

CIS 673: Computer-Aided Verification

- Instructor: Rajeev Alur (alur@cis)
- Class: Mon/Wed 10.30 – 12
- URL: www.cis.upenn.edu/cis673/

Reliability in system design

- Computer systems are getting more complex and pervasive
- In safety-critical applications, bugs are unacceptable: mission control (e.g. ARIANE-5), medical devices
- Bugs are expensive: earlier we catch them, the better. e.g. FDIV in Pentium
- Testing takes more time than designing. Automation is key to improve time-to-market.
- Increasing using of programmable components shifts focus from low-level optimizations to high-level designs
- Goal of formal verification is to provide tools and techniques as design aids to improve reliability

What is Formal verification?

- Build a mathematical *model* of the system:
what are *possible* behaviors?
- Write correctness requirements in a *specification*
language: what are *desirable* behaviors?
- Analysis: Check that model satisfies specification.

- Formal \Rightarrow Correctness claim is a precise mathematical statement
- Verification \Rightarrow Analysis either proves or disproves the correctness claim.

Alternative approaches

- Testing: Execute the actual system on selected inputs.
- Simulation: Simulate a model of the system on selected inputs (not exhaustive).
- FPGA emulation: Done in hardware as opposed to simulation in software.

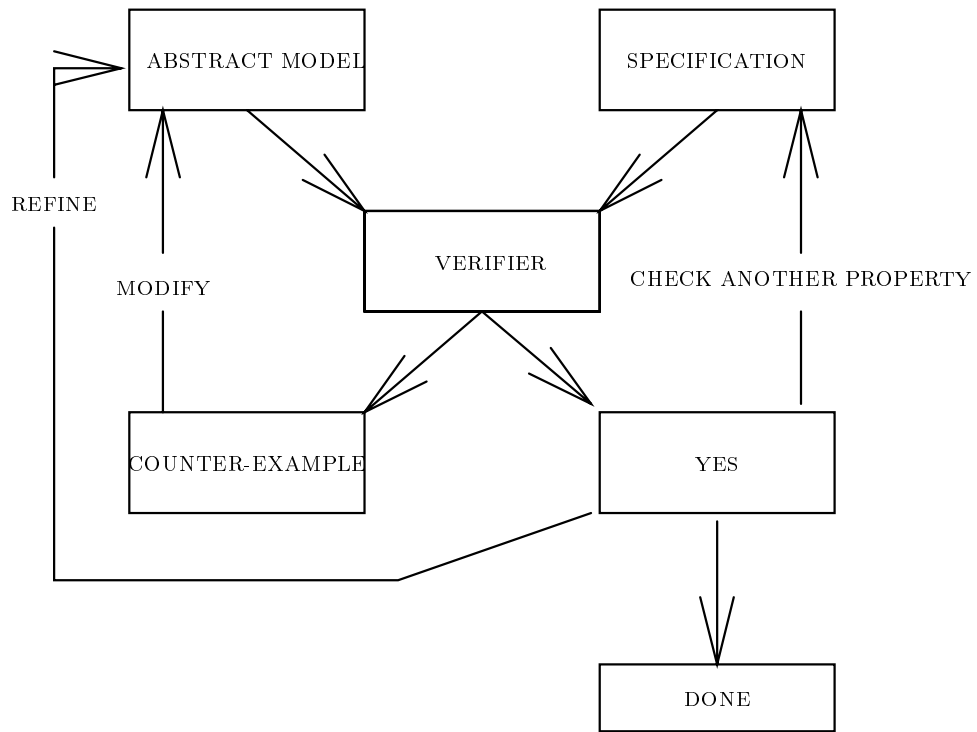
Algorithmic versus interactive verification

- Algorithmic analysis (computer-aided verification)
 - Analysis is performed by an algorithm (tool)
 - Analysis gives counter-examples for debugging
 - Typically requires exhaustive search of state-space
 - Limited by high computational complexity
- Interactive verification
 - Analysis reduces to proving a theorem in a logic
 - Uses interactive theorem prover
 - Requires more expertise

Model Checking

- Term coined by Clarke and Emerson in 1981 to mean checking a finite-state model with respect to the temporal logic CTL
- Applies generally to automated verification
 - Model need not be finite
 - Model may be extracted from code
 - Requirements in many different languages
- Provides diagnostic information to debug the model

Verification methodology



Why study computer-aided verification

- A general approach with applications to
 - hardware verification
 - systems software
 - embedded control systems
- Rapidly increasing industrial interest
- A lot of interesting mathematical foundations
 - Modeling, semantics, concurrency theory
 - Logic and automata theory
 - Analysis algorithms, data structures

Hardware verification

- Success in debugging microprocessor designs
 - Cache coherence
 - Pipelining
- Used as a debugging tools to understand what went wrong
- Who is using it?
 - Design teams: Lucent, IBM, Intel, Fujitsu, Siemens, Motorola, NEC ...
 - CAD tool vendors: Cadence, Synopsys
 - Commercial model checkers: FormalCheck
- Fits well in design flow
 - Designs in VHDL, Verilog
 - Simulation, synthesis, and verification

Beyond Hardware

- Software
 - High-level modeling not common
 - Applications: protocols, telecommunications
 - Languages: SDL, ESTEREL, UML
- Recent trend: integrate model checking in program analysis tools
 - Applied directly source code
 - Sample projects: SLAM (Microsoft), F-SOFT (NEC), CBMC (CMU), Blast (UC Berkeley)
 - Main challenge: extracting model from code
- Control systems
 - Emerging applications in avionics, robotics

Limitations

- Appropriate for control-intensive applications with interesting interaction among components
- Decidability and complexity remains an obstacle
 - Great progress in finding heuristics
 - Flexibility in setting up the problem
- Falsification rather than verification
 - Model, and not system, is verified
 - Only stated requirements are checked: how to capture correctness in a formal language?
- Finding suitable abstraction requires expertise

Overview of topics

- Modeling
- Specification languages
 - temporal logics
 - ω -automata
- Analysis techniques
 - Symbolic model checking (OBDDs, SAT)
 - Partial-order reduction
 - Minimization
- Hierarchical verification
 - Refinement preorders
 - Compositional, assume-guarantee reasoning
- Software model checking

Software MOCHA

- Model checker with combination of techniques
- Closely follows the book
- Suitable for research projects
- Developed in a joint project with UC Berkeley
- Many other model checkers available: NuSMV, Spin, PVS ...

Projects

- Independent study of a related topic
- Implementation of algorithms
- Application of formal verification
- Case studies to compare methods/tools
- Theoretical research

Reactive computation

- Functional view: A program computes a function from inputs to outputs
- Reactive view: A system interacts with environment accepting requests and producing responses
 - Computation may not terminate
 - Correctness refers to behavior over time
 - Examples: Operating system, Automatic Teller Machine, Web server, ...

Modeling

- Goal is to describe control and interaction. Hence, no complex data structures, not much data manipulation
- Explicit construct for nondeterminism (e.g. a channel may or may not lose a message)
- Simple set of operators to build complex models
- Precise mathematical semantics
- In modeling, both the system and the environment are specified
- Same system can have many different models; different levels of abstraction

Modeling Language

- Simplest language: Finite-state machines
- Extended FSMs: FSMs with variables (e.g. counters, channels)
- Communicating FSMs: how to describe interaction between FSMs?
- Reactive modules: Extended FSMs with a rich communication mechanism