Lógica Computacional e Algoritmos

Flávio L. C. de Moura
Departamento de Ciência da Computação
Universidade de Brasília[1]

14 de janeiro de 2022

[1]flaviomoura@unb.br

Introdução

Este material está sendo desenvolvido para dar suporte aos alunos de graduação da Universidade de Brasília nas disciplinas de Lógica Computacional 1 e Projeto e Análise de Algoritmos. Normalmente, o público que cursa estas disciplinas pertence aos cursos de Computação, Matemática, Engenharias e áreas afins, mas acreditamos que este material seja útil para todos que tenham interesse no tema de lógica e algoritmos. O foco deste material está na construção de provas matemáticas tanto em papel e lápis (provas informais) quanto em computador (provas formais). Construção de provas é um tema que costuma ser espinhoso para os estudantes, de forma que aqui tentaremos explorar diversas situações para facilitar este processo. Provaremos propriedades dos números naturais, propriedades de algoritmos, etc. Novas atividades são incorporadas com regularidade, normalmente a cada novo semestre letivo, e portanto este material está em constante atualização.

Ao longo deste estudo utilizaremos diferentes linguagens para a construção de provas. Iniciaremos com a linguagem da lógica proposicional (LP), que nos permitirá resolver diversos problemas interessantes. Estes problemas serão estudados também no contexto computacional. Com isto, queremos dizer que resolveremos problemas manualmente, isto é, em papel e lápis, e também no computador. Apesar da LP possuir limitações de expressividade, ela será útil para que possamos entender a dinâmica da construção de provas, mas a lógica efetivamente usada no dia a dia do matemático ou do cientista da computação é a Lógica de Primeira Ordem (LPO). A LPO nos permitirá expressar propriedades de algoritmos de forma mais natural, mas suas provas serão mais complexas. Durante esta caminhada, estudaremos um assunto fundamental que está presente em diversos contextos: *indução*. Intuitivamente, o conceito de indução é bastante simples, mas a sua aplicação em situações específicas costuma gerar muita dúvida.

A construção de provas mecânicas, ou seja, provas feitas em computador, é uma atividade que tem despertado interesse crescente nas últimas décadas em função da forma como a computação tem se infiltrado no nosso dia a dia. Mas aqui precisamos de uma pequena pausa para explicar o que queremos dizer com provas feitas por computador. Esta explicação é necessária porque existem pelo menos duas abordagens distintas no que se refere a este assunto: os provadores automáticos de teoremas por um lado, e os assistentes de prova por outro.

Um provador automático de teoremas é um programa munido de uma heurística que recebe um teorema como argumento e tenta de forma automática encontrar uma prova para o teorema dado [16, 8, 10]. Um assistente de provas por outro lado, consiste em um programa que requer a orientação do usuário para poder construir uma prova. Ou seja, o usuário vai guiando o sistema na construção de prova, enquanto o sistema verifica se cada passo dado pelo usuário está correto. São exemplos de assistentes de prova o PVS[14], o Isabelle/HOL[12], o Lean[11] e o Coq[19]. Neste material trabalharemos com o assistente de provas Coq, que é um sistema de código aberto e que pode ser instalado em sistemas Linux, MacOs e Windows, e até mesmo ser executado via browser[1].

No contexto de algoritmos e desenvolvimento de *software* é comum a utilização de testes como método de validação. Ou seja, o programa (ou *software*) é executado com diversos parâmetros distintos e se nenhum problema é encontrado, o programa é considerado bom o suficiente para ser utilizado. De fato, a primeira coisa que fazemos após implementar um algoritmo é testá-lo para diversas entradas possíveis. Caso alguma resposta esteja fora do esperado, uma revisão da implementação é feita para corrigir o erro, e então novos testes são realizados. Este processo é repetido até que o programador sinta confiança na implementação, mas depois de todos estes testes é possível dizer que o programa é correto? Certamente

não! Pensando no caso particular da implementação de um algoritmo de ordenação de inteiros, sabemos que existe uma infinidade de listas de inteiros que podem ser utilizadas nos testes, e portanto não é possível testar todas elas. Em se tratando de programas em sistemas críticos (aviação, medicina, sistemas bancários, etc), por menores que sejam as chances, podem existir entradas específicas que não foram testadas, e falhas não são toleradas em sistemas críticos principalmente. O que fazer então para garantir que o programa é 100% correto? A resposta é utilizar a lógica para **provar** a correção do programa! Uma prova matemática de uma propriedade de um programa fornece a garantia de que o programa satisfaz a propriedade provada **sempre**! Esta é a abordagem que utilizaremos aqui e que tem se mostrado cada vez mais importante para o desenvolvimento da Matemática[6, 5, 2, 3] e Computação[9, 15, 13]. Para concluir esta seção e começarmos a colocar a mão na massa, listamos a seguir três exemplos famosos de erros em sistemas computacionais:

1. **Therac-25**: Uma máquina de radioterapia controlada por computador causou a morte de pelo menos 6 pacientes entre 1985 e 1987 por overdose de radiação.

2. **Pentium FDIV**: Um erro na construção da unidade de ponto flutuante do processador Pentium da Intel causou um prejuízo de aproximadamente 500 milhões de dólares para a empresa que se viu forçada a substituir os processadores que já estavam no mercado em 1994.

3. **Ariane 5**: Um foguete que custou aproximadamente 7 bilhões de dólares para ser construído explodiu no seu primeiro voo em 1996 devido ao reuso sem verificação apropriada de partes do código do seu predecessor.

A *Lógica Computacional* (LC) tem por objetivo utilizar a lógica para raciocinar sobre Computação, ou seja, consiste na utilização da lógica para a resolução de problemas computacionais. Este conceito pode ser utilizado de diversas formas, por exemplo, é comum a associação da LC com programação em lógica. Neste caso, a lógica é utilizada como uma linguagem de programação [18]. Uma outra abordagem possível é a simples mecanização do raciocínio lógico, de forma a permitir a resolução de exercícios de lógica no computador, ao invés da resolução usual em papel e lápis [7]. A abordagem que utilizaremos difere das anteriores, mas possui um pouco de cada uma delas como veremos a seguir.

Para explicar a nossa abordagem, suponha que você tenha um grande banco de dados com informações de uma determinada população, e que por alguma razão precise ordenar estas informações por idade em determinado momento; em outro, a ordenação que precisa ser feita é por nome ou outro critério qualquer. O que você faz? Uma alternativa é utilizar alguma implementação já feita e resolver o problema. Uma pergunta que pode ser feita é: será que a implementação utilizada gera a resposta correta? Outra alternativa seria construir/implementar um algoritmo de ordenação, mas a questão sobre a correção ainda continua válida: a implementação construída é correta? A abordagem que utilizamos neste curso fornece as ferramentas necessárias para responder a estas perguntas. Em particular, estudaremos sistemas dedutivos que nos permitirão provar propriedades de programas[4].

Já deixamos claro que vamos **provar** muita coisa aqui. Mas o que é uma prova? Uma resposta possível, "é um argumento feito para convencer alguém"[17]. O problema deste argumento é que pessoas diferentes podem ter compreensões distintas sobre o argumento, de forma que o argumento seja uma prova para uma, mas não para a outra... estranho, não? Uma definição geral e abstrata para a noção de prova não é uma tarefa fácil, mas forneceremos uma definição precisa em um contexto mais restrito, a saber, o da lógica simbólica.

Agradecimentos Este material foi escrito com o apoio do programa Aprendizagem para o Terceiro Milênio (A3M) coordenado pelo CEAD/UnB, que viabilizou o trabalho do estudante Rafael Monteiro Rodrigues na elaboração das soluções das atividades propostas. O estudante Gabriel Silva também fez contribuições importantes na elaboração das atividades, e em particular, na formalização do algoritmo *mergesort*.

Referências Bibliográficas

[1] Emilio Jesús Gallego Arias, Benoît Pin, and Pierre Jouvelot. jsCoq: Towards Hybrid Theorem Proving Interfaces. *Electronic Proceedings in Theoretical Computer Science*, 239:15–27, January 2017.

[2] Jeremy Avigad, Kevin Donnelly, David Gray, and Paul Raff. A formally verified proof of the prime number theorem. *ACM Transactions on Computational Logic*, 9(1):2–es, December 2007.

[3] Jeremy Avigad and John Harrison. Formally verified mathematics. *Communications of the ACM*, 57(4):66–75, April 2014.

[4] M. Ayala-Rincón and F. L. C. de Moura. *Applied Logic for Computer Scientists - Computational Deduction and Formal Proofs*. UTCS. Springer, 2017.

[5] G. Gonthier. A computer-checked proof of the Four Colour Theorem. Technical report, Microsoft Research Cambridge, 2008.

[6] T. Hales, M. Adams, G. Bauer, D. Tat Dang, J. Harrison, T. Le Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. Tat Nguyen, T. Quang Nguyen, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, A. Hoai Thi Ta, T. N. Tran, D. Thi Trieu, J. Urban, K. Khac Vu, and R. Zumkeller. A formal proof of the Kepler conjecture. *ArXiv e-prints*, January 2015.

[7] Cezary Kaliszyk. Web Interfaces for Proof Assistants. *Electronic Notes in Theoretical Computer Science*, 174(2):49–61, 2007.

[8] Cezary Kaliszyk, Stephan Schulz, Josef Urban, and Jiří Vyskočil. System Description: E.T. 0.1. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, volume 9195, pages 389–398. Springer International Publishing, Cham, 2015.

[9] Xavier Leroy. Formal Verification of a Realistic Compiler. *Communications of the ACM*, 52(7):107, 2009.

[10] W. McCune. Prover9 and mace4. http://www.cs.unm.edu/~mccune/prover9/, 2005.

[11] Leonardo de Moura and Sebastian Ullrich. The Lean 4 Theorem Prover and Programming Language. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, Lecture Notes in Computer Science, pages 625–635, Cham, 2021. Springer International Publishing.

[12] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lncs*. Springer, 2002.

[13] R. B. Nogueira, A. C. A. Nascimento, F. L. C. de Moura, and M. Ayala-Rincón. Formalization of Security Proofs Using PVS in the Dolev-Yao Model. In *Booklet Proc. Computability in Europe - CiE*, 2010.

[14] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *CADE*, volume 607 of *Lnai*, pages 748–752. sv, 1992.

[15] Lawrence C. Paulson. A Mechanised Proof of Gödel's Incompleteness Theorems Using Nominal Isabelle. *J Autom Reasoning*, 55(1):1–37, 2015.

[16] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15(2-3):91–110, 2002.

[17] Raymond Smullyan. *Logical Labyrinths*. AK Peters, 2009.

[18] Leon Sterling and Ehud Y Shapiro. *The Art of Prolog: Advanced Programming Techniques*. MIT press, 1994.

[19] The Coq Development Team. The Coq Proof Assistant. Zenodo, October 2021.