Introduction to Formal Methods

Các Phương Pháp Hình Thức Cho Phát Triển Phần Mềm
Outline

- Introduction
- Formal Specification
- Formal Verification
- Model Checking
- Theorem Proving
Introduction

- Good papers to begin with them:
Teaching to unsuspecting youngsters the effective use of *formal methods* is one of the joys of life because it is so extremely rewarding.

“The *Cruelty of Really Teaching Computing Science*” is a 1988 paper by E. W. Dijkstra,
A more mathematical approach is inevitable. Professional software development—not the everyday brand practiced by the public at large—will become more like a true engineering discipline, applying mathematical techniques. I don't know how long this evolution will take, but it will happen. The basic theory is there, but much work remains to make it widely applicable.

(Bertrand Meyer, a pioneer of object technology)
Software engineers want to be real engineers. Real engineers use mathematics. Formal methods are the mathematics of software engineering. Therefore, software engineers should use formal methods.

(Mike Holloway, NASA)
Introduction

- Major goal of software engineers
  - Develop reliable systems
- Formal Methods
  - Mathematical languages, techniques and tools
  - Used to specify and verify systems
  - Goal: Help engineers construct more reliable systems
- A mean to examine the entire state space of a design (whether hardware or software)
  - Establish a correctness or safety property that is true for all possible inputs
Introduction

- Past years of the formal methods
  - Obscure notation
  - Non-scalable techniques
  - Inadequate tool support
  - Hard to use tools
  - Very few case studies
  - Not convincing for practitioners
Introduction

- Nowadays
  - Trying to find more rigorous notations
  - Model checking and theorem proving complement simulation in Hardware industry
  - More industrial sized case studies
  - Researchers try to gaining benefits of using formal methods
  - ...

Introduction

- Formal methods can be applied at various points through the development process
  - Specification
  - Verification

- **Specification**: Give a description of the system to be developed, and its properties

- **Verification**: Prove or disprove the correctness of a system with respect to the formal specification or property
Specification

- Using a language with a mathematically defined syntax and semantics

System properties

- Functional behavior
- Timing behavior
- Performance characteristics
- Internal structure
Specification

- Specification has been most successful for behavioral properties
- A trend is to integrate different specification languages
  - Each enable to handle a different aspect of a system
- Some other non-behavioral aspects of a system
  - Performance
  - Real-time constraints
  - Security policies
  - Architectural design
Specification

- Formal methods for specification of the sequential systems
  - Z (Spivey 1988)
  - Constructive Z (Mirian 1997)
  - VDM (Jones 1986)
  - Larch (Guttag & Horning 1993)

- **States** are described in rich math structures (set, relation, function)

- **Transition** are described in terms of pre- and post- conditions
Specification

- Formal methods for specification of the concurrent systems
  - CSP (Hoare 1985)
  - CCS (Milner 1980)
  - Statecharts (Harel 1987)
  - Temporal Logic (Pnueli 1981)
  - I/O Automata (Lynch and Tuttle 1987)
- **States** range over simple domains, like integers
- **Behavior** is defined in terms of sequences, trees, partial orders of events
Specification

- Formal methods for handling both rich state space and complexity due to concurrency
  - RAISE (Nielsen 1989)
  - LOTOS (ISO 1987)
Case Studies: CICS

- The CICS project
- **CICS**: Customer Information Control System
  - The on-line transaction processing system of choice for large IBM installations
- In the 1980s Oxford Univ. and IBM Hursley Labs formalized parts of CICS with Z
- There was an overall improvement in the quality of the product
- It is estimated that it reduced 9% of the total development cost
Case Studies: CICS

- This work won the Queen’s Award for Technological
  - The highest honor that can be bestowed on a UK company.
Case Studies: CUTE

- CUTE: A Concolic Unit Testing Engine for C
- Developed by a team managed by Gul Agha – 2005

Concolic testing
- use the symbolic execution to generate inputs that direct a program to alternate paths
- use the concrete execution to guide the symbolic execution along a concrete path
Case Studies: CUTE

- CUTE was used to automatically test SGLIB, a popular C data structure library used in a commercial tool.
- CUTE took less than 2 seconds to find two previously unknown errors:
  - a segmentation fault
  - an infinite loop
- The homepage of CUTE:
  - http://osl.cs.uiuc.edu/~ksen/cute/
Case Studies: Intel’s Successes

http://www.cse.ogi.edu/S3S/JohnHarrison.pdf

- Intel uses formal verification quite extensively
  - Verification of Intel Pentium 4 floating-point unit with a mixture of STE and theorem proving
  - Verification of bus protocols using pure temporal logic model checking
  - Verification of microcode and software for many Intel Itanium floating-point operations, using pure theorem proving
- FV found many high-quality bugs in P4 and verified “20%” of design
- FV is now standard practice in the floating-point domain
Case Studies: NASA SATS

- Small Aircraft Transportation System (SATS)

- Use of a software system that will sequence aircraft into the SATS airspace in the absence of an airport controller

- There are serious safety issues associated with these software systems and their underlying key algorithms

http://sats.nasa.gov/
Case Studies: NASA SATS

- The criticality of such software systems necessitates that strong guarantees of the safety be developed for them.
- Under the SATS program NASA Langley researchers are currently investigating rigorous verification of these software systems using formal methods:
  - Modeling and Verification of Air Traffic
  - Conflict Detection and Alerting
  - …
Verification

- Two well established approaches to verification
  - Model Checking
  - Theorem Proving

- Model checking
  - Build a finite model of system and perform an exhaustive search

- Theorem Proving
  - Mechanization of a *logical* proof
Model Checking

- The technical challenge is to devise an algorithm for handling large spaces
- Rebeca uses compositional verification
Model Checking

- There are two general approaches in model checking
  1. Temporal Model Checking
  2. Model checking with a automaton spec

- The difference is between the specification
  - First one : Temporal Logic
  - Second one : Automaton
Model Checking

- Model checking is completely automatic
- It produces counter examples
  - The counter example usually represents subtle error in design
- The main disadvantage: state explosion problem!
Model Checking

- Several approaches for facing the state explosion
  - Ordered binary decision diagrams (BDD) – McMillan
  - Partial Order – Peled
  - Localization reduction – Kurshan
  - Semantic minimization – Elseaidy

- Checking large systems by using appropriate abstraction techniques
  - Burch et al. $10^{120}$ states!
Theorem Proving

- Both the system and its desired properties are expressed in some mathematical logic
- Theorem proving is the process of finding a proof from the axioms of the system
- It can be roughly classified
  - Highly automated programs
  - Interactive systems with special purpose capabilities
- In contrast to model checking, it can deal with infinite space
- Relies on techniques like reduction
Question?