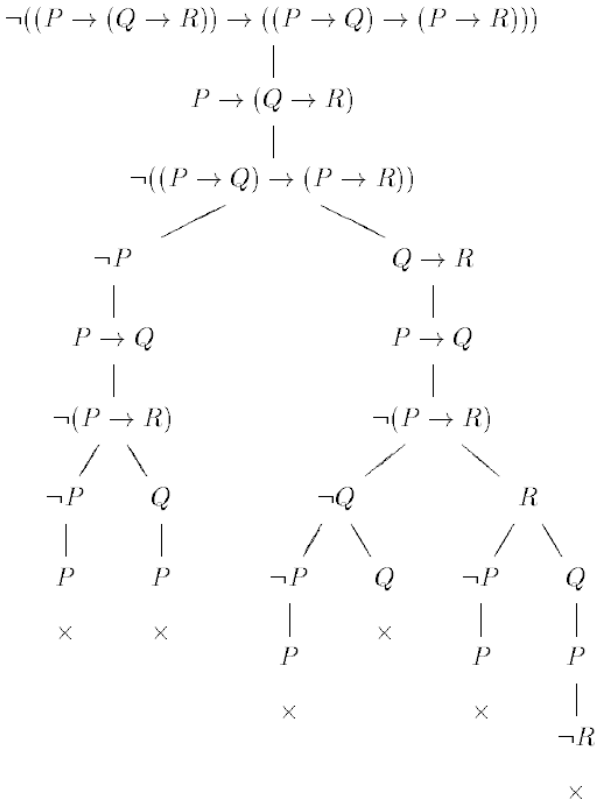

Figura 3 - Exemplo de uma árvore de prova. Fonte - D'AGOSTINO, et al., 1999, p.72.

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. 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.

No tableau clássico, um ramo aberto indica que existe ao menos uma interpretação (modelo) que não satisfaz o sistema lógico.

O processo de prova termina se a árvore de prova se exaurir, ou seja, quando todos os ramos se tornam abertos ou fechados. Independentemente do tipo de tableau, caso todos os ramos estejam fechados, o sistema lógico representa uma tautologia, pois não existe um modelo que o refute (contramodelo). Contudo, se existirem ramos abertos, a fórmula pode ser tanto satisfatória (SAT) quanto uma contradição; uma "não tautologia". Mesmo se a árvore terminar com todos os ramos abertos, isso não implica, necessariamente, que a fórmula seja uma contradição. Supondo-se uma fórmula na forma $\Gamma \vdash \Delta$, no caso da árvore de prova terminar com algum ramo aberto, para verificar uma contradição bastaria verifica se $\Delta \vdash \Gamma$ representa uma tautologia, senão, a fórmula é necessariamente satisfatória. Existem outros métodos mais eficientes de verificação de problema SAT por meio de tableau semântico, porém, o presente trabalho foca prova da validade, ou tautologia, de fórmulas.

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 (conjunções ou negações).

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

γ (gama): fórmulas quantificadas universalmente.

δ (delta): fórmulas 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 (Verdadeiro ou Falso). 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 - Inferências sinalizadas. Fonte - D'AGOSTINO, et al., 1999, p.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.

Apesar da simplicidade estrutural, o tableau de Smullyan apresenta algumas limitações típicas em outros métodos de prova. Um exemplo é a repetição,

em ramos diferentes, de inferências já executadas e que não alteram o resultado final.

A figura 5 apresenta um exemplo de solução por tableau de Smullyan para o sistema (A v B), (A v ¬B), (¬A v C), (¬A v ¬C) ⊢⊥. Note-se a repetição de ramos que ocorrem após os nós 6 e 25.

A ordem de escolha dos nós para aplicação das regras de inferência não muda o resultado final da prova. Contudo, é um relevante fator para o tamanho da árvore de prova.

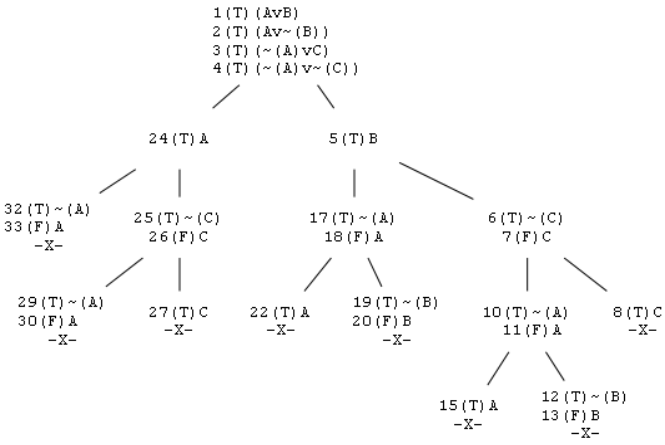

Figura 5 – Exemplo Tableau de Smullyan. Fonte - Próprio autor (gerado pelo software do trabalho).

Note-se que as figuras 3 e 6 representam a prova da mesma fórmula. Contudo, a figura 6 apresenta uma seleção dos nós que prioriza as fórmulas do tipo α e, no caso, a árvore reduz de 29 nós (incluindo os fechamentos) para 17 nós.

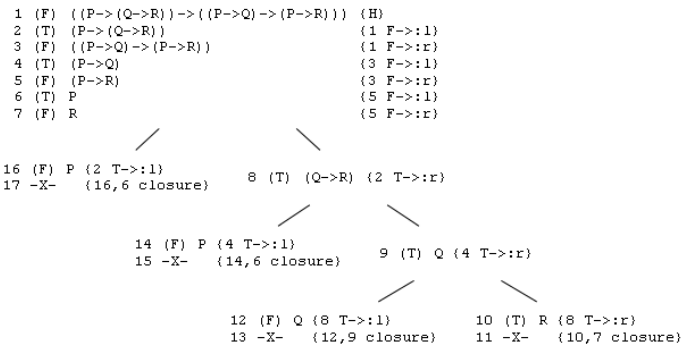

Figura 6 - Exemplo de prioridade na aplicação das regras de inferência. Fonte - Próprio autor (gerado pelo software do trabalho).

Por outro lado, durante as experimentações, verificou-se que a prioridade de aplicação de fórmulas, em especial as fórmulas iniciadas com negação (*not*), em muitos casos é útil, mas em outros pode gerar nós desnecessários. Ou seja, a priorização não é uma técnica absolutamente positiva. Normalmente, é possível avaliar, durante a execução da prova, as circunstâncias nas quais uma técnica de redução pode ser realmente efetiva. Entretanto, essa avaliação pode demandar um esforço computacional desproporcional ao benefício da técnica. Ao final da prova, também é possível avaliar a utilidade de aplicação de regras de inferência e remover os nós que não contribuem para a prova.

Outra técnica de redução adotada é manter a regularidade dos ramos, ou seja, não permitir repetição de nós idênticos ou imediatamente equivalentes. Por exemplo, os nós "(F)A" e "(T)¬A" são imediatamente equivalentes. Para garantir a regularidade, (1) fórmulas do tipo **α** só devem expandir as subfórmulas que não existem no ramo e (2) fórmulas do tipo **β** só devem gerar ramificações se nenhuma de suas subfórmulas existirem no ramo.

Existem diversas outras técnicas que buscam reduzir a árvore de prova como, por exemplo, "clash priority strategy" e "ancessor clash restricted" (D'AGOSTINO, et al., 1999, p.71). Esse é um tema comum em vários artigos e teses relacionados com tableau semântico.

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, p.91). A ideia geral do método é: quando ocorrer uma ramificação, é possível inserir um nó extra com uma fórmula complementar (negativa) à fórmula inserida no outro ramo. Esse nó poderá evitar a ocorrência de repetição durante a

expansão de nós. A figura 7 apresenta uma abordagem operacional da técnica.

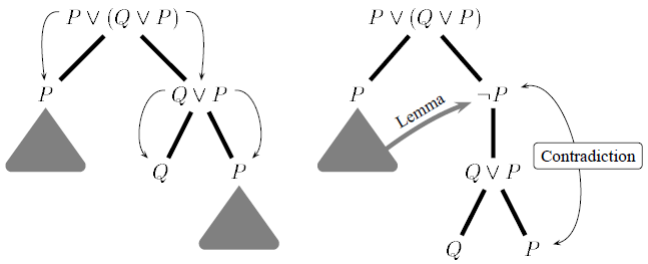

Figura 7 – Aplicação de lema em tableau. Fonte - D'AGOSTINO, et al., 1999, p.619.

Considere-se o ramo *k* com um nó *n* que possui a fórmula θ (theta). Se o ramo *k*, ao longo da prova, se tornar fechado, então $\Gamma \vdash \neg\theta$. Nesse caso, é possível afirmar que ¬θ é um lema, ou seja, uma fórmula previamente considerada como verdadeira. Consequentemente, ¬θ pode ser utilizado em qualquer outro ramo da árvore de prova como uma fórmula válida.

Contudo, inserção de nó com lema pode gerar efeito "indesejado" caso o ramo *k* não se feche, ou seja, caso $\Gamma \nvdash \neg\theta$. Segundo FUCHS (1998) e D'AGOSTINO (1999) essa inserção de fórmula não afeta o resultado final se a prova busca verificar um 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 ao longo da execução da prova (FUCHS, 1998, p.7). As técnicas de geração de lema são diversas. FUNCHS (1998) apresenta uma técnica que, no geral, adia a inclusão de lemas até que eles sejam úteis, mantendo uma coleção de fórmulas candidatas a lema, obtidas ao longo da prova.

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. Por isso, no geral, o lema não deve gerar ramificações.

> *In order to obtain a controlled use of lemmas we want to employ the lemmas only could reorder the search space in a hardly controllable manner. Henceforth, in all connection tableau calculi a start expansion with lemmas is forbidden.* (FUCHS, 1998, p.12)

Um exemplo da dificuldade de controle na inserção de lemas é o seu conflito com a técnica de regularidade. No caso do tableau com lema, a regularidade deve permitir expansão de nós β, mesmo se ocorrer repetição, quando suas subfórmulas forem iguais a fórmulas dos lemas. Os nós lemas são atômicos e não podem sofrer expansão, mas também não podem impedir expansões válidas de fórmulas não atômicas.

2.3.6 TABLEAU KE

Mesmo com as técnicas de redução da árvore de provas já apresentadas, existem uma série "anomalias e limitações" (D'AGOSTINO, et al., 1999, p.75) difíceis para o método de prova por meio de tableau semântico de Smullyan ou do tableau com lema. Um exemplo é a família de fórmulas na Forma Normal Conjuntiva (CNF): se o tableau se fechar, existirão ao menos *k*! ramos, sendo *k* o número de variáveis da fórmula (proposições). Esse é um exemplo no qual a tabela verdade, que contém 2^k linhas, tende a ser menor que o número de ramos do tableau clássico. Por isso, buscou-se adotar outro método chamado 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 uma das suas subfórmulas ao longo do caminho da prova. Exemplos de regras inferências com cortes são *modus ponens*, *modus tollens* e silogismo disjuntivo.

Gerhard Gentzen, em 1935, 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.

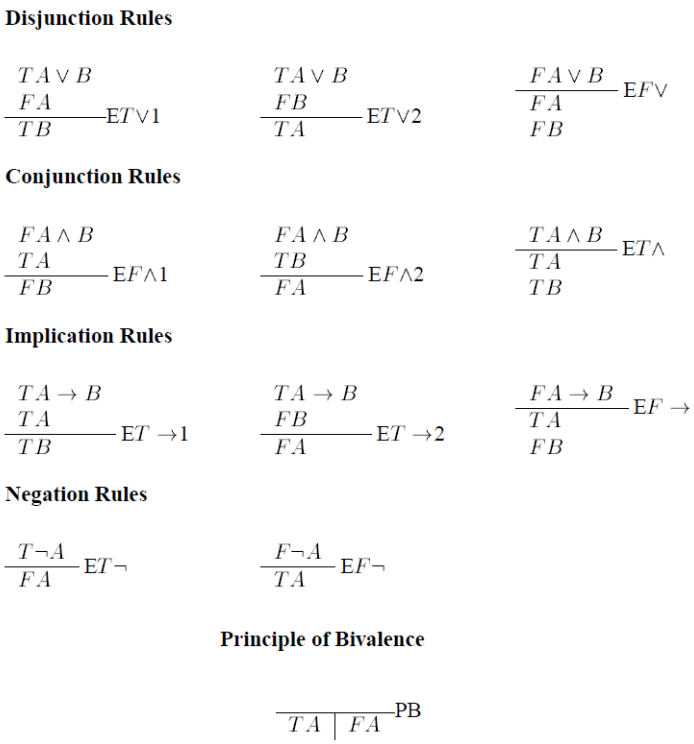

Figura 8 - Regras de inferência do tableau KE. Fonte - D'AGOSTINO, et al., 1999, p.89.

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. A figura 8 apresenta as regras de inferência do tableau KE.

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. O único meio de geração de ramos na árvore de prova KE é utilizando-se 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, p.89.

"Permitir cortes em um sistema de prova automatizado é, de certo modo, permitir o uso de Lemas" (D'AGOSTINO, et al., 1999, p.11). Por isso, similar ao tableau com lemas, a escolha da fórmula do nó PB é o ponto fundamental no método KE. O tableau KE possui regras específicas. Considere-se o ramo ϕ:

1) Uma fórmula é *E-analysed*, se:
   a) É do tipo **α** e as subfórmulas α_1 e α_2 ocorrem em ϕ.
   b) É do tipo **β** e:
      i) se β_1 ocorre em ϕ, então $\neg\beta_2$ ocorre em ϕ, ou,
      ii) se $\neg\beta_1$ ocorre em ϕ, então β_2 ocorre em ϕ.
2) O ramo ϕ é *E-completed* se todas as suas fórmulas são *E-analysed.*
3) Uma fórmula do tipo β é realizada (*fulfilled*) se β_1 ou β_2 ocorre no ramo ϕ.
4) O ramo ϕ é considerado completo (ou exaurido) se é *E-completed* e todas as fórmulas do tipo β são realizadas (*fulfilled*).

O caso de fechamento é o mesmo de qualquer tableau, a existência, no mesmo ramo, de uma contradição. O término da árvore de prova do tableau

KE também é típico: ocorrem quando todos os ramos estiverem completos.

As fórmulas do nó PB são originadas de fórmulas β não realizadas (*not fulfilled*), ou seja, sempre será β1 ou β2. A referência principal adotada para o estudo do método KE (D'AGOSTINO, et al., 1999) destaca a importância da seleção adequada dos nós PB. Porém, apresenta apenas uma heurística simples e intuitiva baseada em fórmulas atômicas com maior repetição. Todavia, nem sempre as fórmulas disponíveis para seleção do nó PB são atômicas. Em vários casos, a adoção de fórmulas não atômicas é a única alternativa para a conclusão do tableau KE.

Como método de seleção do nó PB, ABATE e GORÉ (2007), no artigo sobre implementação de tableaux genéricos, sugerem 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 também sofre de limitações típicas como, por exemplo, a possibilidade de resultar em provas mais extensas e dificuldade de selecionar a fórmula para o nó 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, na figura 11, que a solução do sistema (A v B), (A v ¬B), (¬A v C), (¬A v ¬C) $\vdash\perp$ pelo método do tableau KE é bem menor que a solução apresentada na figura 5. Contudo, também é importante destacar que o sistema apresentado se encontra na forma conjuntiva normal (CNF), caso ótimo do método KE.

D'AGOSTINO (1999) apresenta a árvore KE com nós sem sinal. Ou seja, no lugar de nós com sinais tipo `(F)A`, utiliza-se a fórmula na forma `¬A`. Durante o desenvolvimento do software, percebeu-se que, no tableau KE, o uso de nós com sinais dificulta a comparação de igualdade de subfórmulas como, por exemplo, (F)A = (T)¬A. Note-se que as fórmulas A e ¬A são diferentes e é necessário comparar ambas as formas, com ou sem sinal. Comparando-se as fórmulas sem o sinal do nó, o processo é mais direto, pois, simbolicamente, "A = A" e "A ≠ ¬A". A necessidade de comparação de fórmulas afeta tanto a busca na verificação das regras de inferência com cortes quanto a escolha do nó PB, especialmente na verificação das fórmulas β realizadas (*fulfilled*).

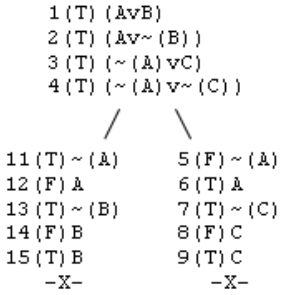

Figura 10 - Tableau KE. Fonte - Próprio autor (gerado pelo software do trabalho).

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. Além disso, esses operadores *n-ários* não resolvem completamente o problema de comparação entre fórmulas.

3 MODELAGEM DO SOFTWARE

O software elaborado para o trabalho foi codificado em Java™. O design do software buscou priorizar a generalização e adaptabilidade do software,

considerando em segundo plano a economia de recursos computacionais.

O software conta com duas aplicações (executáveis): (1) o sistema de prova e (2) um módulo de testes parametrizável com coleta de resultados. O presente capítulo apresenta o software de prova. Os testes serão tratados na análise de resultados.

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 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. Os objetos dessas classes são gerados durante a compilação da linguagem e utilizados pelos demais módulos do software. O

compilador retorna sempre um objeto da classe "LogicalSystem". No entanto, apesar de atuar como AST (*Abstract Syntax Tree*), alguns métodos típicos da interpretação foram implementados nessas classes como, por exemplo, a interpretação de modelos proposicionais (semântica) e o cálculo do tamanho da fórmula ($|\varphi|$). Um plano futuro é desacoplar essa dependência e separar a interpretação da AST. A figura 11 apresenta um diagrama das classes do sistema lógico.

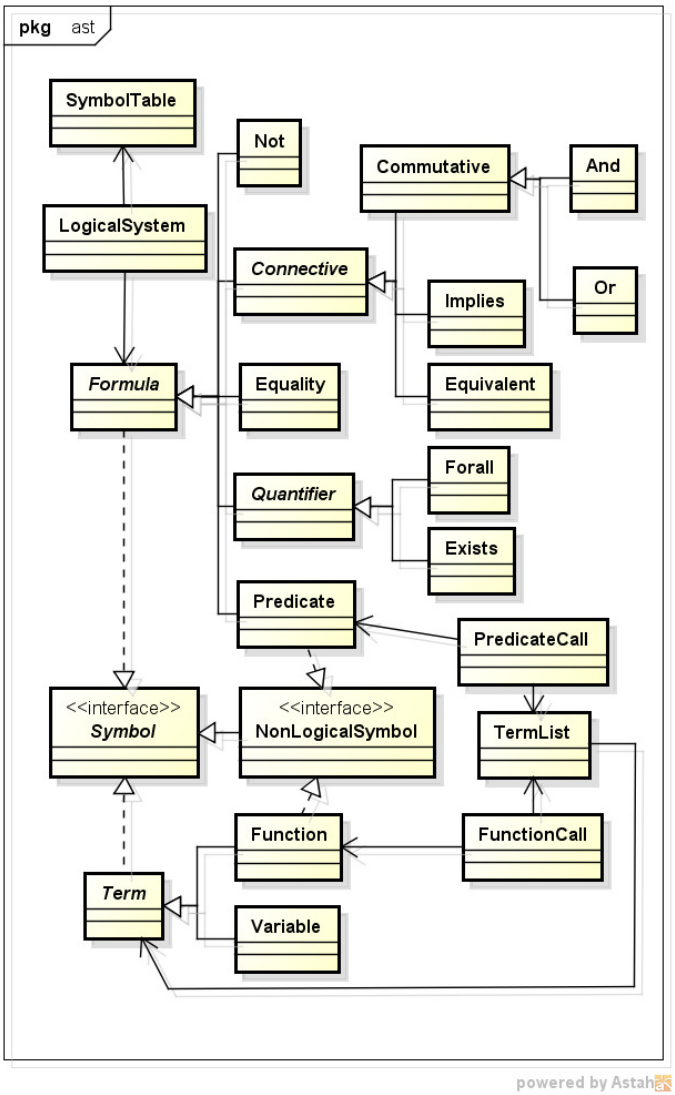

Figura 11 - Diagrama da AST da linguagem do sistema lógico. Fonte - Próprio autor.

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 (ver capítulo 2.3.6 - Tableau KE).

3.3 OS MÓDULOS DE PROVA

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

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 usados pelos motores de inferência. A classe "LogicalReasoning" (raciocínio lógico) é a classe base para todos os métodos de prova implementados no software.

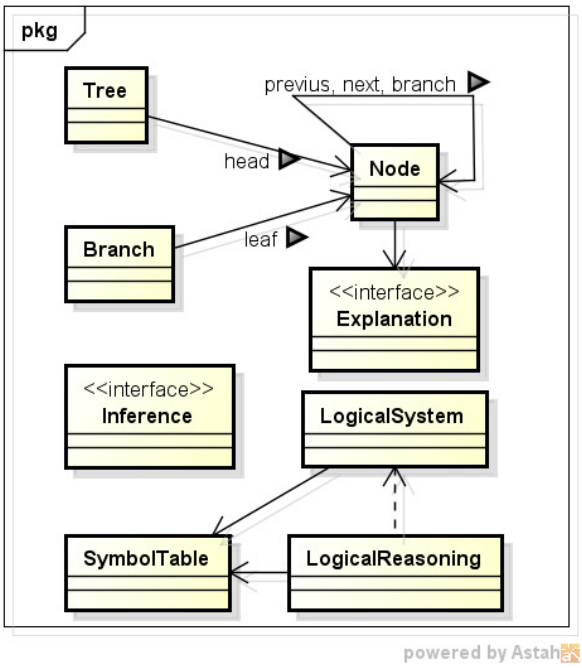

Figura 12 - Diagrama das classes de prova. Fonte - Próprio autor.

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.

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.

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. Essa também é uma oportunidade para trabalhos futuros.

Como os controles são separados, 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.

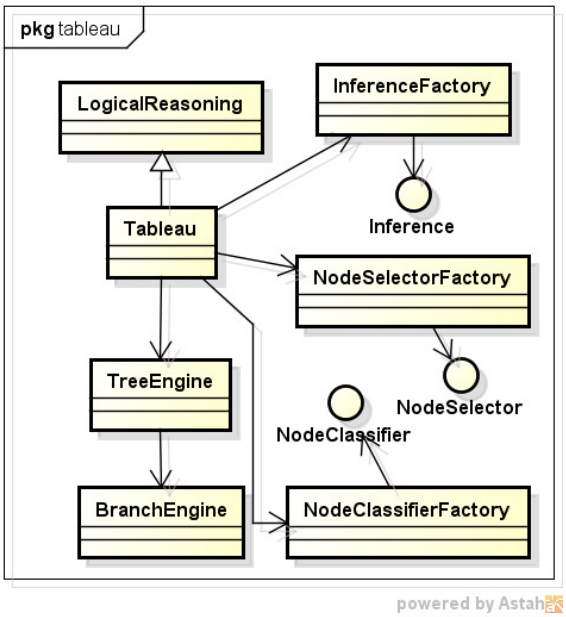

Figura 13 - Diagrama das classes de prova. Fonte - Próprio autor.

Na classe padrão de seleção de nós, a fórmula φ de maior prioridade é a da classe "Not". Dessa forma, sempre que um nó com fórmula "not" é adicionado, se não ocorrer o fechamento do ramo, o próximo nó adicionado normalmente possui a fórmula sem o "not" e com sinal invertido. Esse tipo de aplicação de prioridade, baseada apenas na classe da fórmula isolada do nó, pode resultar no aumento do tamanho das provas. O ideal seria adotar uma prioridade mais dinâmica, baseada em mais informações da árvore obtidas durante o processo de prova.

A seleção de nós pode ou não usar as informações de classificação. Contudo, por motivos óbvios, nós atômicos nunca são selecionados para inferência. Diferentes ordens de escolha de nós para aplicação das inferências causam resultados bem diversos no resultado final da árvore de prova (ver capítulo 2.3.4 Tableau de Smullyan). Ou seja, o estudo de técnicas de seleção de nós é um interessante ponto para eventuais pesquisas futuras.

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 ($\leftrightarrow$). 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, priorizando nós φ e verificando regularidade antes da expansão de nós β.

3.3.2.2 TABLEAU COM LEMMA

Esse módulo possui suas classes específicas de inferência e de seleção de nós. A classe de seleção de nós tenta contornar o conflito entre o controle de regularidade e os nós com lemas (ver capítulo 2.3.5 Tableau com Lema). Os objetos de 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, por exemplo, o tamanho da fórmula ($|\varphi|$), o número de ocorrências da fórmula, etc.

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. Essa seleção usa a heurística de buscar a fórmula de menor tamanho que ocorra maior número de vezes entre as subfórmulas β_1 (da esquerda) e β_2 (da direita) das fórmulas β não realizadas (*not fulfilled*).

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. A 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.

Para verificar a correção das provas, foi elaborado um software de testes que gera, automaticamente, fórmulas sem parênteses. Foram adotados dois geradores de fórmulas: um combinatório e outro aleatório. As fórmulas geradas são testadas nos três métodos de tableau e a base de comparação é o

resultado da tabela verdade, que é o algoritmo mais simples e confiável. O software de testes também coleta diversas informações sobre resultados, características das árvores de prova e tempos de execução. São mais de trinta informações diferentes que podem ser usadas para diversas análises. O presente artigo irá apresentar apenas algumas considerações sucintas e gerais sobre os resultados. Foram elaboradas algumas configurações úteis ao procedimento de testes como, por exemplo, número máximo de testes, tempo de duração da execução (timeout) e condição de saída de dados.

Os geradores criam fórmulas com número fixo de termos no formato "$T_1O_1T_2O_2T_3...T_{(t-1)}O_{(t-1)}T_t$". Cada termo 'T' é composto de uma proposição 'P' e um sinal 'S' $\in$ {"~"," "} (com ou sem sinal "*not*"). O conectivo lógico é 'O' $\in$ {"$\wedge$","$\vee$","$\rightarrow$","$\leftrightarrow$"}. Os nomes das proposições (variáveis) são formados por letras maiúsculas, sendo P $\in$ {A,B,...,Z,AA,AB,...}. Os algoritmos recebem como entrada o número 't' de termos e 'p' de proposições da fórmula a ser gerada.

No gerador combinatório, os 't' termos são combinados com as 'p' proposições, com e sem sinal (s=2), mais todas as possíveis combinações dos 'o' conectivos lógicos (o=4). Para uma dada combinação (t,p) são geradas $(p \times s)^t \times o^{(t-1)}$ fórmulas. Essa técnica cobre uma vasta gama de famílias de fórmulas, porém com um enorme custo (tempo) computacional que limita o tamanho das fórmulas viáveis. Foram testadas, sem ocorrência de erros, as combinações (t,p) de (4,2), (4,4), (5,2), (5,3), (6,2), (8,1) e (6,4).

Para viabilizar a execução de testes em fórmulas maiores, adotou-se um gerador aleatório. Nesse caso, as fórmulas possuem 't' termos, cada um com um sinal 'S', uma proposição 'P' e são conectadas por operadores lógicos 'O', sendo que 'P', 'S' e 'O' são sorteados nos seus respectivos conjuntos. A dispersão aleatória depende do método "Math.random()" da VM Java. A dispersão foi verificada para conjuntos de 10^6 fórmulas com a combinação (t,p) de (10,1), (15,1), (20,1), (30,1), (20,2), (20,10) e (20,100). Ocorreram menos de 16% de repetições de fórmula idênticas na combinação (10,1). As repetições foram menores que 0,001% na combinação (15,1). Não ocorreram repetição de fórmulas nos demais casos. O número de repetições tende a diminuir exponencialmente com o aumento de 't'. Foram executados testes com fórmulas aleatórias para combinação (t,p) de (15,1), (30,1), (20,2), (20,10), (20,20) e (20,100), (100,2), (100,5) e (100,10). Cada execução foi limitada a períodos de, no mínimo, 20 minutos. Não foram encontrados erros no teste final.

A maioria dos erros de prova encontrado e corrigidos durante a fase inicial de testes foi no tableau com lema. Os erros mais difíceis de serem analisados e corrigidos foram no tableau KE.

Por causa da abordagem mais simples e exemplificativa, os resultados do tableau com lema não foram satisfatórios. Na média, as árvores de provas foram maiores do que as do tableau de Smullyan, o método mais simples adotado. Por outro lado, o tableau com lema foi introduzido, principalmente, como preparação para o tableau KE, pois existem outras técnicas para reduzir o tamanho da solução do tableau clássico como, por exemplo, a técnica de "merging" (D'AGOSTINO, et al., 1999, p.90).

Infelizmente, na média, as árvores de prova do tableau KE foram as maiores. Isso aponta para a limitação do método em função da complexidade e importância da seleção do nó PB.

No geral, o tableau de Smullyan apresentou os melhores resultados como, por exemplo, provas mais simples e de menor tamanho, programação mais direta e menos propensa a erros e menor tempo médio de execução. Esse resultado reforça a dificuldade em se estabelecer técnicas efetivas e gerais de redução do tamanho de prova.

O número de nós das fórmulas com resultado de contradição e SAT foram maiores que as tautologias. Esse resultado é natural, pois os ramos abertos, que caracterizam as "não tautologias", só terminam após a exaustão das possibilidades de inferência. Já os ramos fechados podem terminar antecipadamente com a presença de uma contradição.

Considere-se fórmulas com combinação (t,p) nas quais 't=p'. Nesses casos, para testes com até dez termos (t=p=10), a tabela verdade foi o método de prova com menor tempo médio de execução. Todavia, para 't≥14', mantendo 't=p', o tempo médio de execução da tabela verdade sempre foi maior que os demais, demonstrando que a ordem de complexidade relativa dos métodos de tableau semântico é menor que a da tabela verdade, assim com apresentado em D'AGOSTINO (1999).

A observação do funcionamento do software também levanta a questão de controle de execução durante as provas. Por exemplo, um controle relevante seria a limitação de recursos como, por exemplo, consumo de memória e tempo de processamento. Considerando um exemplo ainda mais específico, caso se deseje provar uma tautologia, é possível incluir controles que verifiquem antecipadamente interpretações falsas e encerre o processamento nesses casos. Porém, o objetivo do trabalho é apresentar as soluções completas das fórmulas.

Considerando o software, um dos principais resultados do trabalho, muitas são as oportunidades de melhoria como, por exemplo, adotar testes de unidade mais rigorosos, tentar otimizar a eficiência dos algoritmos, melhorar a documentação do software e incluir comentários relevantes nos programas.

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

5 Conclusões 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 envolveu muita pesquisa e estudo sobre lógica, em especial os fundamentos, algoritmos de prova automatizada de teoremas e solução de problemas SAT. Por isso, este objetivo, do ponto de vista de graduação, tem grande importância. Principalmente considerando que o tema é quase sempre restrito a cursos e trabalhos de pós-graduação.

A elaboração do software enfrentou mais dificuldade com os conceitos lógicos do que com o design ou programação da aplicação. 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 tanto para alunos de graduação quanto para outras pesquisas de pós-graduação relacionadas.

Os resultados dos experimentos demonstraram a correção do software. O software gera diversos dados que podem ser de grande valor em análises futuras como, por exemplo, comparação de desempenho entre os métodos, dispersão de resultados por amostragem, avaliação resultado das diferentes técnicas de redução da árvore de prova, etc.

A elaboração do software apresentou 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 científico 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:

1) Elaborar uma interface gráfica intuitiva e com resultado imediato, incluindo a possibilidade de gerar a árvore sintática de cada fórmula.
2) Melhorar o tratamento de erro do compilador, principalmente para apresentar informações mais detalhadas de erros nas fórmulas.
3) Ampliar a pesquisa para adaptar os algoritmos de solução para lógica de primeira ordem.
4) Adotar outros métodos de prova como, por exemplo, as provas sequenciais tipicamente usadas no curso de matemática discreta.
5) Pesquisar formas de geração do nó PB para ampliar a eficiência do tableau KE.
6) Ampliar a linha de pesquisa para aplicação do software para outras formas de lógica.

6 AGRADECIMENTOS

Agradeço ao Professor Moisés Ramos, Coordenador deste Projeto de Pesquisa, que, além de propor esse fascinante tema, mesmo com os riscos de prazo e complexidade, depositou enorme confiança e apoio, principalmente nos momentos difíceis.

Agradeço à Professora Miriam Lourenço Maia, Coorientadora, pelos conselhos, que foram decisivos.

Agradeço ao Professor Joaquim José da Cunha Júnior, membro da banca avaliadora, pelos elogios, pelo interesse e pelo enorme incentivo a manter a pesquisa nessa área.

Agradeço à Professora Sandra Maria Rodrigues de Morais, pois seria impossível elaborar esse trabalho sem seu apoio e carinho.

Agradeço à Professora Ana Paula Ladeira por ensinar, de forma dedicada e exemplar, o valor e a importância da pesquisa científica na prática.

Agradeço à minha mãe, Ana Adelaide, professora doutora aposentada da Escola de Enfermagem da UFMG, não só por esse projeto de pesquisa, mas por todos os demais projetos de minha vida, principalmente os acadêmicos que, sem nenhuma exceção, foi sempre mais do que presente. Obrigado pelo entusiasmo, pela participação, pelo suporte, pelas revisões, pela sabedoria, pela paciência, pelo exemplo de vida, por tudo.

REFERÊNCIAS

ABATE, Pietro; GORÉ, Rajeev. *Theory and Pratice of a Generic Tableau Engine: The Tableau Workbench*. The Australian National University and NICTA, Canberra, Australia, 2007. Disponível em: “http://twb.rsise.anu.edu.au/files/tableaux07.pdf”. Acessado em: 07/12/2012.

D’AGOSTINO, Marcello; et al. *Handbook of Tableau Methods.* Dordrecht: Kluwer Academic Publishers, Netherlands, 1999. 678 p. 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.

WIKIPEDIA: *Completude (lógica)*. Disponível em: “http://pt.wikipedia.org/wiki/Completude_(l%C3%B3gica)”. Acessado em: 07/12/2012.