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.

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, compiler.

---

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. A lógica é fundamental em praticamente todas as ciências, incluindo a ciência da computação. George Boole aproximou a matemática da lógica proposicional clássica, possibilitando o tratamento algébrico, por meio simbólico, e o cálculo proposicional.

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

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 problema SAT praticamente não é abordado durante a graduação em Ciência da Computação. 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.

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. 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.

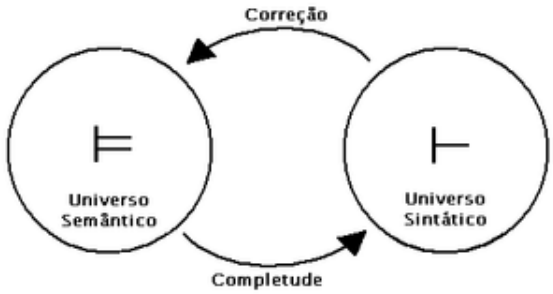

Figura 1 - Relação no cálculo lógico (WIKI-C)

O universo semântico está relacionado com as possíveis combinações de valores das variáveis do sistema, ou seja, sua interpretação. O resultado de um sistema lógico é sempre e apenas verdadeiro ou falso. 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.

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, segundo a tese de Church e Turing, a lógica de primeira ordem é considerada *indecidível*.

Para a lógica proposicional, além de poder ser considerada com subconjunto da lógica de primeira ordem, ou lógica de ordem zero, existem diversas provas de sua completude e correção. 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. Entretanto, 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, em especial alunos de graduação, 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.

Outro objetivo inicial, porém secundário, do software era elaborar provas mais legíveis e de menor tamanho 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 ou sequencial. 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.

2.3 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. 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. Existem diversas teses demonstrando a aplicação de tableau a diversos tipos de lógica e uma enorme variabilidade de técnicas e heurísticas de redução da árvore de prova.

2.3.1 FUNDAMENTOS DO MÉTODO TABLEAU

O método tableau é baseado na prova por contradição. Considerando o sistema lógico, a prova ocorre se $\Gamma \vDash \varphi$, então $\Gamma \cup \{\neg\varphi\} \vDash \bot$.

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. As regras de inferência são ilustradas na figura 2.

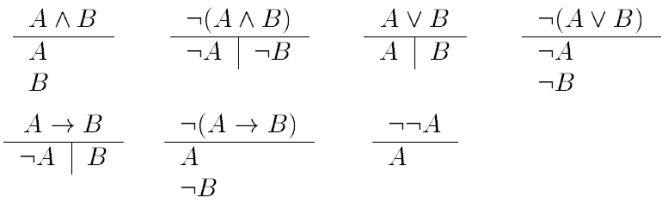

Figura 2 - Regras de inferência. Fonte - D'AGOSTINO, et al., 1999, pág. 57.

Cada aplicação das regras de inferência expande (aumenta) a árvore de prova adicionando um novo nó. As inferências de fórmulas conjuntivas ("e") e negativas ("não") 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 ao longo do processo de prova 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 outro ramo. As fórmulas dos nós antecedentes que ainda não foram inferidas no ramo corrente podem ser novamente inferidas. 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. 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 nó.

No geral, 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. 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:

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 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 TABLEAU DE SMULLYAN

Segundo Marcello D'Agostino (D'AGOSTINO, et al., 1999), o nome "tableau" foi proposto por Evert W. Beth, um dos principais idealizadores do método. Inicialmente, ele apresentou a técnica em forma de tabelas. 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.

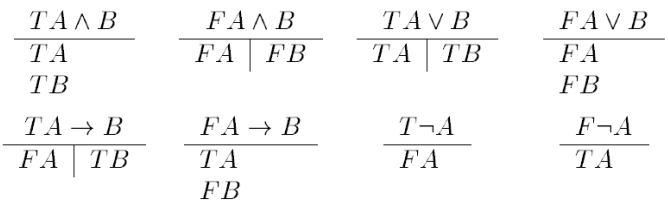

Figura 3 - Inferências sinalizadas. Fonte - D'AGOSTINO, et al., 1999, pág. 57.

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.

Na prática de elaboração do software, 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.

2.3.4 TABLEAU COM LEMA

A ideia geral do tableau com lema 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 repetição durante a expansão de nós. Note-se a figura 4 que apresenta uma abordagem operacional da técnica.

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.

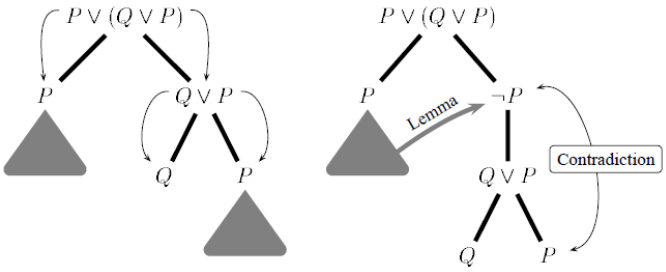

Figura 4 – Aplicação de *lemma* em tableau. Fonte - D'AGOSTINO, et al., 1999, pág. 619.

A 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 lema não afeta o resultado final se a prova buscar 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, pág. 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 devem 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, pág. 12)

2.3.5 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.

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. A figura 6 apresenta as regras de inferência do tableau KE.

Para que as provas sejam completas e corretas, 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 5 – Princípio da bivalência (PB) Fonte - D'AGOSTINO, et al., 1999, pág. 89.

"Permitir cortes em um sistema de prova automatizado é, de certo modo, permitir o uso de Lemas" (D'AGOSTINO, et al., 1999, pág. 11). Por isso, similar ao tableau com lemas, a escolha da fórmula do PB é o ponto fundamental.

O tableau KE possui regras específicas. Para um 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 formulas são *E-analysed.*
3) Uma fórmula do tipo β é realizada (*fulfilled*) se β_1 ou β_2 ocorre no o 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 ou o fechamento da árvore de prova do tableau KE também é típico: ocorrem quando todos os ramos estiverem fechados ou completos.

As fórmulas do 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 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.

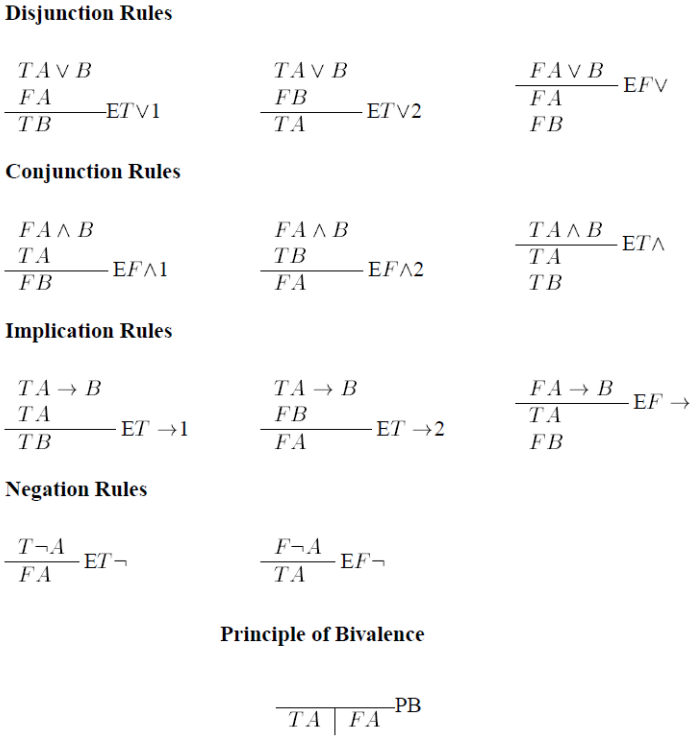

Figura 6 - Regras de inferência do tableau KE. Fonte - D'AGOSTINO, et al., 1999, pág. 89.

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 também sofre de limitações típicas. Por exemplo, a possibilidade de resultar em provas mais extensas e dificuldade de selecionar a fórmula 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.

D'AGOSTINO (1999) apresenta a árvore KE com nós sem sinal, 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 escolha do nó PB quanto a busca nas regras de inferência com cortes.

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. Esse tipo de operador *n-ário* também é típico nas fórmulas em CNF. Por outro lado, essas técnicas são conflitantes com o objetivo genérico de tornar as soluções das provas mais humanamente legíveis. Além disso, operadores *n-ário* não resolvem o problema da comparação completa de 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.

3.1 A LINGUAGEM E O COMPILADOR

Inicialmente, o objetivo era lidar com lógica de primeira ordem. 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. 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. 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 e utilizadas pelos outros módulos do software. Elas atuam como árvore sintática abstrata para o compilador. O compilador retorna sempre um objeto da classe "LogicalSystem". No entanto, apesar de atuar como AST (Abstract Syntax Tree), alguns métodos operacionais, que são típicos da interpretação, foram implementados nessas classes como, por exemplo, a avaliação de modelos (interpretação) e o cálculo do tamanho da fórmula (|φ|).

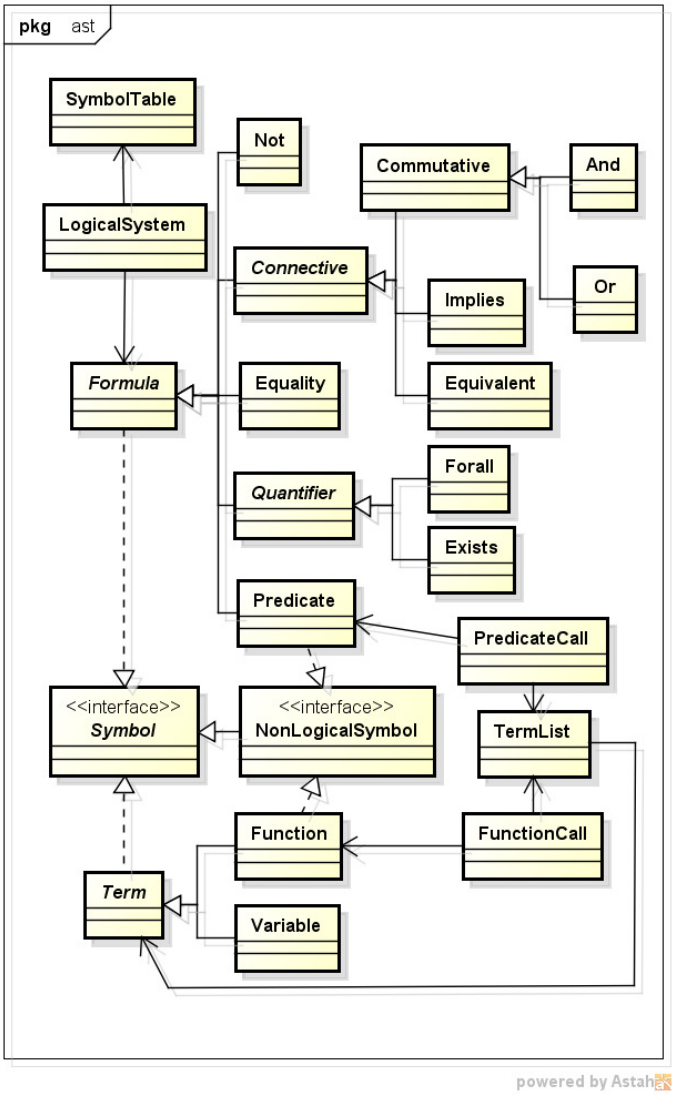

Figura 7 - Diagrama da AST da linguagem do sistema lógico. Fonte - Próprio autor.

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 isolados, 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.

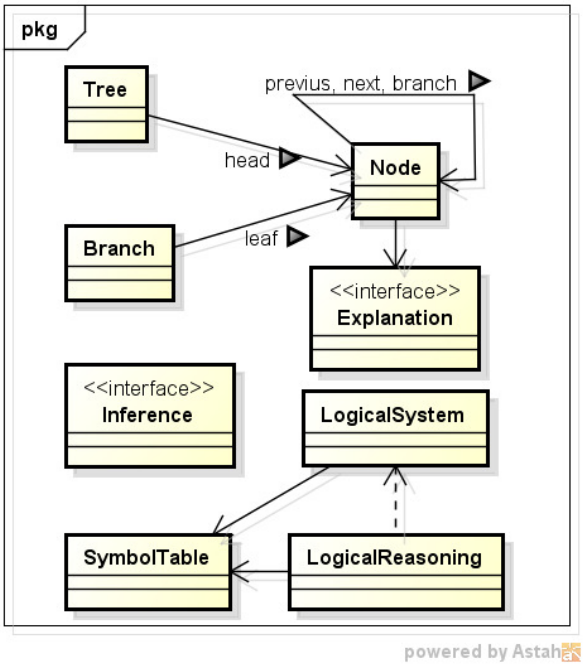

Figura 8 - Diagrama das classes de prova. Fonte - Próprio autor.

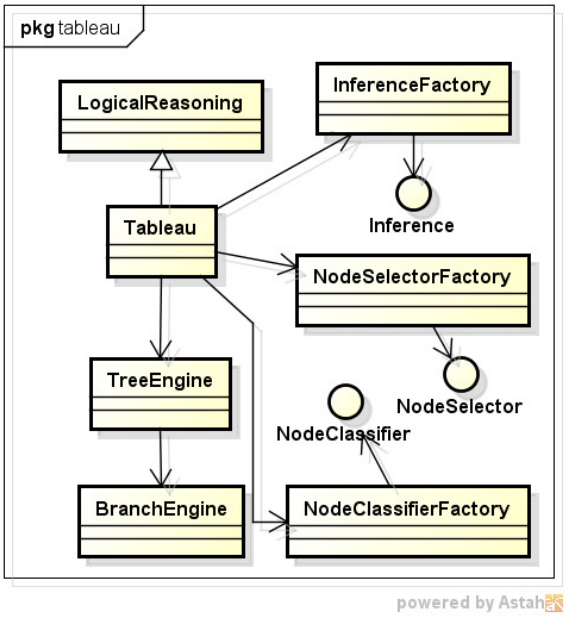

Figura 9 - Diagrama das classes de prova. Fonte - Próprio autor.

Uma classe utilitária relevante é a "DotTreeGenerator", que gera arquivos na linguagem "DOT" para elaborar imagens das árvores de prova no programa Graphiz.

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.
3. Classificação dos nós de acordo com as fórmulas ou o contexto da execução.
4. Seleção de nós que serão usados na inferência, incluindo o controle dos nós que foram ou ainda devem ser inferidos.

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.

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. A classificação padrão atribui prioridade mais alta para os nós φ e mais baixa para os β.

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. Esse tipo de seleção baseada apenas na classe da fórmula do nó pode resultar no aumento do tamanho das provas. O ideal seria adotar uma prioridade dinâmica, baseada em mais informações da árvore de prova. Ou seja, o estudo de técnicas de seleção de nós é um interessante ponto para eventuais de 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 (↔). 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.

3.3.2.2 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 da fórmula e o 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. Por outro lado, o tableau com lema foi introduzido mais como preparação para o tableau KE, pois existem outras técnicas para reduzir a solução do tableau clássico como, por exemplo, a técnica de "merging" (D'AGOSTINO, et al., 1999, pág. 90).

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. A seleção usa a heurística de buscar fórmulas de menor tamanho que ocorra maior número de vezes entre os nós β não realizados.

4 RESULTADOS OBTIDOS

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

A observação do funcionamento do software também levanta a questão de controle de execução das provas. Um controle relevante seria a limitação de recursos como consumo de memória e tempo de processamento. Por exemplo, caso se deseje provar teoremas, 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. O processamento poderia se encerrado nesse momento.

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 tema do 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.

Considerando o objetivo inicial de elaborar um software de prova em lógica proposicional, a conclusão parece ser positiva. 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.

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) Criar um módulo de teste.
3) Adaptar o software para lógica de primeira ordem.
4) Pesquisar formas de geração de lemas e de nós PB.
5) Ampliar a linha de pesquisa para outras formas de lógica.
6) Melhorar o tratamento de erro do compilador.

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.

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.