A Review of Formal Methods

200611499 이낙원
200611520 진경훈
200611521 최정명
200911411 이상규

Contents

- Introduction
- Definition and overview
- Specification Methods
- Life cycles and technologies

INTRODUCTION

- Better programs
- The best methods
- Revolutionary paradigm shift
- A highly controversial subject

Definition and overview of Formal methods

- A broad view of **formal methods** includes all applications of discrete mathematics to software engineering problems. This application usually involves modeling and analysis where the models and analysis procedures are derived from or defined by an underlying mathematically precise foundation. (Leveson 1990)

Definition and overview of Formal methods

- First, formal methods involve the use of a formal language.
  - Then why formal methods involve a formal language?
- Second, formal methods support formal reasoning about formula in the language.
  - For what?

Use of Formal methods

- Directly applicable during the requirements, design, and coding phases
- Important consequences for testing and maintenance
- The development and standardization of many programming languages

What can be formally specified

- Precise and rigorous specifications
  - Should describe what a system should do
  - But not how it is done
  - Aspects of a system other than functionality
    - For example, formal methods are sometimes applied in practice to ensure software safety and security properties of computer programs.

What can be formally specified

- Example of the most well known formal methods
  - **Z (pronounced "Zed")**
  - ML
  - Communicating Sequential Processes (CSP)
  - Vienna Development Method (VDM)
  - Larch

What can be formally specified

- **Z notation** (pro
  - Formal specific
    and modeling
  - Z is based on t
    used in axioma
    first-order prec

Ausdrücke		Formeln	
Ausgabe	*Erklärung*	*Ausgabe*	*Erklärung*
$[x:T \mid P \circ E]$	Mengenkomprehension (E optional),z.B. $[x:\mathbb{Z} \mid x>5 \circ x*x] = [36,49,64,\ldots]$	$x=y, x\neq y$	(Un-)Gleichheit
		$x\in S, X\notin S$	(Nicht-)Enthaltensein
		$S\subseteq T$	Teilmenge
$(\lambda x:T \mid P \circ E)$	λ-Ausdruck, äquivalent zu $[x:T \mid P \circ (x,E)]$	$S\subset T$	echte Teilmenge
		$\neg P$	Negation
fx	Funktionsanwendung	$P\wedge Q$	Konjunktion
$(\mu x:T \mid P \circ E)$	eindeutige Beschreibung(E optional),z.B. $(\mu x:\mathbb{N}_1 \mid x*x=x\circ x+x)=$	$P\vee Q$	Disjunktion
		$P\Rightarrow Q$	Implikation
if P then E1 else E2	bedingter Ausdruck (analog zu Formeln)	$P\Leftrightarrow Q$	Äquivalenz
		$\forall x:T \circ P$	Allquantifizierung
		$\exists x:T \circ P$	Existenzquantifizierung
let $x==E1 \circ E2$	lokale Definition (analog zu Formeln)	$\exists_1 x:T \circ P$	eindeutige Existenz
		$a\underline{R}b$	Infix-Relation, äquivalent $(a,b)\in R$

Reasoning about a Formal Description

- Usable formal methods provide a variety of techniques for reasoning about specifications and drawing implications.

Reasoning about a Formal Description

- Formal methods provide reasoning techniques to explore these questions.
  - Does a description imply a system should be in several states simultaneously?
  - Do all legal inputs that yield one and only one output?
  - What surprising results, perhaps unintended, can be produced by a system?

Tools and Methodology

- The ultimate end product
  - Not solely working system
  - Specifications, Demonstrations that the program meets its specification
- A proof is very hard to develop after the fact. Consequently, proofs and programs should be developed in parallel.

Tools and Methodology

- Formal methods have also inspired the development of many tools.
  - Programs to help maintain and automate proofs are example of such tools.
- In some sense, no programmer can avoid formal methods, for every programming language is, by definition, a formal language.

Limitation of Formal methods

- Requirement problem
- Physical implementations problem
- Implementation Issues problem

The Requirement Problem

- "You cannot go from the informal to the formal by formal means."
- Formal methods can be used to verify a system, but not to validate it.
  - Verify vs. Validate

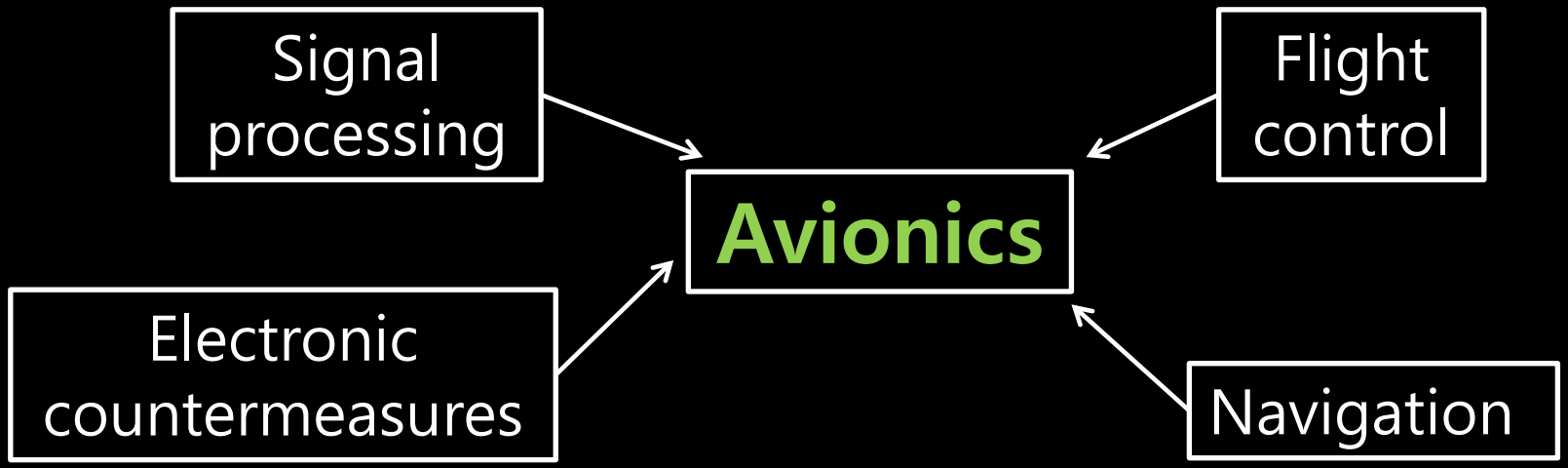
The Requirement Problem

- The three most important problems in software development are:
  - The thin spread of application domain knowledge
  - Changes in and conflicts between requirements
  - Communication and coordination problems

  - **Key Exeptional Designer!**

The Requirement Problem

- The application knowledge of the exceptional designer is not limited to one discipline.



The Requirement Problem

- However, empirical evdence
  - Contribution to the capturing requirement
  - Fewer errors
- Still, formal methods can never replace deep application knowledge on the par of the requirements engineer.

Physical implementations

- Run on an idealized abstract machine, but not when run on any physical machine.
  - For instance, an abstract machine might be assumed to have an infinite memory.
  - A compiler may not correctly implement a language as specified.
  - Memory chips and integrated circuits may contain bugs.

Physical implementations

- These limitations do not imply that formal methods are pointless.
- Formal proofs explicitly isolate those locations where an error may occur.

Implementation Issues

- The **gaps** between
  - Users' intentions & formal specifications
  - Physical implementations & abstract proofs

  ➡ Create inherent limitations.

Implementation Issues

- Formal methods are most well developed for addressing issues of functionality, safety, and security, but not of **Scalability**.
- Issue of **scaling** can be a deciding factor in the choice of method.

Implementation Issues

- **Alternative**
  - To select a small subset of components for formal treatment, thus finessing the scalability issue.
  - To develop formal specifications at the start of the life cycle and then automatically derive the source code for the system.
  - To partially introduce formal methods by introducing them throughout an organization or project but allowing a variable level of formality.

Implementation Issues

- Training and education issues
  - Many programmers have either not been exposed to the needed mathematical background or do not use it in their day-to-day practice.
  - Old one need to be retrained or modified, and new one need to be hired.

SPECIFICATION METHODS

- Users of formal specification techniques need to understand the strengths and weaknesses of different methods and languages before deciding on which to adopt
- A **method** states what a specification must say.
- A **language** determines in detail how the concepts in a specification can be expressed.

Semantic Domains

- A **formal specification language** contains an alphabet of symbols and grammatical rules that define well-formed formulae.
- A language can have several models, but most will find some models more natural than others.

Semantic Domains

- The objects in the language's semantic domain that satisfy a given specification can be **nonunique**.
  - Because of this nonuniqueness, specification is at a higher level of abstraction than the objects in the semantic domain.
- These concepts can be defined more precisely using mathematics.

Semantic Domains

- Specification language can be classified based on their semantic domains. Three major classes of semantic domains exist.
  - Abstract data type (ADT) specification languages
    - Used to describe algebras
  - Process specification language
    - Describe state sequences, event sequences etc.
  - Programming languages
    - Obvious example of languages with multiple models

Model-Oriented and Property-Oriented Methods

- Model-oriented methods have been described as constructive or operational.
- In a model-oriented method, a specification describes a system directly by providing a model of the system.
- In effect, a model-oriented specification is a program written in a very high-level language.

Model-Oriented and Property-Oriented Methods

- Property-oriented methods are described as definitional or declarative.
- Property-oriented specification describes a minimum set of conditions that a system must satisfy.
- But the specification does not provide a mechanical model showing how to determine the output of the system from the inputs.

Model-Oriented and Property-Oriented Methods

- Two classes of property-oriented methods exist.
  - Algebraic & Axiomatic
    - In algebraic methods, the properties defining a program are equations in certain algebras.
    - ADT are often specified by algebraic methods.
    - Other types of axioms can be used in axiomatic methods.

Use of Specification Methods

- In general, formal methods provide for more precise specifications.
  - Misunderstandings and bugs can be discovered earlier in the life cycle.
  - Since the fault is detected, the cheaper it can be removed, formal specification methods can dramatically improve both productivity and quality.

Use of Specification Methods

- Formal specifications should not be presented without a restatement of the specification in a natural language.
  - Very few sponsors of a software project will be inclined to read a specification whose presentation is entirely in a formal language.

LIFE CYCLES AND TECHNOLOGIES WITH INTEGRATED FORMAL METHODS

- To get their full advantages in a cost-effective manner, formal methods should be incorporated into a software organization's standard procedures.
- Two methods of integrating formal methods in software processes can be distinguished.
  - One with heavy use of automated tools
  - One with nonmechanical, nonautomated proofs

Verification Systems and Other Automated Tools

- An automated verification system provides a means for the user to demonstrate the existence of a formal proof of a software system.
- The usage of verification systems varies.
  - Some allow the user of high level of control.
  - The system is useful for bookkeeping.
  - Verifiers often use decision procedures in restricted domains.
    - Decision procedures show whether or not a proof exists without explicitly constructing the proof.

Verification Systems and Other Automated Tools

- Model checking
  - One creates a state transition diagram as a model of a specification.
  - Model checking either establishes that the properties are true for the state transition diagram model or provides a counterexample.

The Cleeanroom as a Life Cycle with Integrated Use of Formal methods

- The Cleanroom methodology integrates nonmechanized formal methods into the life cycle.
  - Cleanroom combines formal methods and structured programming with SPC, the spiral life cycle and etc.
- Cleanroom fosters attitudes such as emphasizing defect prevention over defect removal.

The Cleeanroom as a Life Cycle with Integrated Use of Formal methods

- The design and coding phases of Cleanroom development are distinctive.
  - Analysts must develop proofs of correctness, along with designs and code.
- Testing does play a very important role in Cleanroom.
  - It serves to verify that reliability goals are attained.

CONCLUSIONS

- Formal methods can provide:
  - More precise specifications
  - Better internal communication
  - An ability to verify designs before executing them during test
  - Higher quality and productivity
- Even if formal methods are not integrated into an organization's process, they can still have positive benefits.

CONCLUSIONS

- Technologies that are increasingly widespread today draw on formal methods. Knowledge of formal methods is needed to completely understand these popular technologies and to use them most effectively.
  - Rapid prototyping
  - Object-oriented design
  - Structured programming
  - Formal inspections