CSEP590 – Model Checking and Software Verification

University of Washington

Department of Computer Science and Engineering

Summer 2003

Administration

- Instructor
  - David Richardson (daverich@cs)
    - Office: Sieg C112
    - Office hours: After class on Wed.
- TA
  - Evan Wellbourne (evan@cs)
    - Office: ??
    - Office hours: TBD

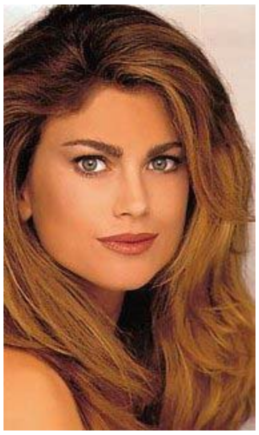
Administration(2)

- Class Time
  - Wednesday, 6:30-9:20pm
  - EE1-037 (this may change…)
  - Format
    - 80 min lecture, 20 min break, 80 min lecture, 10 min open questions.
- Web: http://www.cs.washington.edu/csep590
- Be sure to signup for the csep590 mailing list (see web for details)

Course Work

- Your grade will be based on the following
  - Weekly homework assignments (some may be biweekly and more project-based)
  - Final Exam (last day of class)
  - Other?? A few paper reviews

What is Model Checking?



This kind of model??

Unfortunately, no….

What is Model Checking?(2)

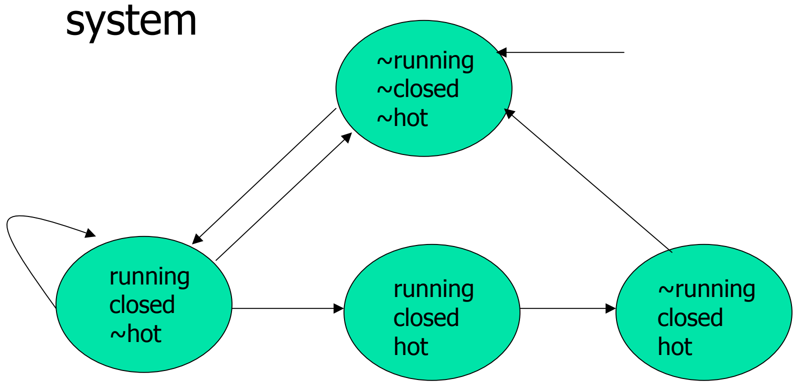
- Model checking is an automatic verification technique for finite state concurrent systems.
  - Set of components that execute together
- Developed independently by Clarke and Emerson and by Queille and Sifakis in early 1980's.
- Protocols (digital circuits, more recently software) modeled as state-transition systems.
- Specifications are a formula f in propositional temporal logic.
- Verification procedure: exhaustive (but efficient) search of the state space of the design to see if model satisfies f.

A Small Example

- Consider a system: simple microwave oven
  - States of the system correspond to values of 3 boolean variables:
    - Either door is closed or not closed
    - Either microwave is running or it is stopped
    - Either the food in the microwave is warm or it is cold

A Small Example(2)

- Model microwave as a simple transition system



A Small Example(3)

- Using Temporal Logic, one can say
  - Specification: microwave doesn't heat the food up until the door is closed
  - => ~hot holds until closed
  - Formula f = (~hot) U closed
- Given f and model, model checking can return whether or not the model satisfies f
- If not, a counterexample is returned, showing a path of execution whereby the system fails to satisfy the formula

A Small Example(4)

- Clearly, this example is too basic to be of any use
- However, the general idea remains the same

Advantages of Model Checking

- No proofs (as with theorem provers or proof checkers)
- Procedure is completely automatic.
- Fast (linear in size of model and in size of specification)
- Counterexamples
- Partial specifications allowed
- Logic is very expressive: allows for easy modeling of real-world protocols

Disadvantages of Model Checking

- State explosion: if modeled system has many components that can transition in parallel.
  - => number of states can grow exponentially with number of processes (size of system)
- Data paths
  - Variables in the model can take on a potentially infinite number of values

Can this problem be fixed?

- Much work has been done recently
  - 1987: Ken McMillan developed a symbolic model checking approach where the system was represented using Binary Decision Diagrams
    - Data structure for representing boolean functions
    - Concise representations for transition systems, fast manipulation
    - Good for synchronous systems
  - Partial Order Reduction: reduce number of states that must be explicitly enumerated
    - Good for asynchronous systems
- Other techniques (we'll see some later in course)

Today's Model Checkers

- Can handle systems with between 100 and 300 state variables
- Systems with 10^{120} reachable states have been checked!
- Using appropriate abstraction techniques, systems with an essentially unbounded number of states can be checked

A Brief History of Automatic Verification

- Goal: automatic verification of systems
- In the beginning….there were just input-output systems
  - Correctness: partial correctness + termination
  - Semantics: input-output relation
  - Specification language: propositional logic

History(2)

- In the late 1960's – Reactive systems
  - Don't compute anything
  - React to user input, don't terminate (event loop)
    - Termination can be bad! - Deadlock
  - Correctness: safety + progress + fairness +…
  - Semantics: Kripke Structures, transition systems (~automata)
  - Specification language: temporal logic

History(3)

- Temporal logic
  - Formalized in early 20th Century
  - Primitives: always, sometimes, until, since…
  - 1977: Pnueli decides to use temporal logic as a specification language
    - System satisfying a property corresponds to Kripke structure being a model of temporal formula

History(4)

- How automate?
  - Given a reactive system S and a temporal formula f, give an algorithm to determine if S satisfies f.
  - Late 1970's, early 1980's: reduced to proof systems
    - Give a proof system for checking validity in the logic
    - Extract from S a set of formulas F
    - Prove that F $\rightarrow$ f is valid using proof system
    - Doesn't work, too expensive.

History(5)

- Early 1980's: reduction to model checking problem
  - Construct Kripke structure K of S
  - Check if K is a model of f
- As we saw, the problem is state explosion (but people are making it better all the time)

History(6)

- 1990's – present
  - Industrial applications
  - Success in hardware verification
  - Groups in all major companies (IBM, Lucent, Intel, Microsoft, Motorola, Siemens…)
  - Many commercial and non-commercial tools
  - Extensions into software systems!! (holy grail)
- As leading professionals in top industries, this topic should hopefully be interesting to you ☺

History(7)

- A few success stories
  - 1992 – SMV system at CMU used to verify the IEEE Future+ cache coherence protocol
    - Found actual errors in an IEEE standard!
  - 1995 – Concurrency Workbench analyzed active structural control system to make buildings more resistent to earthquakes
    - Timing error found that could cause controller to worsen, NOT dampen vibrations experienced during an earthquake
  - And there are many, many others for hardware and protocol verification

Software Verification

- Why is this so freaking hard??
  - Data
  - Asynchronous behavior
  - Hmmm, this smells a lot like the halting problem….?
- Nonetheless, we'll examine it in the course

Software Verification(2)

- What is being done?
  - Use partial order reduction to reduce the number of states that are generated
    - Used by VeriSoft
    - Applications to Java
  - Use static analysis to extract a finite state synchronization skeleton from the program, model check the result
    - Bandera – Kansas State
    - Java PathFinder – NASA Ames
    - Slam Project (Bebop) - Microsoft