Aula 1

1 Disciplina

Verificação Formal de Software

Software and cathedrals are much the same. First we build them, then we pray.

Verificação Formal de Software

Reliability Correctness Safety Robustness

Formal Methods

Formal methods are methods based on mathematical techniques for the rigorous specification (modelling), development (synthesis) and verification (analysis) of software and hardware systems, with the aim of achieving higher levels of quality.

Verificação Formal de Software

URL:`www.dcc.fc.up.pt/nam/web/Teaching/vfs20/index.html`

Método de avaliação

1. realização de 4 trabalhos práticos (40) : 2 escritos e 2 práticos
2. exame (60)

2 Programa

Programa da disciplina

1. Breve introdução aos métodos formais e a técnicas de verificação formal.
2. Verificação de Dedutiva de Programas
3. Verificação por model checking de sistemas reativos

Verificação de Dedutiva de Programas

1. Cálculos de correcção (Lógica de Hoare)
2. Pré-condições mais fracas e algoritmos de geração de condições de verificação.
3. Geração de obrigações de prova
4. Ferramentas para a especificação, verificação e certificação de programas: Dafny
5. Correção de programas imperativos e orientados a objectos com Dafny

Verificação por model checking de sistemas reativos

1. Modelação de sistemas paralelos: sistemas transição
2. Paralelismo e comunicação
3. Propriedades temporais lineares: segurança, liveness e fairness
4. Lógicas temporais: linear (LTL) e ramificada (CTL e CTL*).
5. Modelação e especificação usando um model checker (SPIN)
6. Algoritmos de model checking para LTL e CTL
7. Model checking simbólico: BDDs e OBDDs.
8. Técnicas de implementação de model checking.
9. Algoritmos de decisão baseados em autómatos

3 Bibliografia

Bibliografia

Livros recomendados

- Logic in Computer Science, [HR04] (Cap. 3, 4, 6)
- **Rigorous Software Development**, [AFPMdS11]
- **Principles of Model Checking** [BKL08]
- Principles of the Spin Model Checker[BA08]

4 Formal Methods

Formal methods in the design of Information and Computer Systems (ICS)

1. Formal specifications: languages Z, VDM, B, JML
2. Ensure that the specifications satisfy certain properties
3. Derive implementations from the specifications (synthesis)
4. Verify the implementations w.r.t the specifications

Critical systems and software errors

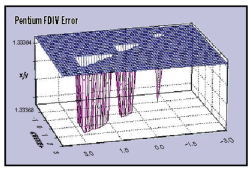
- Arianne-5, 1996
- Marte "Path Finder"
- Airbuses

Control systems:

- Nuclear Plants
- Traffic Conttollers
- Medical Tools
- etc.

In general, for each 1000 lines of code there is an error

Ups!



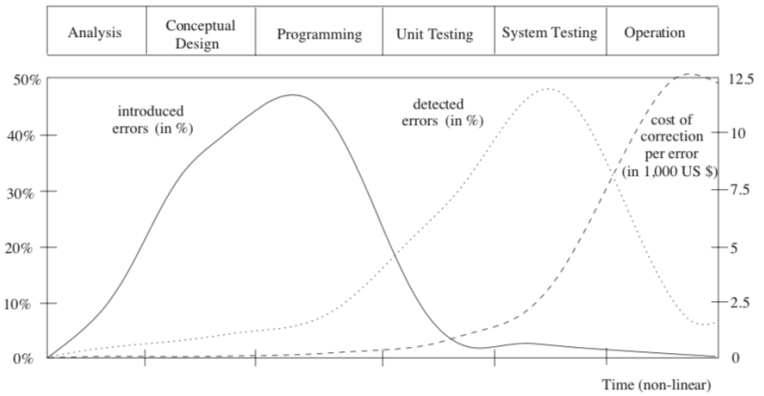
Intel Pentium bug caused loss of reputation and money.

Ariane 5 crashed within a few minutes after launch

Software bug caused Toyota to recall 1.2M Prius cars

Software race condition caused northeast blackout of 2003

Software cycle of life, errors and costs



Formal Methods Tools

- How to ensure at the specification level the desired behaviour: *model validation problem*
- How to ensure that the implementation has the same behaviour as the specification: *formal relation between the specification and the implementation problem*
- Study of the specification: animation, transformation or proving of properties

- Implementations can be:
  - derived from specifications;
  - ensure their correctness: by construction (correct-by-construction), verification, by formal proofs

Central notions and techniques/verification tools

- The operational essence of the modelleded systems is captured by a transition system
  - different mechanisms imply different interpretations of the notions of: states, transitions, state transformations
- The behavioural essence of the modelled systems is captured by some program logic

Main approaches for Specification

- The behaviour of the system is described by:
  - operations, available mechanisms, or actions that can be performed *Specification languages of this type are referred as state-based or model-based*
  - manipulated data; how they evolve, or the way in which they are related. *This class of specifications includes algebraic specifications or axiomatic specifications*

State-based Specifications

- Capacity of describe the notion of state
- Describe how the system operations modify the state
- Formalization based on:
  - discrete mathematics;
  - set theory;
  - category theory;
  - logic

State-based Specifications

- Abstract State Machines: sistem described by states and by a finite set of transition rules between states
- Category and set theories: states describedt by mathematical structures (sets, relations or functions); transitions expressed as invariants, preconditions and post-conditions.
- Automata based models: to model systems with a concurrent behaviour; to define how the system reacts to events; adequated for reactive systems, concurrent or communication protocols.
- Modelling languages for Real-time systems (*cyberphysical*): capacity of modelling physical concepts such as time, temperature, slope, etc.; (e.g. synchronous concurrent systems)

Examples

- Abstract Machines:
  - ASM_Gopher was te base of a formalization of the Java language;
  - B method which methodology is similar to object-oriented modelling. Originate several implementations: Atelier B, BRILLANT, ProB, Rodin, etc.
- Category and set theory:
  - Formal methods Z and VDM are based in predicate logic and set theory. Were base of other systems such as: RAISE, Alloy (extends Z to allow partial analysis)
  - Specware, Charity are formalisms based on Category theory.

Examples

- Real-time modelling languages:
  - Lustre is a synchronous dataflow language and SCADE is a modelling graphical environment (based on Lustre) that allow to express synchronous concurrency based on dataflow.
  - Uppaal and Kronos are model checkers based on timed automata
  - Hytech and KeYmaera, ares based on hybrid automata in order to model dynamic systems with an interactive behaviour(e.g.*transportation systems*)

Algebraic Specifications

- Collections of declarations, function signatures, and axioms that declare the behaviour of each function symbol
- Examples of tools and languages:
  - CASL, OBJ, Clear, Larch, ACT-ONE
  - LOTOS - based on CCS (Calculus of Communicating Systems), and allows to specify concurrent systems

Declarative Modelling

- Logic-based languages, Functional languages, and rewriting languages
  - Logic-based languages: Prolog based on first-order logic
  - Functional languages based on λ-calculus: Scheme, SML, Haskell and OCaml; proof assistants such as ACL2, Coq, PVS, HOL, Isabelle e Agda, are based on typed variants and extensions of λ-calculus. By the Curry-Howard isomorphism the type is a formula and the λ-term a proof.
  - Rewritting systems such as ELAN or SPIKE: the behaviour of functional symbols is described by equational systems and the execution is based of the notion of reduction (as in the λ-calculus).

5 Verification

Verification of Information and Computer Systems (ICS)

1. asynchronous/synchronous
2. analogic/digital hardware
3. mono/multi processors
4. languages: imperative, functional, logic, object oriented
5. sequential or multi-*threaded*
6. Convencional operating systems or real-time
7. Embedded systems
8. Distributed systems

Types of ICS

1. Transformational: reads input data and produces an output; should terminate. Ex: compiler
2. Interactive: interact with the user through events; do not terminate. Ex: operating system
3. Reactive: the interaction is determined by the environment. Ex: flights database access; train controllers

Formal Verification

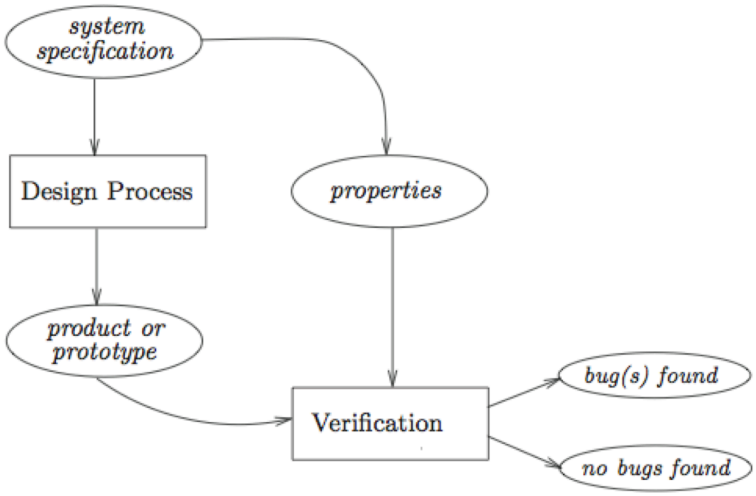
Modelling systems platform

Specification Language Linguagem de especificação] to describe the properties one wants to verify

Verification Method proofs can be made

- "by hand, in paper"
- with some automation
- using automatic or interactive computational tools

Formal Verification



Verification Approaches

Deduction systems vs. Models *DS*: the system is described by a set of formulas Γ, the specification is a formula φ, and we want

$$\Gamma \vdash \varphi$$

(in general semi-automatic)

M: The system is described by a model é descrito $\mathcal{M}$, the specification is a formula φ, and we want

$$\mathcal{M} \models \varphi$$

. (in general automatic for finite models)

Automation Degree automatic/interactive

Complete vs. Properties A specification describes one property or the behaviour of the whole system.

Domain Hardware/Software; sequential/functional or concurrent/reactive

Static vs Dynamic The verification is performed during runtime or before execution.

Verification Methods

Program verification

Interactive, Deduction systems, Property verification; Terminating

Model checking

Automatic, Model based, Verification of properties, concurrent and reactive systems, dynamic

But the approaches are not strict and techniques by be mixed . For example for embedded system or *proof carrying code* systems.

Proof Methods

There are 3 categories:

- Proofs made by hand and can be informally described
- Tools that allow the formal definition of the proofs.
- Computer assisted proofs

Logics:

- propositional logic, first-order logic, high-order logic
- classic logic versus intuicionistic logic
- modal and temporal logics

Proof Tools

- Automatic proof tools: use a decidable logic fragment
  - ELAN: first-order rewrite
  - ACS2: first-order logic
  - SMT Solvers (Satisfability Module Theory): Yices, CVC3, Z3, Alt-Ergo, Simplify: integers, reais, "arrays", etc.
  - Allow reason about infinite sets
- Interactive proof tools: allow more expressive logics, potentially nondecidable. Coq, Matita, HOL, etc
  - Combine two capacities: proof check and assisted proof construction
  - Proofs are build interactively using tactics: case, elim, change, rewrite, simpl, discriminate, injection, induction.

6 Bibliografia

Referências

[AFPMdS11] José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simão Melo de Sousa. *Rigorous Software Development: An Introduction to Program Verification*. Springer, 2011.

[BA08] Mordechai Ben-Ari. *Principles of the Spin Model Checker*. Springer, 2008.

[BKL08] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of Model Checking*. MIT Press, 2008.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. CUP, 2004.