Formal Verification of Software

Bernhard Beckert

UNIVERSITÄT KOBLENZ-LANDAU

Summer Term 2005

Web Page

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

www.uni-koblenz.de/˜beckert/Lehre/Verification

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)

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

Types of Requirements

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

Types of Requirements

Types of Requirements

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

Different Formal Methodsx

- deductive verification
- model checking
- static analysis
- run-time checks
  (of formel specification)

Types of Requirements

Types of Requirements

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

Different Formal Methodsx

- deductive verification
- model checking
- static analysis
- run-time checks
  (of formel 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 feasable 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

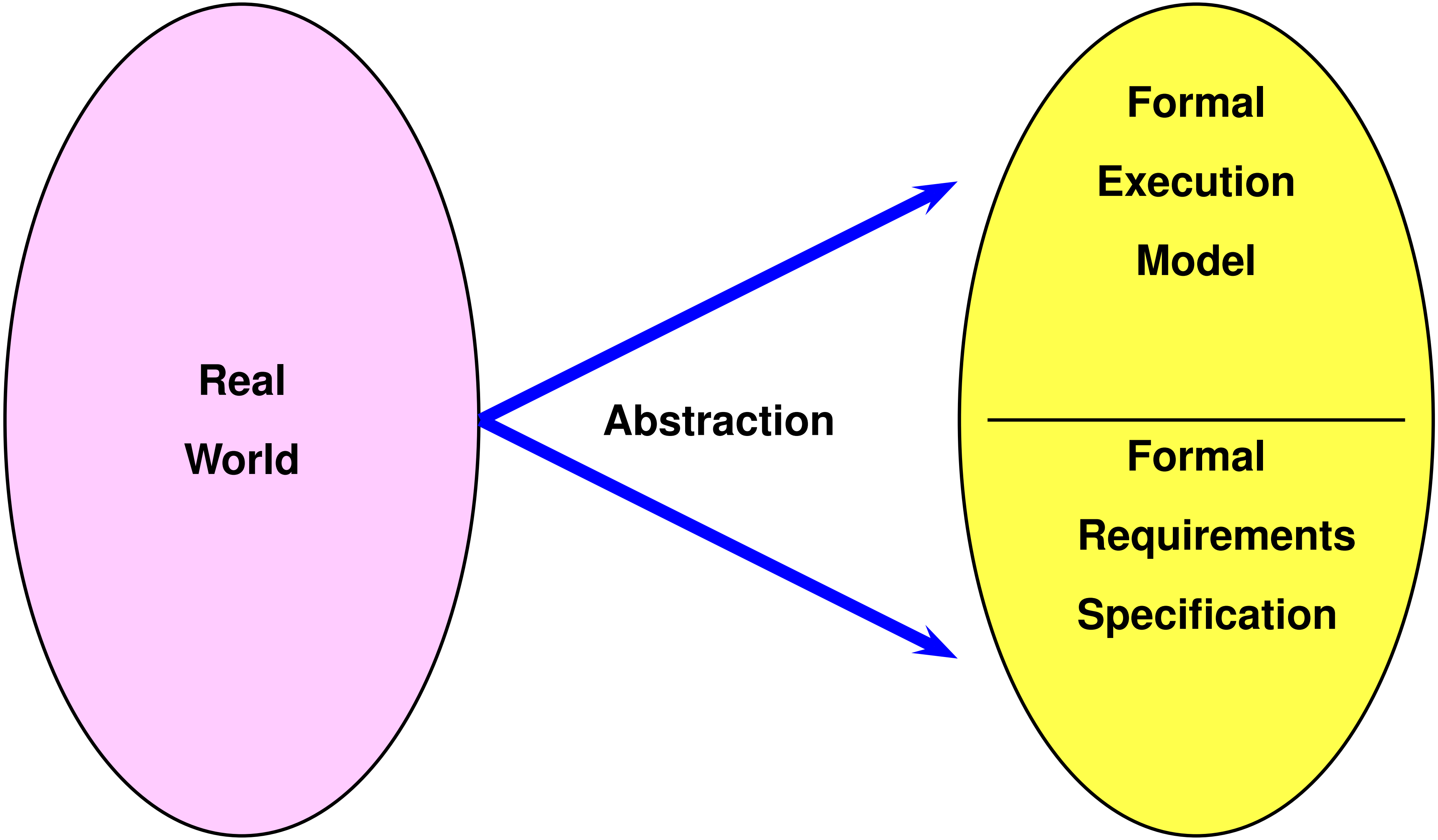
But

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

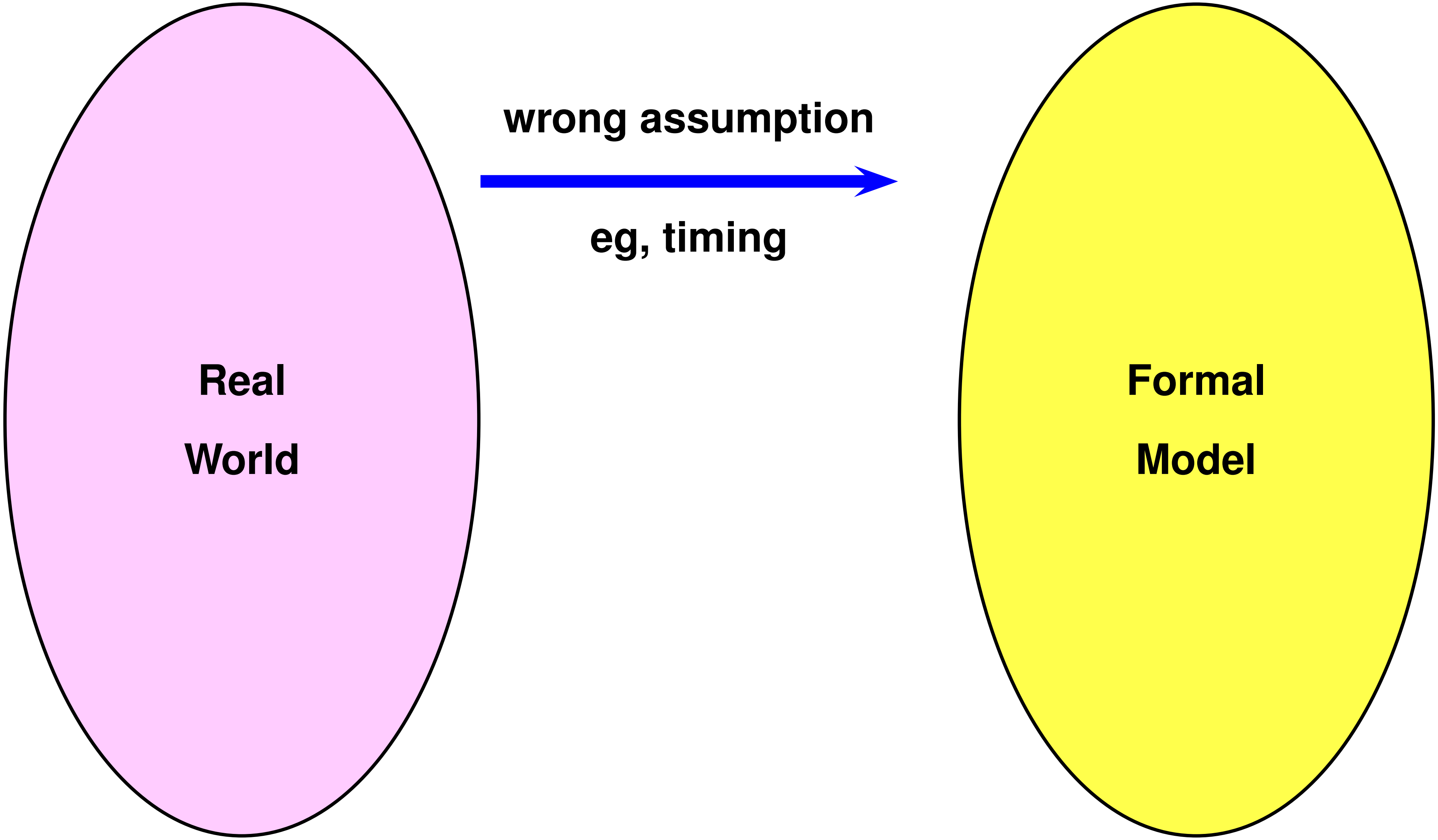
A Fundamental Fact

Formalisation of system requirements is hard

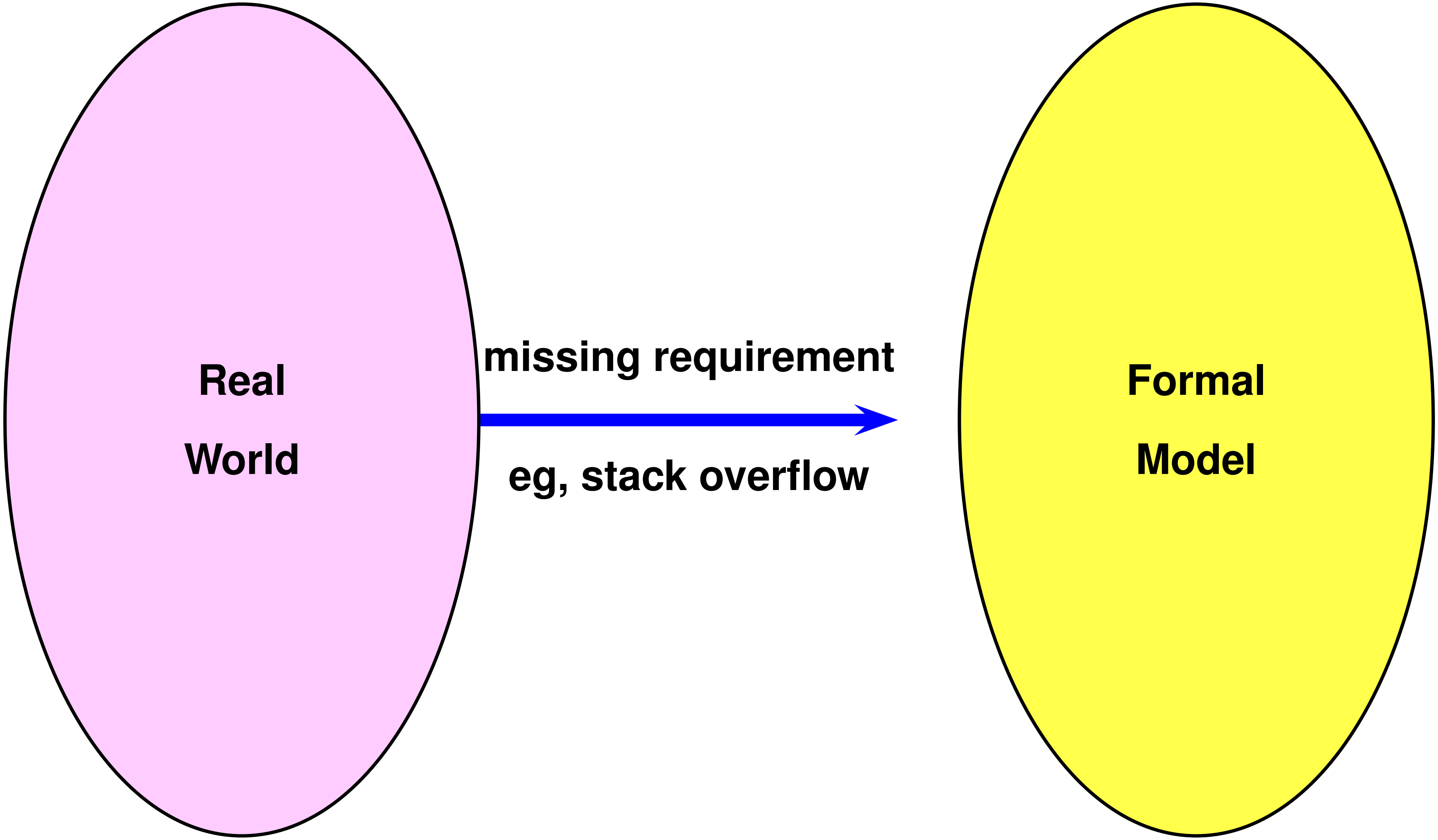
Difficulties in Creating Formal Models



Difficulties in Creating Formal Models



Difficulties in Creating Formal Models



Real

World

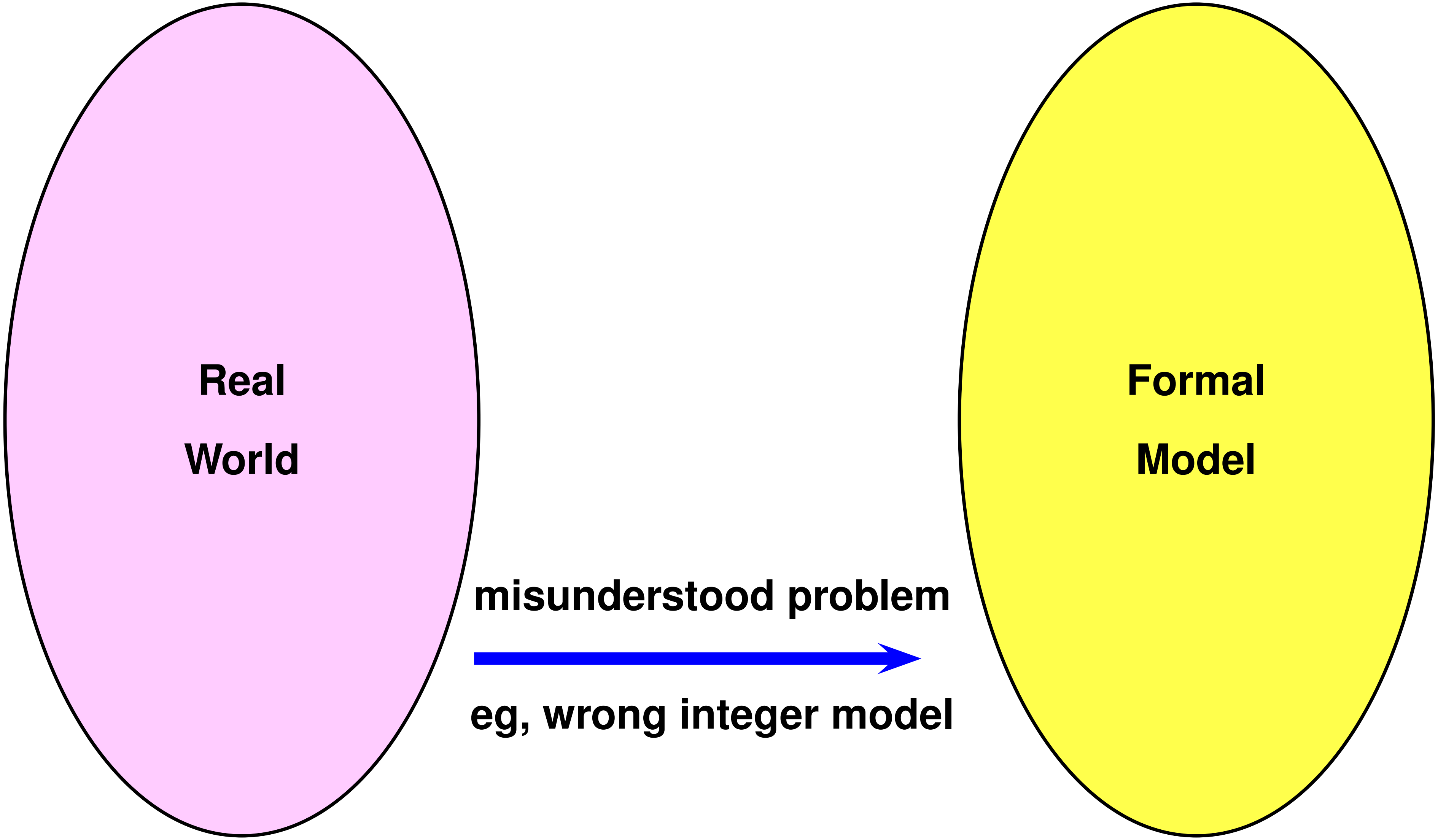
missing requirement

eg, stack overflow

Formal

Model

Difficulties in Creating Formal Models



Real World

Formal Model

misunderstood problem

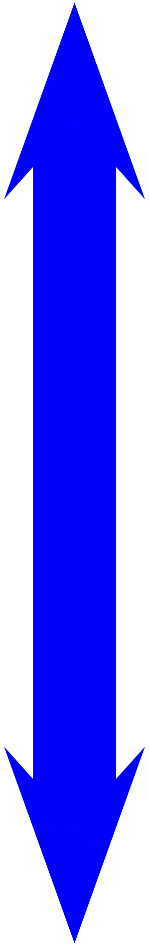
eg, wrong integer model

Another Fundamental Fact

Proving properties of systems can be hard

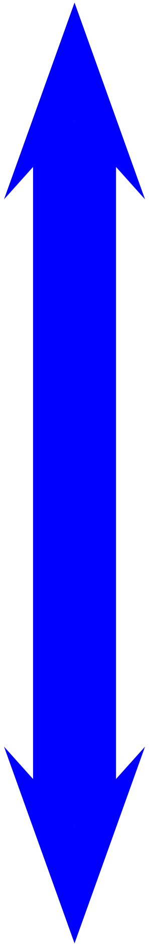
System Abstraction Level

- 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

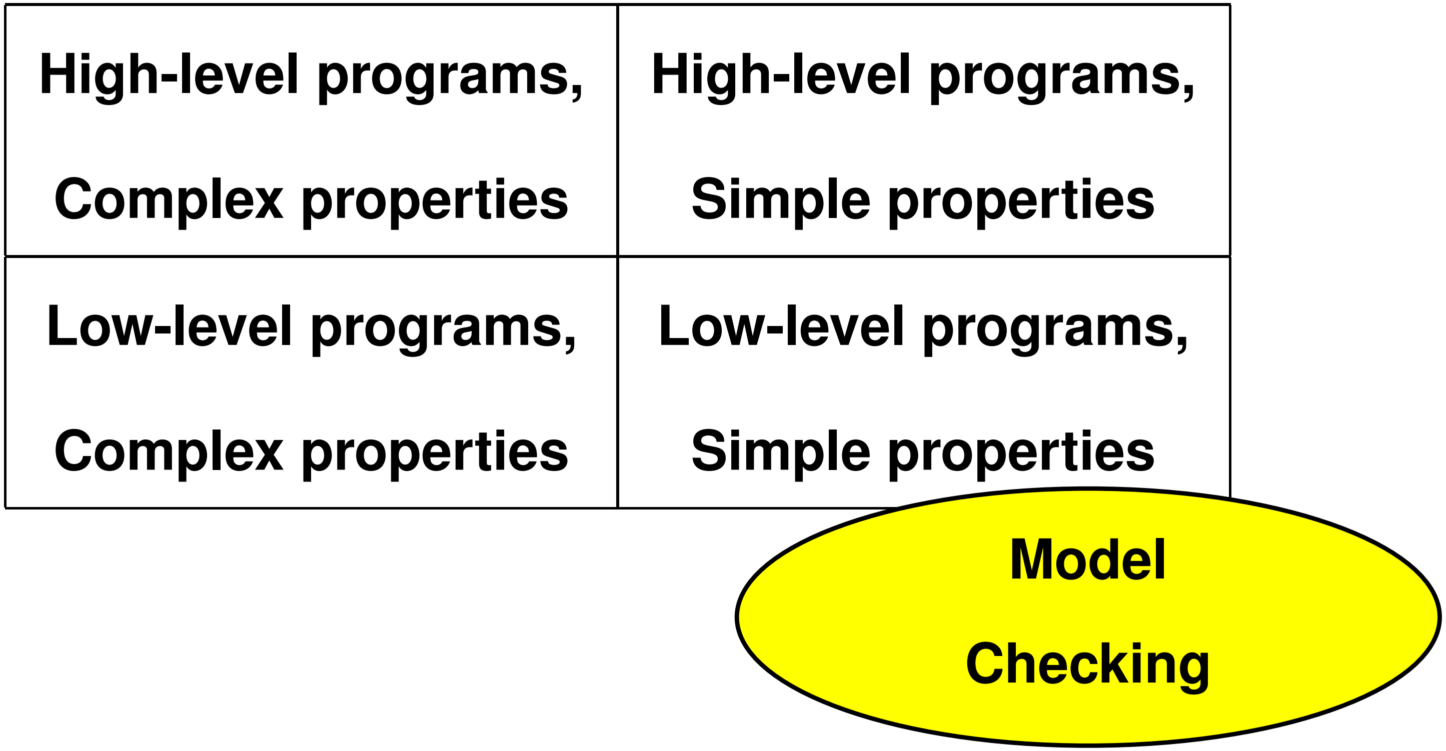
- Low level of abstraction
  - Finitely many cases
  - Approximation, low precision
  - Automatic proofs are (in principle) possible
- High level of abstraction
  - General properties
  - High precision, tight modeling
  - Automatic proofs (in general) impossible!



Main Approaches

High-level programs, Complex properties	High-level programs, Simple properties
Low-level programs, Complex properties	Low-level programs, Simple properties

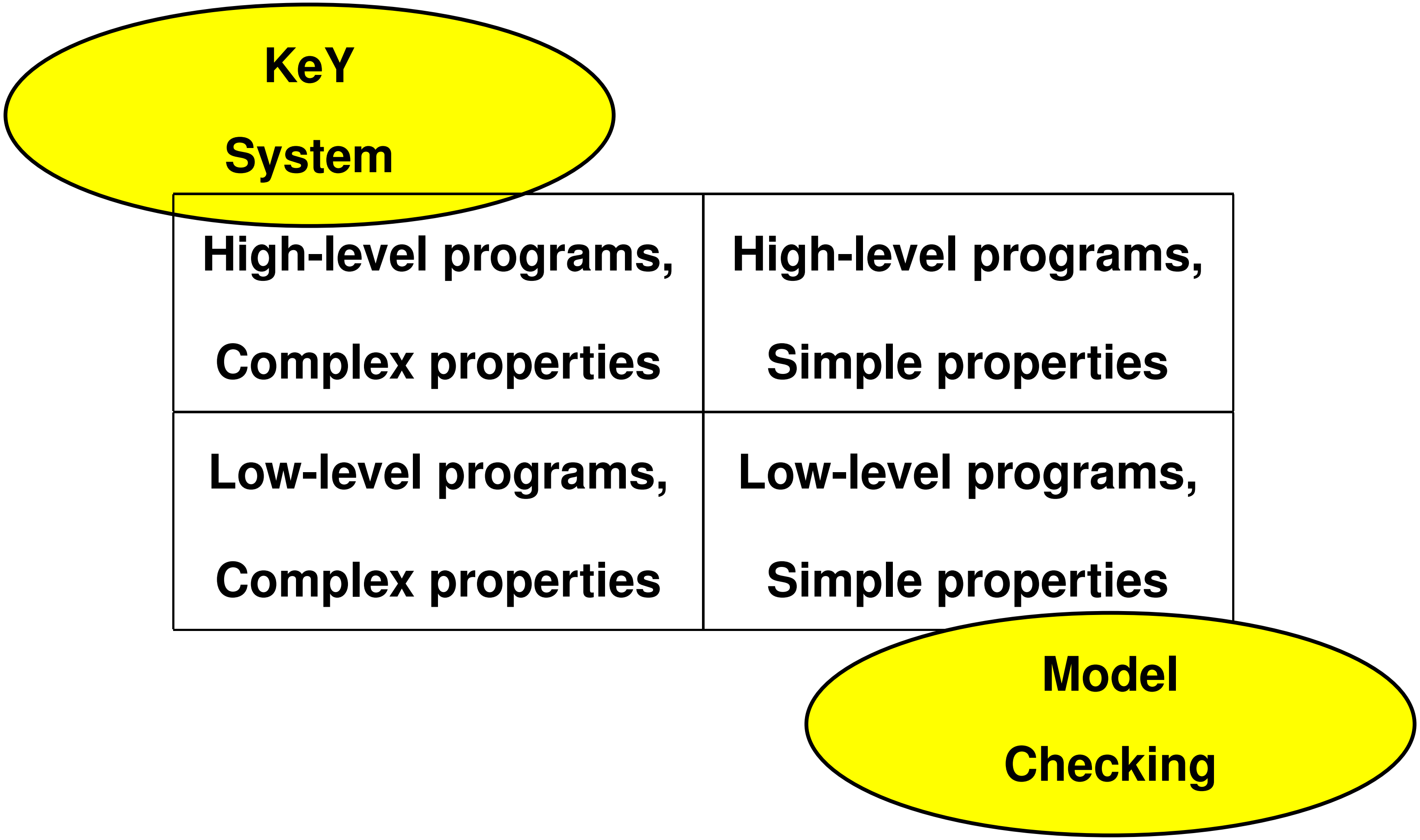
Main Approaches



High-level programs, Complex properties	High-level programs, Simple properties
Low-level programs, Complex properties	Low-level programs, Simple properties

Model Checking

Main Approaches



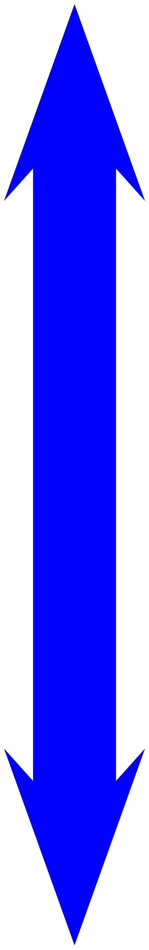
KeY System

High-level programs, Complex properties	High-level programs, Simple properties
Low-level programs, Complex properties	Low-level programs, Simple properties

Model Checking

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