A Fundamental Fact

Formalisation of system requirements is hard
Difficulties in Creating Formal Models

Real World

Abstraction

Formal
Execution
Model

Formal
Requirements
Specification

Formal Methods in Software Engineering – p.15
Difficulties in Creating Formal Models

Real World

Formal Model

Wrong assumption
eg, timing
Difficulties in Creating Formal Models

- Real World
  - missing requirement
    - eg, stack overflow
  - Formal Model
Difficulties in Creating Formal Models

Real World

misunderstood problem
eg, wrong integer model

Formal 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>
</tr>
</thead>
<tbody>
<tr>
<td>Simple properties</td>
</tr>
<tr>
<td>Low-level programs, Complex properties</td>
</tr>
<tr>
<td>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
SLAM at Microsoft

- 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.