Verificação de Programas

Nelma Moreira

Departamento de Ciência de Computadores da FCUP

Verificação de Programas/Program Verification
Aula/Lecture 1

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

Reliability
Correctness
Safety
Robustness

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.

URL:`www.dcc.fc.up.pt/~nam/Teaching/vp2024/`

Assignments

1. 3 practical assignments (40%)
2. final exam (60%)

1. Brief introduction to formal methods and formal verification
2. Deductive verification
3. Automatic theorem provers: SAT and SMT solvers
4. Brief introduction to interactive theorem provers

1. Partial and total correctness calculus (Hoare logics).
2. Weak-preconditions and verification condition generators.
3. Tools for the specification, verification and certification programs: Dafny
4. Correction of imperative and object oriented programs with Dafny

Automatic and interactive theorem provers

1. Proposition satisfiability problem (SAT).
2. Decidable theories: integers, reals, *arrays*, pointers etc.
3. SMT, Satisfiability modulo theories: algorithms and combination.
4. SMT tools (Z3)
5. Introduction to interactive theorem provers (Coq)

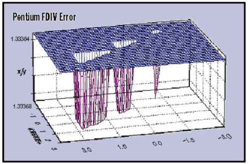
Livros recomendados/Books

- Logic in Computer Science, [HR04] (Cap. 3, 4, 6)
- Rigorous Software Development, [AFPMdS11]
- Semantics with Applications [NN07]
- Deductive procedures [KS16]

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

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

Critical systems and software errors

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

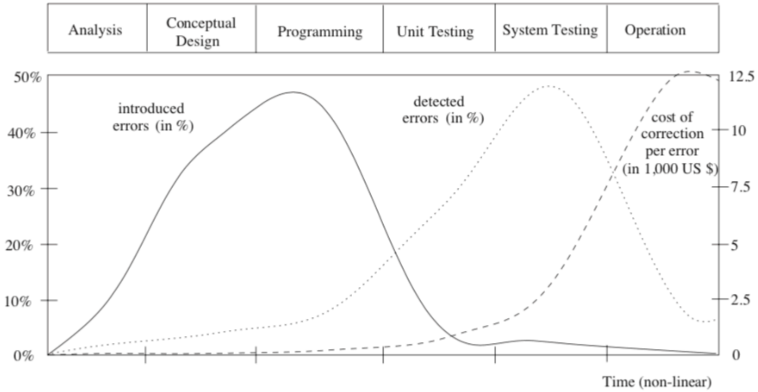
Control systems:

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

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

But, for instance Windows 95 had at least 5000 errors!

Software cycle of life, errors and costs



- 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

- 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

- 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

- 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

- 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

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 modelled 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

Central notions and techniques/verification tools

- The operational essence of the modelled 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

Central notions and techniques/verification tools

- The operational essence of the modelled 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

- 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

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

- 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

- 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

- 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

- 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

- 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

- 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

- 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

- Abstract State Machines: system described by states and by a finite set of transition rules between states
- Category and set theories: states described by mathematical structures (sets, relations or functions); transitions expressed as invariants, pre-conditions 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)

- Abstract State Machines: system described by states and by a finite set of transition rules between states
- Category and set theories: states described by mathematical structures (sets, relations or functions); transitions expressed as invariants, pre-conditions 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)

State-based Specifications

- Abstract State Machines: system described by states and by a finite set of transition rules between states
- Category and set theories: states described by mathematical structures (sets, relations or functions); transitions expressed as invariants, pre-conditions 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)

State-based Specifications

- Abstract State Machines: system described by states and by a finite set of transition rules between states
- Category and set theories: states described by mathematical structures (sets, relations or functions); transitions expressed as invariants, pre-conditions 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)

- Abstract Machines:
  - ASM_Gopher was the 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.

- Abstract Machines:
  - ASM_Gopher was the 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.

- Abstract Machines:
  - ASM_Gopher was the 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.

- Abstract Machines:
  - ASM_Gopher was the 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

- Abstract Machines:
    - ASM_Gopher was the 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.

- Abstract Machines:
  - ASM_Gopher was the 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.

- 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, are based on hybrid automata in order to model dynamic systems with an interactive behaviour(e.g.*transportation systems*)

- 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, are based on hybrid automata in order to model dynamic systems with an interactive behaviour(e.g.*transportation systems*)

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, are based on hybrid automata in order to model dynamic systems with an interactive behaviour(e.g.*transportation systems*)

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, are based on hybrid automata in order to model dynamic systems with an interactive behaviour(e.g.*transportation systems*)

- 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

- 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

- 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

- 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

- 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).

- 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).

- 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).

- 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).

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

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

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

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

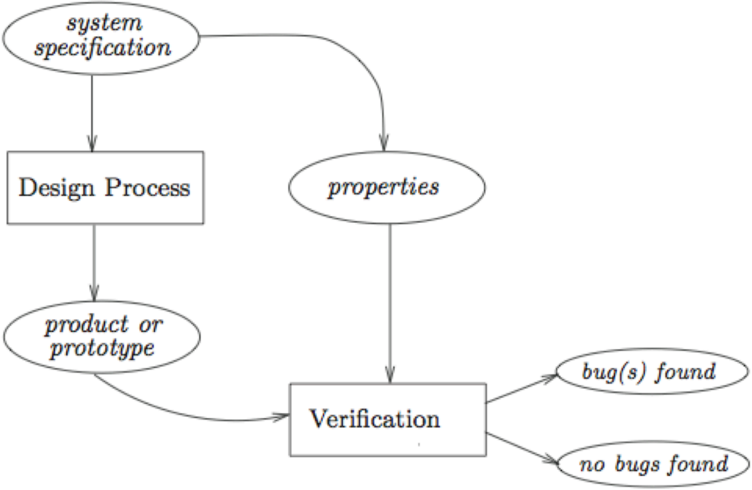
Modelling systems platform

Specification Language 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



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 $\mathcal{M}$, 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.

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 may be mixed. For example for embedded system or *proof carrying code* systems.

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, Lean, etc
  - Combine two capacities: proof check and assisted proof construction
  - Proofs are build interactively using tactics: case, elim, change, rewrite, simpl, discriminate, injection, induction.

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

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

Daniel Kroening and Ofer Strichman.
Decision Procedures:An Algorithmic Point of View.
Texts in Theoretical Computer Science. An EATCS Series.
Springer, 2016.

H. Nielson and F. Nielson.
Semantics with Applications: an appetizer.
Springer, 2007.