Trabalho sobre Provadores Automáticos de Teoremas

Cláudio Busatto, Henrique Weber, Jefferson Stoffel, João Luiz Grave Gross

Trabalho realizado para cadeira Lógica para a Computação – INF05508 – Professor Luis Lamb – Turma A

Índice

Resumo. *Apresentamos uma visão geral sobre provadores automáticos de teoremas. Para tanto, damos uma visão histórica sobre provadores automáticos e suas implementações. No capítulo sobre técnicas falamos dos conceitos básicos sobre provadores automáticos e por que técnicas eles funcionam. Por fim, damos alguns breves exemplos de aplicações práticas de provadores automáticos.*

1. Introdução

A prova automática de teoremas (automated theorem proving, ATP) trata do desenvolvimento de programas capazes de demonstrar que uma determinada proposição é conseqüência lógica de outras proposições. A ATP é atualmente o campo mais desenvolvido da área de raciocínio automático.

Um sistema de ATP processa uma série de conjecturas, hipóteses e axiomas, escritos em uma linguagem formal que permite expressar de maneira precisa e não-ambígua o problema a ser resolvido, para gerar uma prova, que descreve como e por que as conjecturas seguem dos axiomas e hipóteses em uma forma que possa ser compreendida por seres humanos e, possivelmente, processada por outros programas. A linguagem utilizada representa uma lógica, freqüentemente a lógica clássica de primeira ordem, por ser expressiva o suficiente para permitir a especificação de problemas arbitrários de maneira razoavelmente intuitiva, mas outros tipos de lógica também são possíveis, como lógicas de alta ordem e lógicas não-clássicas. Muitas vezes a prova gerada não apenas é uma demonstração da validade de uma conjectura, mas também descreve um processo que pode ser implementado para resolver um problema.

Apesar do nome, os sistemas de prova automática de teorema são freqüentemente usados de maneira interativa por um especialista no domínio da aplicação, que guia o processo de inferência, ou determina lemas intermediários que devem ser provados no decorrer da demonstração. Isto permite que o problema seja resolvido em tempo hábil, já que a resolução de um problema pode consumir tempo exponencial, ou mesmo ser indecidível, dependendo da lógica em questão.

Um problema relacionado é o de verificação de provas, em que o sistema verifica a validade de uma prova já existente. Esse problema é bem mais simples, e é sempre decidível.

2. História

Pode-se dizer que a história da prova automática de teoremas começou em 1957, com a publicação do artigo "The Logic Theory Machine", por Newell, Shaw e Simon. Esse artigo descreve os resultados obtidos com o Logic Theorist, um programa escrito em 1955 e 1956 que tentava simular os processos de dedução humana para provar teoremas de lógica proposicional. O Logic Theorist foi capaz de provar 38 teoremas do Principia Mathematica, e introduziu diversos conceitos básicos da ATP e técnicas de inteligência artificial. Ele foi apresentado em uma conferência em 1956, que inaugurou o termo "inteligência artificial", despertando o interesse de outros na área de ATP.

Em 1960 o matemático Hao Wang desenvolveu programas capazes de provar todos os teoremas de lógica proposicional do Principia Mathematica em poucos minutos, e também a maior parte dos teoremas de lógica de predicados. Usando o tipo de análise lógica iniciado por Herbrand e Gentzen, Wang avançou substancialmente os trabalhos de Newell, Shaw, e Simon. Por esse trabalho ele recebeu a primeira Milestone Award for Automated Theorem Proving, em 1983.

Entre 1956 e 1959, Herbert Gelernter e Nathaniel Rochester desenvolveram o Geometry Theorem-proving Machine (GTM), que usava encadeamento reverso (backward chaining), partindo da conclusão em direção às premissas, criando metas secundárias cuja validade implica a conclusão final. O GTM era capaz de provar a maior parte dos problemas de exame de ensino médio dentro de seu domínio, levando cerca de 20 minutos para resolver os problemas mais difíceis.

Nesse meio tempo, Paul Gilmore trabalhava em métodos não-heurísticos, derivados dos procedimentos de prova da lógica clássica. Gilmore inspirou-se na técnica do tableau semântico de Beth; o procedimento utilizado era rudimentar, mas foi provavelmente o primeiro procedimento de prova mecanizado para o cálculo de predicados, e era capaz de provar teoremas de dificuldade moderada.

Em 1965, J.A. Robinson promoveu a idéia do princípio da resolução, que mais tarde foi melhorado utilizando restrições e estratégias de resolução, que reduzem o número de cláusulas a serem resolvidas. Nesse período, também surgiu um procedimento chamado eliminação de modelos, que levou ao desenvolvimento de um formato de resolução linear muito restrito, chamado SL-resolution, que foi usado nas primeiras versões da linguagem de programação Prolog. Também nesse período, na União Soviética, Maslov inventou o método inverso, bastante similar à resolução.

Entre 1963 e 1967, desenvolveu-se o projeto Semi Automated Mathematics (SAM), em que foi criada uma série de sistemas mais interativos do que os provadores automáticos até então apresentados. Esses sistemas possuíam uma boa interface com o usuário, e uma linguagem com grande poder de expressão. O primeiro dos sistemas SAM era primariamente um verificador de provas, mas a capacidade de prova de teoremas aumentou nas versões posteriores.

No começo dos anos 70, a resolução recebeu críticas e outros métodos de manipulação de fórmulas foram propostas. Woody Bledsoe questionou a capacidade de sistemas lógicos uniformes de manipular a matemática da teoria de conjuntos e apontou que determinados problemas são resolvidos de maneira muito mais simples pelo método de encadeamento reverso do que pelo princípio da resolução. Arthur Nevins desenvolveu paralelamente dois provadores capazes de processar teoremas que a maior parte dos provadores por resolução não eram capazes de manipular. Knuth e Bendix publicaram um artigo focando no potencial das regras de reescrita (rewrite rules).

Em 1975 e 1976 surgiu interesse em procedimentos de prova semelhantes à resolução que utilizam grafos como representação. No final dos anos 70, Peter Andrews desenvolveu o TPS, um provador de teoremas para a teoria de tipos. Embora esteja fundado nas técnicas lógicas (não-heurísticas), o TPS também apresenta traços da abordagem de simulação humana. Ele foi capaz de provar o teorema de Cantor, além de vários teoremas de primeira ordem.

Em 1976, Kenneth Appel e Wolfgang Haken provaram o teorema das quatro cores, que diz

que qualquer divisão de um plano em regiões contíguas pode ser colorida com apenas quatro cores, sem que nenhuma região que faça fronteira com outra possua a mesma cor. A validade do teorema havia sido estudada desde 1852 e permanecia uma questão em aberto até então. Esse foi o primeiro teorema importante provado por um sistema de ATP, ainda que boa parte do problema tenha sido resolvida manualmente. Em 1997, uma prova mais simples foi publicada por Robertson, Sanders, Seymour e Thomas, utilizando outro sistema de ATP, para dissipar as dúvidas que ainda existiam quanto à prova original. Em 2005, o teorema foi novamente provado por Gonthier usando um sistema de ATP de propósito geral.

A partir dos anos 80, os provadores automáticos de teoremas começaram a ser usados comercialmente, principalmente para verificação de hardware, mas também no campo de engenharia de software. Predominaram os sistemas de ATP interativos, embora tenha havido avanços significativos com relação a provadores totalmente automatizados.

O caso mais famoso de bug em hardware aconteceu em 1994, no sistema de divisão em ponto flutuante no processador Pentium. Ficou conhecido como Pentium 'FDIV' bug (FDIV era a instruções de divisão de números com ponto flutuante, que potencialmente gerava resultados incorretos). A possibilidade do erro ocorrer era muito rara, mas a Intel foi forçada a fazer recall do produto devido a grande número de reclamações de usuários e pressão dos distribuidores. Desde então, tanto AMD quando Intel tem utilizado provadores automáticos de teoremas para sua verificação e desenho de seus circuitos integrados.

3. Técnicas Usadas

3.1. Teorema de prova magro

Um teorema de prova 'magro' (Lean Theorem Prover) é um teorema de prova automático implementado com o objetivo de alcançar a máxima eficiência utilizando a mínima quantidade de código e recursos. Todo possível esforço é realizado para eliminar overhead, ou seja, tempo gasto na inicialização do programa, e isso é alcançado através de implementações bem feitas (geralmente complexas) dos sistemas de dedução. Apenas com as mais importantes e eficientes técnicas e métodos são implementadas. Provadores magros geralmente são implementados em Prolog e podem ser muito pequenos, com algumas centenas de bytes de código fonte. Um dos principais provadores automáticos que fazem uso do teorema de prova magro é o ileanCoP.

3.2. Método de Analítico de Tableaux

Em teoria de prova, a semântica de tableaux, conhecido também por Beth Tableaux ou Hintikka Tableaux está intimamente ligada e foi intensamente estudada para casos da lógica clássica de primeira ordem. Seu principal objetivo é “quebrar” fórmulas complexas em outras de menor tamanho, para que possa ser analizada de uma forma mais simples.

Mais especificamente, o sistema de tableaux é um método de prova por refutação, onde prova-se um teorema pelo insucesso na tentativa de construção sistemática de um modelo para sua negação. Originalmente, o método prova através da impossibilidade de satisfação da negação de uma determinada fórmula.

3.3. Indução Matemática

O Princípio de Indução Matemárica é um método de prova fundamental de demonstração na Matemárica. Basicamente, pode-se definir o método da seguinte forma:

(i) P(1) é verdadeira; e

(ii) qualquer que seja n ε IN, sempre que P(n) é verdadeira, segue que P(n + 1) é verdadeira. Então, P(n) é verdadeira para todo n ε IN.

Ou seja, dado um valor inicial “n” que pertence ao conjunto em questão, se provamos que o termo seguinte, ou seja, “n + 1”, pertence ao mesmo conjunto, então qualquer valor pode ser obtido através do mesmo processo. A clássica analogia ligada ao teorema é do efeito dominó: se você possuir uma longa fila de peças de dominó igualmente espaçadas, e tiver certeza de que a primeira peça, quando derrubada, derruba a peça subsequente a ela, então todas as peças cairão.

Porém, um erro relativamento comum associado às leis citadas acima é de que precisamos ter provado que um termo “n” qualquer pertence a um conjunto P para que exista um elemento P(n+1), mas esta afirmação não é válida. Na verdade, tem-se que que o Teorema não exige que assumamos P(n) verdadeira para todo n ε IN, ou para qualquer n em geral. O que a hipótese (ii) exige é que sempre que n pertencer à categoria P(n) Verdadeira, então n + 1 também pertença, não exigindo qualquer afirmação acerca de P(n) ser falso.

Outro importante fator a considerar é de que indução matemática difere grandemente da indução empírica das ciências naturais, pois nestas é comum enunciar leis gerais após a observação de um número finito de experimentos na área em questão, e estas leis permanecem como verdade até que se prove o contrário. Já a prova por Indução Matemática estabelece determinadas sentenças sobre os naturais é sempre verdadeira.

3.4. Método Geométrico de Wu

Em geometria elementar, o clássico algoritmo de Alfred Tarski nos dá um procedimento decisivo para provar teoremas, porém este algoritmo não é eficiente o suficiente para provar teoremas não triviais. Em 1977, o matemático chinês Wu Went-Sün introduziu um método algébrico com o qual ele e seus discípulos da escola chinesa - Chou Shang-Ching, Xiao Shan-Gao e Zhang Jing-Zhong – demonstraram uma grande variedade de teoremas geométricos elementares não-triviais.

Infelizmente, o método de Wu apenas pode provar certos tipos de teoremas geométricos envolvendo incidência, congruência e paralelismo. Sendo mais específico, o método de Wu pode ser usado para resolução de um problema cuja hipótese e tese podem ser convertidas em equações polinomiais com coeficientes racionais.

3.5. Teorema de Prova Interativo

É um campo da lógica em que se preocupa com o desenvolvimento de ferramentas para obter provas formais com a interação máquina-homem. Existe um editor de prova interativa, ou alguma interface do tipo, no qual um usuário pode orientar a máquina para a busca de provas.

Mesmo que hajam problemas que possam ser provados totalmente de forma automática um usuário pode remover caminhos de pesquisa infrutíferos, fornecer lemas para que o provador possa atingir o objetivo, reduzindo a quantidade de processamento para realizar uma prova.

Neste método incluem vários provadores de teorema, dentre eles HOL4, HOL Light, Isabelle, ProofPower.

3.6. Modelo de Verificação

A verificação de modelos (“Model Checking”) é um método onde se verifica automaticamente a validade de propriedades acerca do comportamento de sistemas reativos. Esta técnica verifica propriedades de um sistema através da enumeração exaustiva de todos os estados alcançáveis. Para realizar a validação das propriedades é necessário seguir alguns passos: especificar quais são as propriedades que o sistema deverá ter para que seja considerado correto; a construção do modelo formal do sistema (o modelo deverá capturar todas as propriedades essenciais do sistema para verificar a correção do mesmo); execução do verificador de modelos, para validar as propriedades especificadas no sistema, nesse passo já se tem as propriedades e o modelo. Assim aplica-se o verificador e podemos garantir se o modelo tem ou não as propriedades desejadas. Caso todas sejam verdadeiras, então o sistema está correto. Caso não obedeça a alguma propriedade, então é gerado um contra exemplo mostrando o porquê da não verificação da propriedade. Dessa forma podemos

detectar o erro e realizar a correção, o processo é realizado até que o sistema obedeça todas as propriedades.

Atualmente o model checking é muito empregada na verificação formal de hardwares e softwares (não tão frequente devido à indecidibilidade, fazendo que a abordagem não possa ser totalmente algorítmica; tipicamente ela pode provar ou não provar uma dada propriedade), onde existem dois métodos para implementar o model checking: a abordagem lógica (como os verificadores SMV, PRISM, VIS, SVE e o pacote MDG) e a abordagem que utiliza a teoria dos autômatos (como os verificadores SPIN, COSPAN e FDR).

4. Principais Implementações

4.1. LCF

LCF (Logic for Computable Functions) é um teorema de prova automático interativo desenvolvido nas universidades de Edinburgh e Stanford por Robin Milner e outros na década de 1970. LCF introduziu o uso da linguagem de programação ML, uma linguagem funcional de propósito genérico, porém considerada impura, visto que ela permite programação imperativa, diferentemente de linguagens funcionais puras, tais como Haskell. Linguagens de programação lógica, como exemplo Prolog, frequentemente definem "o que" deve ser computado, mais do que "como" computar, como seria normal em uma linguagem imperativa. Os teoremas no sistema ML são proposições de um tipo abstrato de dados (TAD) especial de teorema. Ele assegura que teoremas sejam derivados, usando apenas as regras de interferência, dadas pelas operações de abstração de tipos.

O sistema automático de prova LCF também permite que um prova seja construída para trás (backwards), através de uma meta, conjectura a ser provada. Uma tática do LCF é uma função que reduz a meta a zero ou mais submetas. Assim que todas as submetas são solucionadas, LCF (usando escrituração complexa) constrói a prova de avanço correspondente, produzindo o teorema desejado. Alguns de seus sucessores incluem os ATPs Isabelle e HOL.

4.2. Isabelle

Isabelle é um assistente de teorema de prova genérico, projetado para interação lógica em uma variedade de teorias formais, tendo surgido coo sucessor do HOL. Atualmente proporciona procedimentos de prova eficientes para Teoria de Construção de Tipos (Constructive Type Theory - por Martin-Löf, 1984), lógicas de primeira ordem, teoria dos conjuntos de Zermelo-Fraenkel e lógica de alta ordem.

Ele permite que fórmulas matemáticas sejam expressadas em uma linguagem formal e proporciona ferramentas para provar essas fórmulas com cálculos lógicos. Também trabalha com lógicas de uma família bastante ampla. Alguns membros dessa família são construtivos, outros clássicos. Eles incluem lógicas de primeira ordem e de alta ordem. Algumas são baseadas em conjuntos, outras em tipos e funções, outras em domínios. A família está evoluindo, pois novos membros nascem, outros amadurecem e se desenvolvem e alguns desaparecem. Como isto é aplicado a um problema particular, as lógicas tornam-se cada vez mais refinadas e Isabelle conseguir lidar com isso. Contudo, as lógicas dinâmicas e de relevância podem cair fora do âmbito de Isabelle.

Suas principais aplicações são na formalização das provas matemáticas e em verificação formal, as quais incluem provas de correção de hardware ou software de computadores e provas de propriedades de linguagens e protocolos computacionais.

4.3. HOL

HOL (Higher Order Logic) é um sistema de provadores iterativos de teoremas que aplica separadamente estratégias de lógica e implementação. Este provador segue a mesma linha de provadores LCF (Logic for Computable Functions), os quais são utilizados na forma de bibliotecas

em determinadas linguagens de programação. Estas bibliotecas trazem consigo Tipos Abstratos de Dados (TAD's) de teoremas já provados, simplificando, desta forma, a criação de novos objetos deste tipo. Desta forma, pode-se criar resultados mais complexos através destas combinações de implementações já criadas e, portanto, confiáveis.

Atualmente mantêm-se três sistemas HOL com suporte ativo. São eles:

(1) HOL4: Evolução do HOL88, o qual surgiu por inspiração da implementação original HOL de Mike Gordon. HOL88 foi implementado em Common Lisp, ao contrário do HOL4, que seguiu o padrão do resto da família de usar o Standard ML. Ambas vêm com uma grande quantidade de codificações em bibliotecas para a prova de teoremas.

(2) HOL Light: Versão simplificada de HOL. Tornou-se bastante popular, devido à facilidade de uso. Esta biblioteca foi implementada originalmente em Caml Light, mas atualmente é disponibilizada a versão em Ocaml.

(3) ProofPower: Surgida com fins comerciais, esta biblioteca é destinada a prover suporte especial para trabalhar com notação Z (uma linguagem de especificação formal usada para descrever e modelar sistemas) e ajudar na área de especificação formal. Este projeto hoje é do tipo Open Source.

4.4. Gandalf

Gandalf é o nome de uma família de provadores automáticos de teoremas. Uma das características que diferencia o Gandalf, é que ele contém um grande número de estratégia diferentes de busca e é capaz de selecionar automaticamente a estratégia mais adequada para a solucionar um teorema. Começou a ser desenvolvido em 1995, por Tanel Tammet, tendo partivipado nas competições CASC (The CADE ATP System Competition – o campeonato mundial de provadores automáticos de teoremas) anuais, com resultados impressionantes:

- CASC-19 em 2003: campeão da classe SAT.
- CASC-18 em 2002: campeão da classe SAT.
- CASC-JC: melhor provador da classe SAT.
- CASC-17: campeão da classe SAT.
- CASC-15: campeão da classe MIX.
- CASC-14: campeão da classe MIX.

Gandalf é implementado em Scheme e compilado para o C usando o compilador Hobbit. No modo normal, o Gandalf tenta apenas encontrar insatisfatibilidade. Para encontrar satisfatibilidade tem que ser chamado com um "-sat flag".

Uma de suas principais estratégias é o corte em período de tempos: geralmente executa um número de buscas com diferentes estratégia uma após a outra, até que a prova seja encontrada ou o tempo acabe. Além disso, durante cada execução específica, o programa usualmente modifica sua estratégia quando o tempo se aproxima do fim. Cláusulas selecionadas de execuções mal sucedidas as vezes são usadas em execuções posteriores.

4.5. Otter

Otter (Organized theorem-proving techniques for effective research) é a quarta geração de sistemas de dedução produzidos no Argonne National Laboratory, que tem entre antecessores PG1, RG1, NIUTP, AURA e ITP. O programa começou a ser escrito (em linguagem C) em 1987, por John Slaney, Rusty Lusk e William McCune.

Seu funcionamento consiste em ler um arquivo de entrada contendo um conjunto de fórmulas e algumas informações de controle. O conjunto de fórmulas representa a teoria juntamente com a negação da conclusão (todas as provas são por contradição). Os métodos, juntamente com a prova,

se encontrada, são escritas em um arquivo de saída.

Sua lógica usa declarações na lógica de primeira ordem com igualdade. Ele aceita como entradas tanto cláusulas quanto fórmulas quantificadas. As fórmulas quantificadas são imediatamente transformadas em cláusulas. Todas as regras de inferência e algoritmos de busca do Otter operam em cláusulas.

4.6. Snark

SRI's New Automated Reasoning Kit (SNARK) é um provador automático de teoremas, para lógicas de primeira ordem com igualdade, destinado a aplicações em inteligência artificial e engenharia de software. SNARK foi desenvolvido para lidar com grandes conjuntos de afirmações.

Tem sido usado como o componente de raciocínio do sistema High Performance SRI Knowledge Base (HPKB), que deduz as respostas às perguntas com base em grandes repositórios de informação, além de ser utilizado como núcleo dedutivo do Amphion (sistema da NASA), para atender os pedidos do usuário, como por exemplo um cálculo de astronomia.

O SNARK não apresenta facilidades para a prova por indução matemática, ele possui capacidade de raciocínio abdutivo, que tem sido utilizado em aplicações em linguagem natural.

O provador é implementado na linguagem LISP.

5. Aplicações Práticas

Além das evidentes aplicações na lógica, provadores automáticos de teoremas foram e têm sido utilizados para resolver muitos outros problemas.

William McCune utilizou o provador Otter para achar conjuntos mínimos de axiomas, novos axiomas unitários e também para resolver problemas envolvendo estruturas algébricas. Fujita, Slaney e Bennett (os dois primeiros pesquisadores de ATP, e o último um matemático) também utilizaram provadores automáticos para demonstrações matemáticas em quase-grupos.

Verificação de software é outra área promissora para ATP, que vem sendo lentamente explorada. Um exemplo é o Karlsruhe Interactive Verifier. Trata-se de uma plataforma experimental para verificação interativa de programas, desenvolvida pela universidade de Kalrsuhe. Tem sido aplicada com êxito na verificação de softwares acadêmicos.

Os ATPs também tem sido utilizados em engenharia de software. Um exemplo é o Kerberos, um protocolo de autenticação desenvolvido no MIT pelo projeto Athena. É inspirado no protocolo Needham-Schroeder e tem seu funcionamento descrito em uma lógica de primeira ordem. Já o provador SETHEO é utilizado para verificar teoremas formulados em lógica BAN.

As aplicações de provadores, contudo, não se limitam ao escopo acadêmico. Verificação de hardware é um dos exemplos de sua aplicação na indústria, sendo uma das principais aplicações de ATPs atualmente.

Circuitos podem ser verificados através da comparação de sua implementação com as suas especificações formais. Este problema de comparação resolve o problema de satisfatibilidade, logo é NP-completo. Uma vez que verificação formal é computacionalmente cara, as empresas tomam uma decisão econômica entre tempo e dinheiro para atingir 100% de precisão em seus produtos. A maioria das empresas optam por uma simulação como estratégia para depuração de seus componentes. Os engenheiros podem conceber a entrada dos testes manualmente ou pseudo-aleatoriamente. No entanto, estes métodos não podem abranger todas as entradas possíveis e alguns bugs podem eventualmente ocorrer.

Intel é uma empresa que desenvolveu métodos de verificação com base em demonstrações automáticas de teoremas. A empresa começou a trabalhar em tais métodos, em julho de 1997, após a descoberta de um bug na unidade de ponto flutuante do Pentium Pro durante sua execução. Não só a Intel, bem como a AMD e outras empresas de desenvolvimento de hardware também aplicam ATP para verificar seus chips.

6. Conclusão

Percebe-se claramente a importância do uso e do constante desenvolvimento de sistemas automáticos de prova, afinal é através deles que diversas aplicações, como circuitos e sistemas de inteligência aritificial, são devidamente testados, a fim de verificar a existência de determinadas combinações que podem causar erro durante a operação do sistema. Seu desenvolvimento foi também de suma importância para a computação como um todo, pois impulsionou a criação de máquinas capazes de por em prática a teoria desenvolvida em torno da prova de teoremas que demandavam alto poder de processamento.

O estudo e uso de provadores automáticos de teoremas apresenta-se, portanto, como uma forma bastante eficiente de garantir a confiabilidade de programas e projetos que podem ser convertidos em funções lógicas, permitindo, desta forma, a criação de modelos que podem operar em situações das mais adversas com maior garantia de funcionamento.

Referências

Shang-Ching Chou. (1984) Proving Elementary Geometry Theorems Using Wu's Algorithm.Departamento de Matemática, Universidade do Texas, Estados Unidos da América.

Mike Gordon. (1996) From LCF to HOL: a short history. Universidade de Cambridge, Reino Unido.

Robin Milner & others. (1973) ML Programmin Language. Universidade de Edinburgh, Reino Unido.

Lawrence C. Paulson. Isabelle: The Next 700 Theorem Provers. Universidade de Cambridge, Reino Unido.

Bernhard Beckert, Joachim Posegga. Lean Theorem Proving: Maximal Efficiency from Minimal Means. Universidade de Karlsruhe, Departamento de Lógica, Complexidade e Dedução, Alemanha.

Tammet T. (1997) Gandalf, Journal of Automated Reasoning 18(2).

Moskewicz M., Madigan C., Zhao Y., Zhang L., Malik S. (2001) Chaff: Engineering an Efficient SAT Solver, *39th Design Automation Conference*, (Las Vegas, USA).

T. Tammet. Gandalf version c-2.5. http://deepthought.ttu.ee/it/gandalf/GandalfSystemDescription.html. Acessado em: 15 de junho de 2010.

Mark E. Stickel, Richard J. Waldinger, Vinay K. Chaudhri. A Guide to SNARK. http://www.ai.sri.com/snark/tutorial/tutorial.html. Acessado em: 13 de junho de 2010.

PUC-Rio. Verificação de Modelos. http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0115648_03_cap_02.pdf. Acessado em: 13 junho de 2010.

Doron Peled, Patrizio Pelliccione, Paola Spoletini. Model Checking. University of Warwick Coventry, United Kingdom.

Abramo Hefez. (2009) Indução Matemácia. Departamento de Matemática Aplicada, Universidade Federal Fluminense.

Letícia Carvalho Pivetta Fendt. (2005) Um Método dos Tableaux com um Refinamento para Evitar Repetições Desnecessárias de Fórmulas Universais nos Ramos das Árvores de Prova. Centro Universitário Luterano de Ji-Paraná

Dov M. Gabbay, Franz Guenthner. (2001) Handbook of philosophical logic. Holanda.

Larry Wos, Gail W. Pieper. (2000) The collected works of Larry Wos, Volumes 1-2. World Scientific Publishing Co., Estudos Unidos da América.