3
Provadores de Teoremas

3.1
Fundamentos

Provadores de teorema são programas de computador utilizados para provar teoremas. Dependendo da lógica escolhida, o problema de decidir a validade de uma conjectura pode variar do trivial até o impossível. Para o caso frequente da lógica proposicional, o problema é decidível, mas NP-completo, e apenas algoritmos de tempo exponencial são conhecidos para resolvê-lo. Para a lógica de primeira ordem, tal problema é recursivamente enumerável, isto é, dados recursos ilimitados, qualquer conjectura válida pode ser provada, mas conjecturas inválidas não podem ser sempre reconhecidas. Por este motivo, observa-se que a aplicabilidade de provadores totalmente automáticos fica restringida. Em contrapartida, aumenta-se a necessidade da intervenção humana na condução das provas, nos chamados *provadores interativos de teoremas* ou *assistentes de provas*. Isto é, o especialista do domínio do problema interage com o software na construção de provas.

Embora possa variar de ferramenta para ferramenta, provadores de teoremas compartilham um conjunto mínimo de componentes fundamentais. Inicialmente, a afirmação que se quer provar precisa ser informada ao provador. Para isso o provador precisa implementar alguma *linguagem lógica* (proposicional, primeira ordem, etc.). Normalmente, provadores fornecem uma linguagem própria, baseada na linguagem lógica que suportam, com a qual o usuário do sistema pode especificar conjecturas que se deseja provar, teorias e hipóteses.

Provadores de teoremas implementam *sistemas dedutivos* e, portanto, se baseiam nas regras de inferência e axiomas estabelecidos nesses sistemas dedutivos para conduzirem suas provas. Outro componente importante é o *método de desenvolvimento* de prova utilizado. Provadores podem adotar um método *bottom-up* (dos axiomas para o teorema), onde a busca pela prova é iniciada a partir do zero, isto é, o provador tenta aplicar os axiomas e regras disponíveis, em sequências de tentativa e erro, até que o teorema desejado seja encontrado. Dizemos, neste caso, que a busca pela prova segue um

estilo *forward*, caracterizando o sentido em que as inferências são aplicadas. O problema deste método é que o provador pode demorar para alcançar o resultado procurado, permanecendo preso a longos ciclos de tentativa e erro. Dependendo do caminho escolhido o provador pode, sequer, encontrar uma prova, mesmo que ela exista. Outro problema é que a primeira inferência a ser aplicada, raramente é conhecida de antemão. No método *top-down* (do teorema final para os axiomas básicos), por sua vez, a busca pela prova começa pela definição de uma fórmula representando a afirmação que se deseja provar. O provador avalia a fórmula e infere quais premissas sujeitas a qual regra poderiam levar a obtenção da fórmula em questão como conclusão. O processo prossegue, de maneira recursiva sobre as fórmulas das premissas encontradas, até que sobrem apenas axiomas, obtendo, neste caso, a prova da fórmula inicial. Ou seja, nesse estilo de busca por prova, as inferências são aplicadas no sentido contrário, do teorema para os axiomas, caracterizando o estilo conhecido como *backward*.

Um provador também deve definir uma *estratégia* de prova. Estratégias são procedimentos definidos pelos usuários que servem para guiar o provador durante a busca pela prova. Uma estratégia pode, por exemplo, definir regras de inferência de alto nível, compostas por conjuntos de regras fundamentais do sistema dedutivo do provador. Pode também, estabelecer uma ordem para aplicação destas regras. Estratégias podem ser definidas de forma que o provador sempre as use ou podem funcionar como heurísticas, utilizadas em determinadas ocasiões. Estratégias não aumentam a expressividade de um provador, mas podem melhorar sua performance no processo de busca por uma prova. 3 As características apresentadas acima podem ser encontradas, de alguma forma, na grande maioria dos provadores existentes atualmente. A lista de provadores é extensa e, como exemplo, podemos citar Otter (Ott), E (E), SPASS (Spa), Vampire (Vam) e Waldmeister (Wal), todos baseados na linguagem de primeira ordem. Para lógicas de mais alta ordem (*higher order logic*), temos ACL2 (Acl), Coq (Coq), HOL (Hol), Nqthm (Nqt), Isabelle (Isa), PVS (Pvs) e Simplify (Sim).

O interesse por provadores de teoremas tem sua origem nos anos 60, com as primeiras implementações de programas de computador para resolver o problema SAT (Pap94). O método de Resolução (Rob65), um sistema dedutivo altamente mecanizável e, portanto, fácil de implementar computacionalmente, criou bastante entusiasmo na comunidade e incentivou as pesquisas sobre o assunto nos anos 60 e 70. Nesta época, prova de teoremas por computador era um tema de pesquisa integrante da área de Inteligência Artificial. Nos anos 70 o entusiasmo diminuiu, com a constatação por parte dos pesquisadores da difi-

culdade de se provar teoremas interessantes automaticamente. A aplicabilidade dessas ferramentas nas áreas de verificação de hardware e software reativou o interesse por provadores de teoremas nos últimos anos. Inicialmente voltados para solucionar problemas estritamente matemáticos, hoje, provadores são utilizados em variados campos de conhecimento, como, por exemplo, Engenharia, Ciência da Computação e Ciência Social. Mackenzie apresenta uma descrição detalhada dessa evolução em (Mac95).

3.2 Provadores Orientados a Objetivos

Muitos dos provadores de teoremas citados acima, como, por exemplo, HOL e Isabelle, seguem uma forma de implementação baseado no provador LCF (*Logic for Computable Functions*), criado no início dos anos 70 por Robin Milner e outros (Gor00). Por este motivo são chamados de provadores do tipo LCF.

Em provadores LCF, teoremas são representados por um *tipo abstrato de dados* em uma *linguagem de programação funcional fortemente tipada*. O sistema de tipo da linguagem garante que teoremas são derivados usando somente regras de inferência fornecidas pelas operações do tipo abstrato. Portanto, regras são funções escritas na linguagem funcional do provador. No caso do provador LCF original e de HOL, a linguagem usada é ML (Pau91). O usuário pode escrever regras de inferência complexas a partir de regras mais primitivas existentes, escrevendo novas funções. A abordagem LCF garante que as funções construídas só executem inferências lógicas corretas. Linguagens funcionais foram inspiradas em *cálculo lambda*, cuja correspondência com prova formal de teoremas é estabelecida pelo chamado *Isomorfismo de Curry-Howard* (GLT89).

A busca por provas no estilo *forward* é a maneira padrão, do ponto de vista técnico, de se obter provas em provadores LCF. Apesar disso, esta pode não ser a maneira mais adequada para se tentar encontrar uma prova, conforme os problemas que discutimos na seção anterior. Por isso, provas no estilo *backward* são mais naturais, do ponto de vista do usuário que conduz a prova, isto é, seguir do fim para o início, a partir da conjectura que se quer provar, até obter a prova final desejada, aplicando as inferências e axiomas apropriados. Em provadores LCF este tipo de busca por prova é obtido tomando a conjectura que se quer provar como um objetivo inicial (*goal*) e, aplicando funções na linguagem funcional do provador, que quebram esse objetivo em novos objetivos (*subgoals*). Tais funções são chamadas de *táticas*. Além de quebrar objetivos em subobjetivos, táticas registram a sequência

de inferências empregada, que pode, posteriormente, ser usada para provar o objetivo inicial, na forma nativa de processamento desse tipo de provador, usando a busca *forward*.

A técnica de quebrar objetivos em subobjetivos não está restrita a provadores do tipo LCF e constitui uma forma de processamento comum para provadores de teoremas no estilo *backward*, conhecida como busca por prova orientada a objetivos (ou, usando o termo em inglês, *goal-directed proof search*).

3.3 Problemas no Uso de Provadores de Teoremas

Um dos maiores problemas no uso de provadores de teoremas para resolver problemas de domínios não matemáticos está em como estes problemas são especificados para uso pelo provador. Para teorias matemáticas, como, por exemplo, a teoria dos números, esta tarefa é mais fácil, uma vez que tais teorias já se encontram axiomatizadas. Para os problemas não matemáticos serem entendidos corretamente pelo provador, o usuário precisa descrevê-los em termos dos axiomas do domínio do problema e fornecer ao provador todas as informações necessárias para que ele possa encontrar resultados apropriados. Certas conclusões que fazemos automaticamente quando pensamos sobre o problema não são intuitivas para a máquina. Se o problema não for especificado por completo, o provador não conseguirá realizar seu objetivo. Estamos falando de uma atividade, essencialmente, de modelagem. Modelagem é abstração, é um processo cognitivo do usuário usando o provador. É um processo iterativo, que envolve sucessivas sessões de experimentação, com erros, acertos e muito aprendizado. Uma vez que o problema foi corretamente e completamente especificado, provadores se tornam ferramentas de grande utilidade. Contudo, chegar até este ponto, pode ser um processo demorado e custoso.

A linguagem de especificação, isto é, aquela oferecida pelo provador para que o usuário formule o problema que deseja avaliar precisa ser dominada pelo usuário do sistema. O problema, após modelado, deve ser traduzido para esta linguagem, entendida pelo provador. Não basta conhecer como representar o problema em linguagens matemáticas como a lógica proposicional ou de primeira ordem, mas como estas linguagens estão codificadas no provador. Muitas vezes, a sintaxe usada se parece mais com a de linguagens de programação do que com a de linguagens matemáticas propriamente ditas.

Outro ponto importante sobre a linguagem do provador, está nos recursos disponíveis na mesma para que o usuário possa especificar estratégias. Como discutimos na seção anterior, o usuário escreve estratégias para guiar o provador durante a busca por uma prova. É necessário entender como os comandos

disponíveis na linguagem interferem no processamento do provador para que se possa criar estratégias apropriadas.

Ao final do processo, quando o provador exibe a prova para o problema especificado, o usuário precisa ainda interpretar o resultado fornecido. É preciso entender como o resultado é apresentado pelo provador e traduzir de volta as informações obtidas para o domínio do problema sendo analisado. A análise do resultado do provador nos leva a outra questão: é possível confiar numa prova apresentado pelo provador? Um provador é um programa, para que seja confiável devemos ter certeza que seu sistema dedutivo está corretamente implementado em seu código. É preciso garantir que evoluções no código do provador não danifiquem sua capacidade de encontrar provas. Geuvers resume algumas medidas que podem ser tomadas para aumentar a confiança nos provadores em (Geu09). Uma destas soluções propõe que a prova encontrada pelo provador seja algum tipo de objeto independente que possa ser submetido a um verificador de prova. Dada uma representação de uma prova, verificar se trata-se realmente de uma prova é uma tarefa mais fácil do que encontrar a prova. Provadores podem automaticamente submeter provas encontradas à avaliação de *verificadores de prova*, aumentando a confiança dos usuários nos resultados apresentados. A figura 3.1 mostra um esquema gráfico representando esse modelo.

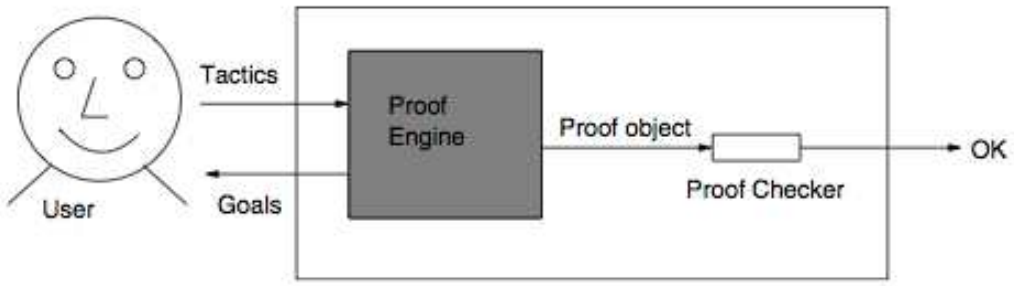

Figura 3.1: Provadores apoiados por Verificadores de Prova

A maior parte desses problemas estão relacionados com a maneira como o usuário interage com o provador, seja especificando o problema, instruindo caminhos de raciocínio ao provador ou interpretando os resultados apresentados. Muitos desses problemas poderiam ser removidos ou, pelo menos minimizados com uso de soluções apropriadas para facilitar a comunicação entre essas duas partes. Aspectos de interação humano-computador e princípios de design de interface com o usuário se tornam, portanto, elementos importantes na construção deste tipo de sistema. O próximo capítulo discute este assunto em detalhes.