Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação 2018/2

Verificação Formal de Programas

Flávio Leonardo Cavalcanti de Moura*

Apresentação

Um programa de computador é um conjunto de instruções que direciona o processador a realizar determinadas operações. Estes programas em geral são feitos para resolver problemas específicos como, por exemplo, ordenar uma lista de números, realizar operações bancárias, medir a frequência cardíaca de um paciente, controlar um avião, etc. No entanto, a prática mostra que erros na construção de programas são frequentes. Alguns exemplos famosos são:

1. **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.
2. **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.
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 construção de programas corretos é fundamental na sociedade informatizada em que vivemos, e a utilização de métodos formais é cada vez mais comum:

1. A Intel e AMD utilizam assistentes de provas na verificação de processadores.
2. A Microsoft utiliza métodos formais na verificação de programas e drivers.
3. CompCert: Compilador C verificado em Coq.
4. A AirBus e a NASA utilizam assistentes de provas na verificação de programas de aviação.
5. A Toyota utiliza métodos formais na verificação de sistemas híbridos de controle.
6. A linha 14 do metrô de Paris é totalmente controlada por um programa de computador verificado formalmente.

Apesar da utilização cada vez mais frequente de métodos formais na construção de programas, esta não é uma tarefa fácil. Como evitar erros na construção de programas? Uma abordagem possível é construir o programa e depois realizar testes para verificar se os resultados são os esperados. Assim, se um resultado está incorreto, o programador pode fazer uma nova inspeção do código corrigindo os erros, e em seguida repetir este processo enquanto for possível. Uma vez que todos os testes realizados pelo programador tenham retornado as respostas esperadas, podemos dizer que o referido programa é correto? Certamente não porque podem ainda existir entradas que falham e não foram consideradas pelo programador. Como resolver este problema? Podemos testar todas as entradas possíveis? Se o número de entradas possíveis for finito

*contato@flaviomoura.mat.br

certamente podemos testá-las todas, e poderemos responder com certeza (após um tempo finito) se o citado programa é correto ou não. Mas, e se o número de possíveis entradas for infinito? Existem técnicas cada vez mais sofisticadas para a verificação de programas uma vez que a inspeção manual dos mesmos está suscetível a erros, além de ser demorada e cara. Uma outra possibilidade consiste em utilizar a lógica para construir o programa juntamente com sua especificação, e provar que o programa satisfaz esta especificação [5]. Isto significa expressar a correção do programa por meio de teoremas que, uma vez provados, garantem a correção do programa. Nesta abordagem utilizam-se os chamados *assistentes de prova* [6, 3, 4] e/ou os provadores automáticos para especificar e formalizar programas. A desvantagem do primeiro método (testes exaustivos) é evidente (correção parcial), mas então por que a construção de programas via assistentes de prova não é ainda tão popular? Uma razão para isto está na dificuldade em se obter mão de obra qualificada com sólidos conhecimentos em Computação e Matemática. Além disto, o tempo necessário para se completar a tarefa pode demandar meses ou anos para ser concluída. Por exemplo, a projeção inicial dos proponentes do projeto Flyspeck (formalização da conjectura de Kepler) foi de algo em torno de 20 anos!

Neste curso estudaremos aspectos teóricos e práticos no que se refere a verificação formal de programas. A abordagem a ser utilizada é focada na utilização da lógica para a construção de programas via assistentes de prova. O assistente de prova que será utilizado é o Coq (`http://coq.inria.fr`), um sistema que foi implementado na linguagem funcional Ocaml (`https://ocaml.org`). O paradigma funcional de programação é baseado em um formalismo conhecido como cálculo-λ. Este formalismo, como veremos, é capaz de expressar qualquer função computável, inclusive as funções que não terminam, isto é, que entram em *loop*. As funções que nunca terminam podem ser excluídas do cálculo-λ por meio dos sistemas de tipos, mas este tema ficará para um outro curso.

Neste curso utilizaremos o Coq [6, 2] para construirmos programas corretos via o mecanismo de extração de código (em Ocaml, Haskell ou Scheme) deste assistente de prova. O Coq é uma ferramenta robusta que tem sido utilizada com sucesso na formalização de resultados importantes, como por exemplo:

1. Teorema das quatro cores: Formalizado por George Gonthier em Coq, 2005.
2. Teorema dos números primos: Formalizado por Avigad et. al. em Isabelle, 2004.
3. Teorema de Feit-Thompson: Formalizado por George Gonthier em Coq, 2012.
4. Conjectura de Kepler (projeto Flyspeck, 2014), dentre outros.

Público alvo

Esta disciplina tem como foco alunos dos cursos das áreas de Computação e Matemática (incluindo as engenharias) que têm interesse na verificação formal de programas de computador.

Avaliação

O processo de avaliação será feito via o desenvolvimento de atividades diretamente no assistente de provas Coq.

Informações adicionais

- Dias, horários e local das aulas:
  - Terças e Quintas (20h50 - 22h40)
- Para a realização das atividades em Coq é necessário que o aluno tenha o Coq (versão 8.8.1) já instalado no seu computador pessoal desde o primeiro dia de aula.
- Os alunos que não tiverem com um computador pessoal durante a aula poderão desenvolver as atividades em grupo, e posteriormente completá-las em casa ou no LINF.

- A bibliografia principal é [2]. Uma versão em Francês está disponível gratuitamente [1]. Adicionalmente, um arquivo **pdf** contendo exercícios e a teoria abordada em aula será disponibilizado para os alunos na página `http://flaviomoura.mat.br`.

Bibliografia

[1] Y. Bertot and P. Castéran. Le coq'art. http://www.labri.fr/perso/casteran/CoqArt/coqartF.pdf, 2015.

[2] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. EATCS - Texts in Theoretical Computer Science. Springer, 2004.

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

[4] 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.

[5] Benjamin C. Pierce, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Catvalin Hriatcu, Vilhelm Sjoberg, and Brent Yorgey. *Software Foundations*. Electronic textbook, 2014. `https://softwarefoundations.cis.upenn.edu/`.

[6] The Coq Development Team. The coq proof assistant, version 8.7.2, February 2018.