Formale Spezifikation und Verifikation
(Formal Methods in Software Engineering)

Bernhard Beckert

Universität Koblenz-Landau

Summer Term 2008
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/
Spezifikation-Verifikation/
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/
Spezifikation-Verifikation/

Make this a lively course

📝 Ask questions!
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/
Spezifikation-Verifikation/

Make this a lively course

☐ Ask questions!

☐ Don’t fall asleep
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/
Spezifikation-Verifikation/

Make this a lively course

- Ask questions!
- Don’t fall asleep
- Keep cool
Contents

Why verification?
Advantages and disadvantage. Costs and gains.
Contents

- Why verification?
  Advantages and disadvantage. Costs and gains.

- Basics of deductive program verification:
  Hoare Logic and Dynamic Logic
Contents

- Why verification?
  Advantages and disadvantage. Costs and gains.

- Basics of deductive program verification:
  Hoare Logic and Dynamic Logic

- Deductive verification of object-oriented programming languages
  (using Java as an example)
Contents

- Why verification?

- Basics of deductive program verification:
  - Hoare Logic and Dynamic Logic

- Deductive verification of object-oriented programming languages
  (using Java as an example)

- Writing and understanding formal specifications
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 $\neq$ 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
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.
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

<table>
<thead>
<tr>
<th>Types of Requirements</th>
<th>Different Formal Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>functional requirements</td>
<td>deductive verification</td>
</tr>
<tr>
<td>communication, protocols</td>
<td>model checking</td>
</tr>
<tr>
<td>real-time requirements</td>
<td>static analysis</td>
</tr>
<tr>
<td>memory use</td>
<td>run-time checks (of formal specification)</td>
</tr>
<tr>
<td>security</td>
<td></td>
</tr>
<tr>
<td>robustness</td>
<td></td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>
### 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
Formal proof can replace many test cases

Formal methods can be used in automatic test case generation

Formal methods improve the quality of specifications