IA159 Formal Verification Methods

Introduction

Jan Strejček

Department of Computer Science
Faculty of Informatics
Masaryk University

Agenda

Agenda

- basic information about the course
- quick overview
- motivation

What does “Formal Verification Methods” mean?

formal methods are a collection of notations and techniques for describing and analyzing systems. Methods are **formal** in the sense that they are based on some mathematical theories, such as logic, automata or graph theory. [Pel01]

verification is the process of applying a manual or an automatic technique that is supposed to establish whether the code either satisfies a given property or behaves in accordance with some higher-level description of it. [Pel01]

formal verificatin methods are techniques (usually based on mathematical theories) for analysing systems with the aim to improve their quality and reliability.

Focus of the course

- The course is focused on theoretical and algorithmic bases of verification methods.
- The software engineering aspects of verification methods are beyond the scope of this course.

Literature

- There is no single reading material covering the course.
- Two main sources:
  - *D. A. Peled: Software Reliability Methods, Springer, 2001.*
  - *E. M. Clarke, O. Grumberg, and D. A. Peled: Model Checking, MIT, 1999.*
- Other sources (mainly recent journal or conference papers) will be referred.

Connections to other courses

The course assumes familiarity with the following notions:

- IB005 Formal Languages and Automata I (aka FJA I)
  - pushdown automata
- IA006 Selected topics on automata theory (aka FJA II)
  - infinite words, Büchi automata, bisimulation equivalence
- IA040 Modal and Temporal Logics for Processes
  - temporal logics, mainly LTL
- IV113 Introduction to Validation and Verification
  - automata based LTL model checking

Connections to other courses

Outher relevant courses:

- MA015 Graph Algorithms
- IV010 Communication and Parallelism
- IB002 Design of Algorithms I
- IV022 Design and Verification of Algorithms
- PA008 Compiler Construction

Connections to other courses

Courses following (is some sense) our course:

- IV115 Parallel and Distributed Laboratory Seminar
- IV074 Laboratory for Parallel and Distributed Systems
- IA072 Seminar on Concurrency

Examination

- There will be an oral exam at the end.
- No intrasemestral tests, no written exams, no homeworks.

Overview of verification methods

Verification methods

- testing
- deductive verification (with use of theorem provers)
- equivalence checking
- reachability and model checking
- static analysis and abstract interpretation
- combined methods

Testing

- simple, feasible, very good cost/performance ratio
- very effective in early stages of debugging process
- applicable directly to real systems
- cannot guarantee that there are no errors
- in practice: standard technique for enhancing the quality of systems, wide tool support

Deductive verification

- applicable to models of real systems
- needs a huge effort of an expert on both deductive verification and systems under verification
- can guarantee that (a model of) a real system satisfies a given property
- in practice: used rarely (e.g. partial correctness of FPU in AMD processors)

Equivalence checking

- applicable to models of real systems
- needs a detailed formal specification of a system under verification
- there are no algorithms for reasonable equivalences and infinite-state systems
- in practice: some specific applications (e.g. equivalence of different levels of hardware design)

Reachability and model checking

- applicable to (usually finite-state) models of real systems
- needs formal specification of a system under verification
- fully automatic, but feasible only for relatively small finite-state systems
- in practice: a standard technique for verification of simple hardware designs, used also for verification of small systems (e.g. communication protocols)

Static analysis and abstract interpretation

- applicable directly to source code of real systems, feasible
- can verify only a specific class of properties (including many interesting properties)
- may produce false alarms (the number of false alarms grows with the ability to find real bugs)
- automatic (verification of some properties may require provision of a suitable abstraction)
- in practice: some static analysis is performed by almost every compiler, there are very efficient tools (e.g. Coverity, Stanse) able to work with big pieces of real software, for example a linux kernel

Combined methods

- popular combinations:
  - abstraction + model checking
  - model checking + counter-example guided abstraction refinement (CEGAR)
  - abstract interpretation + CEGAR
  - testing + model checking
  - testing + symbolic execution (a case of abstract interpretation)
- the aim is to develop methods which are applicable directly to (source code of) real systems and (more or less) automatic
- may be incomplete and/or produce false alarms
- in practice: already has some specific applications, e.g. verification of Windows drivers by Static Driver Verifier
- definitely the most promising approach

Actual topics

- formal aspects of testing: coverage criteria, model-based testing, whitebox fuzz testing
- deductive software verification: verification of flowcharts, axiomatic program verification
- theorem prover ACL2 (with a demo)
- LTL $\rightarrow$ BA via alternating 1-weak BA
- partial order reduction
- LTL model checking of pushdown systems
- abstraction and CEGAR
- abstract interpretation (???)

Motivation

- Formal verification is used in Microsoft, Intel, AMD,...
- Formal verification is usually a supplementary method, the main methods are testing or simulation.
- In development of execution cluster of Core i7, formal verification has been used as a primary validation vehicle (simulation has been dropped)
- only 3 bugs escaped to silicon (2 other bugs were detected during the pre-silicon stage by full chip testing)
- this number is usually about 40
- the previous minimum is 11
- More information in Kaivola et al: *Replacing Testing with Formal Verification in Intel Core i7 Processor execution Engine Validation*, CAV 2009, LNCS 5643, Springer, 2009.

Coming next week

Software testing

- Do my tests cover the whole code?
- What does it mean “to cover the whole code”?
- What can I do if I do not know the code?