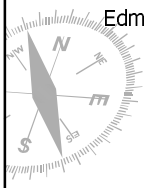
Formal Methods: State of the Art and Future Directions

Edmund M. Clarke, Jeannette M. Wing

Presented by Yifei Li

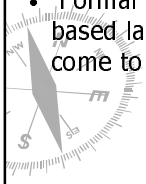


Motivation

- Cases where software failures caused a lot of troubles:
  - 2003 Blackout
  - Medical Devices
  - Phone System failures

Motivation

- How can we develop reliable software systems?
- Formal methods, which are mathematically based languages, techniques and tools, come to help



State of the Art

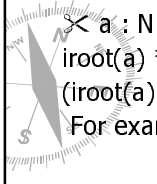
- Formal methods have been successfully applied to system *specification* and *verification*
- In verification, *model checking* and *theorem proving* are two main approaches

Specification

- Formal methods specify system behavior using mathematical syntax and semantics
- Benefits: system is specified *precisely* and better understood

Specification

- Suppose we want to write a program that calculates square root of an integer
- iroot: N ⟶ N
  ∀ a : N 🕐
  iroot(a) * iroot(a) <= a < (iroot(a)+1) * (iroot(a)+1)
  For example, 1*1 <= 3 < 2*2



Specification

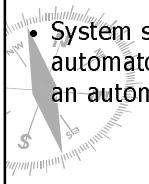
- Oxford University and IBM collaborated in 1980s on using Z to formalize part of IBM's customer information control system, a transaction server.
- Product quality was improved and cost was cut down by 9%

Model Checking

- Model checking builds a finite model of a system and checks a certain property holds in that model.
- Currently, two approaches to model checking are generally used in practice

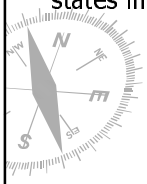
Model Checking

- System specifications are expressed in temporal logic and systems are modeled as finite state transition systems
- System specifications are given as an automaton and systems are also modeled as an automaton



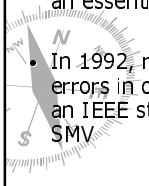
Model Checking

- Pro: The process is completely automatic
- Con: It does not scale well to the number of states in a model



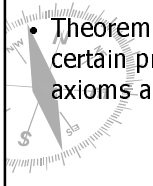
Model Checking

- Represent state transition systems more efficiently or eliminate unnecessary states
- Currently, model checkers can check systems with an essentially unlimited number of states
- In 1992, researchers at CMU found a number of errors in cache coherence protocol described by an IEEE standard using a model checking tool call SMV



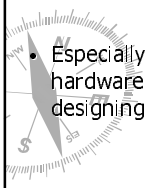
Theorem Proving

- Some mathematical logic is used to express both a system itself and its desired properties
- Theorem provers try to find a proof that a certain property does hold based on some axioms and inference rules



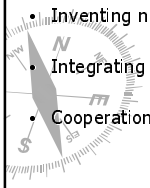
Theorem Proving

- Pro: It can deal directly with infinite state spaces
- Con: Some theorem provers need to interact with human. That process may be slow and error-prone
- Especially useful in the mechanical verification of hardware designs. IBM uses such tools when designing many CPUs such as PowerPC



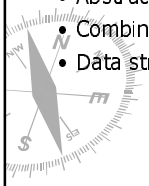
Future Directions

- Ideally, we wish formal methods could guarantee system reliability
- Fundamental research
- Inventing new methods and tools
- Integrating different methods
- Cooperation between researchers and practitioners



Future Directions

- Fundamental concepts such as
  - Composition
  - Decomposition
  - Abstraction
  - Combination of mathematical theories
  - Data structures and algorithms

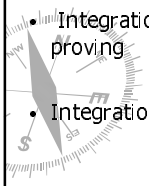


Future Directions

- New tools and methods
  - Early playback
  - Incremental gain for incremental effort
  - Ease of use and learning
  - Oriented toward error detection

Future Directions

- Integration of methods
  finding suitable style and meaning for using different methods together
- Integration of model checking and theorem proving
- Integration with system development process



Discussions

- If the mathematical logic used for theorem proving is not both sound and complete, how do we know we're given an wrong answer?
- Are there any limit on the ability of formal methods that we need to be aware of when doing further research on it?
- Is there an easy way to translate software requirements specified in natural language into formal method specifications?