Einführung in die formale Spezifikation von Software

Bernhard Beckert

Universität Koblenz-Landau

Winter 2008/2009
Web page

All information relevant to this lecture can be found on the web page

www.uni-koblenz.de/~beckert/Lehre/Formale-Spezifikation/
This Course / Web Page

Web page

All information relevant to this lecture can be found on the web page

www.uni-koblenz.de/~beckert/Lehre/Formale-Spezifikation/

Make this a lively course

Ask questions!
Contents

- Introduction: Formal Methods and Formal Specification
Contents

- Introduction: Formal Methods and Formal Specification
- Design by Contract
Contents

- Introduction: Formal Methods and Formal Specification
- Design by Contract
- Invariants, Pre- and Post-Conditions
Contents

- Introduction: Formal Methods and Formal Specification
- Design by Contract
- Invariants, Pre- and Post-Conditions
- Object Constraint Language (OCL)
Contents

- Introduction: Formal Methods and Formal Specification
- Design by Contract
- Invariants, Pre- and Post-Conditions
- Object Constraint Language (OCL)
- Java Modelling Language (JML)
Contents

- Introduction: Formal Methods and Formal Specification
- Design by Contract
- Invariants, Pre- and Post-Conditions
- Object Constraint Language (OCL)
- Java Modelling Language (JML)
- State Charts
Contents

- Introduction: Formal Methods and Formal Specification
- Design by Contract
- Invariants, Pre- and Post-Conditions
- Object Constraint Language (OCL)
- Java Modelling Language (JML)
- State Charts
- Abstract Data Types
What are Formal Methods?

Software Development Methods

- Analysis
- Modelling (Specification)
- Implementation
- Validation (Verification, Testing)
What are Formal Methods?

Software Development Methods

- Analysis
- Modelling (Specification)
- Implementation
- Validation (Verification, Testing)

... using ...

- Languages and notations with (mathematical) precise semantics
- Logic-based techniques
What are Formal Methods?

**Software Development Methods**

- Analysis
- Modelling (Specification)
- Implementation
- Validation (Verification, Testing)

...using...

- Languages and notations with (mathematical) precise semantics
- Logic-based techniques

**Note**

formal ≠ theoretical
Why Formal Methods?

**Quality: Important for . . .**

- Safety-critical applications  
  (railway switches)
- Security-critical applications  
  (access control, electronic banking)
- Financial reasons  
  (phone cards)
- Legal reasons  
  (electronic signature, EAL6/7 in Common Criteria)
Why Formal Methods?

**Quality: Important for . . .**

- Safety-critical applications (railway switches)
- Security-critical applications (access control, electronic banking)
- Financial reasons (phone cards)
- Legal reasons (electronic signature, EAL6/7 in Common Criteria)

**Productivity: Important for . . .**

Obvious reasons
Why Formal Methods?

Quality through . . .

- Better and more precise understanding of model and implementation
- Better written software (modularisation, information hiding, . . .)
- Error detection with runtime checks
- Test case generation
- Static analysis
- Deductive verification
Why Formal Methods?

Productivity through

- Error detection in early stages of development
- Re-use of components (requires specification and validation)
- Better documentation, maintenance
- Test case generation
- Knowledge about formal methods leads to better software development
Testing

Run the system at chosen inputs and observe its behaviour

- Randomly chosen
- Intelligently chosen (by hand: expensive!)
- Automatically chosen (need formalized spec)
Testing

- Run the system at chosen inputs and observe its behaviour
  - Randomly chosen
  - Intelligently chosen (by hand: expensive!)
  - Automatically chosen (need formalized spec)

- What about other inputs? (test coverage)
Testing

- Run the system at chosen inputs and observe its behaviour
  - Randomly chosen
  - Intelligently chosen (by hand: expensive!)
  - Automatically chosen (need formalized spec)

- What about other inputs? (test coverage)

- What about the observation? (test oracle)
Testing

- Run the system at chosen inputs and observe its behaviour
  - Randomly chosen
  - Intelligently chosen (by hand: expensive!)
  - Automatically chosen (need formalized spec)

- What about other inputs? (test coverage)

- What about the observation? (test oracle)

Challenges can be addressed by/require formal methods
Favourable Development

Design and specification

- Unified Modeling Language – UML
  Graphical language for object-oriented modelling
  Standard of Object Management Group (OMG)

- Object Constraint Language – OCL
  Formal textual assertion language
  UML Substandard
Design and specification

- Unified Modeling Language – UML
  Graphical language for object-oriented modelling
  Standard of Object Management Group (OMG)

- Object Constraint Language – OCL
  Formal textual assertion language
  UML Substandard

- Consolidation and documentation of design knowledge
  Patterns, idioms, architectures, frameworks, etc.
Favourable Development

Design and specification

- Unified Modeling Language – UML
  - Graphical language for object-oriented modelling
  - Standard of Object Management Group (OMG)

- Object Constraint Language – OCL
  - Formal textual assertion language
  - UML Substandard

- Consolidation and documentation of design knowledge
  - Patterns, idioms, architectures, frameworks, etc.

Industrial implementation languages

- Java, C#
Types of Requirements

- functional requirements
- communication, protocols
- real-time requirements
- memory use
- security
- robustness
- etc.
Types of Requirements

- functional requirements
- communication, protocols
- real-time requirements
- memory use
- security
- robustness
- etc.

Different Formal Methods

- deductive verification
- model checking
- static analysis
- run-time checks (of formal specification)
### Types of Requirements

- **functional requirements**
- communication, protocols
- real-time requirements
- memory use
- security
- robustness
- etc.

### Different Formal Methods

- deductive verification
- model checking
- static analysis
- run-time checks (of formal specification)
Limitations of Formal Methods

Possible reasons for errors

- Program is not correct (does not satisfy the specification)
  Formal verification proves absence of this kind of error

- Program is not adequate (error in specification)
  Formal specification/verification avoid/find this kind of error

- Error in operating system, compiler, hardware
  Not avoided (unless compiler etc. specified/verified)
Limitations of Formal Methods

Possible reasons for errors

- Program is not correct (does not satisfy the specification)
  Formal verification proves absence of this kind of error

- Program is not adequate (error in specification)
  Formal specification/verification avoid/find this kind of error

- Error in operating system, compiler, hardware
  Not avoided (unless compiler etc. specified/verified)

No full specification/verification

In general, it is neither useful nor feasible to fully specify and verify large software systems. Then, formal methods are restricted to:

- Important parts/modules
- Important properties/requirements
The Main Point of Formal Methods is Not

- To show “correctness” of entire systems (What IS correctness? Always go for specific properties!)
- To replace testing entirely
- To replace good design practices

There is no silver bullet that lets you get away without writing crystal clear requirements and good design, in particular, Formal Methods aren’t one.
But

- Formal proof can replace many test cases
- Formal methods can be used in automatic test case generation
- Formal methods improve the quality of specifications
A Fundamental Fact

Formalisation of system requirements is hard
Difficulties in Creating Formal Models

Diagram showing the relationship between the Real World and Formal Requirements Specification through the process of abstraction.
Difficulties in Creating Formal Models

- Real World
- Formal Model

Wrong assumption

eg, timing
Difficulties in Creating Formal Models

Real World -> Formal Model

missing requirement
eg, stack overflow
Difficulties in Creating Formal Models

- Real World
- Formal Model

misunderstood problem
eg, wrong integer model
Another Fundamental Fact

Proving properties of systems can be hard
System Abstraction Level

- **Low level of abstraction**
  - Finitely many states
  - Tedious to program, worse to maintain
  - Automatic proofs are (in principle) possible

- **High level of abstraction**
  - Complex datatypes and control structures
  - Easier to program
  - Automatic proofs (in general) impossible!
Specification Abstraction Level

Low level of abstraction
- Finitely many cases
- Approximation, low precision
- Automatic proofs are (in principle) possible

High level of abstraction
- General properties
- High precision, tight modeling
- Automatic proofs (in general) impossible!
## Main Approaches

<table>
<thead>
<tr>
<th>High-level programs, Complex properties</th>
<th>High-level programs, Simple properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-level programs, Complex properties</td>
<td>Low-level programs, Simple properties</td>
</tr>
</tbody>
</table>
### Main Approaches

<table>
<thead>
<tr>
<th>High-level programs, Complex properties</th>
<th>High-level programs, Simple properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-level programs, Complex properties</td>
<td>Low-level programs, Simple properties</td>
</tr>
</tbody>
</table>

Model Checking
## Main Approaches

<table>
<thead>
<tr>
<th>KeY System</th>
<th>Model Checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-level programs, Complex properties</td>
<td>High-level programs, Simple properties</td>
</tr>
<tr>
<td>Low-level programs, Complex properties</td>
<td>Low-level programs, Simple properties</td>
</tr>
</tbody>
</table>
Proof Automation

“Automatic” Proof
- No interaction
- Sometimes help is required anyway
- Formal specification still “by hand”

“Semi-Automatic” Proof
- Interaction may be required
- Very often proof tool suggests proof rules
- Proof is checked by tool
SPIN at Bell Labs

Feature interaction for telephone call processing software

- Tool works directly on C source code
- Web interface to track properties
- Work farmed out to large numbers of computers
- Finds shortest possible error trace
- 18 months, 300 versions, 75 bugs found
- Main burden: Defining meaningful properties
Device drivers running in “kernel mode” should respect API

Third-party device drivers that do not respect APIs responsible for 90% of Windows crashes

SLAM inspects C code, builds a finite state machine, checks requirements

Being turned into a commercial tool right now
Future Trends

- Design for formal verification
- Combining automatic methods with theorem provers
- Combining static analysis of programs with automatic methods and with theorem provers
- Combining test and formal verification
- Integration of formal methods into SW development process
- Integration of formal method tools into CASE tools
Formal Methods

- Are (more and more) used in practice
- Can shorten development time
- Can push the limits of feasible complexity
- Can increase product quality
Formal Methods

- Are (more and more) used in practice
- Can shorten development time
- Can push the limits of feasible complexity
- Can increase product quality

Those responsible for software management should consider formal methods, in particular, where safety-critical, security-critical, and cost-intensive software is concerned.