

CS6910: Testing/Verification of Concurrent Programs

Introduction to a different approach: Model Checking

Traditional Approaches Not Sufficient

- Testing
 - Finds many bugs early on, but after some time bugs are harder to find
 - Have I found all bugs in my system when testing doesn't find any new bugs anymore?
 - Requires test harness and drivers: Manual effort
 - Not scalable for concurrent programs
 - How to control scheduling?
 - ...

Any alternative approach?

- Model checking has been used successfully in hardware
- Hardware design bear similarity with concurrent software
 - Very complex
 - Inherently concurrent
 - Designed by high level language
- Inventors of model checking won 2008 Turing award

What is Model Checking?

- Model checking is the process of exhaustively checking that system behavior is correct
 - Fully automated
 - Static analysis
 - Does not require any test vectors
 - Systematic and exhaustive analysis
 - Provides error trace for debugging

Tool View of a Model Checker

System

\models
satisfies

Property (Specification)

```

    graph LR
      subgraph Inputs
        M[SW/HW model] --> MC
        P[property] --> MC
      end
      subgraph MC [Model Checker (Formal Verification Tool)]
      end
      subgraph Outputs
        MC --> C[correct]
        MC --> ET[error-trace]
        MC --> DK[don't know]
      end
    
```

- **“Correct”**: All behaviors are checked and correct.
- **“Error-trace”**: A step-by-step explanation of the bug.
- **“Don't know”**: Tool ran out of time, or out of memory; but so far, no bug was found.

Model Checker

System

\models
satisfies

Property (Specification)

- How do we describe the system?
 - Finite-state models
 - Formally: Kripke Structures
- How do we specify properties of interest?
 - Temporal Logics

Finite-State Machines (FSM)

- Set of states S
 - $S = \{s_0, s_1, s_2, s_3\}$
- Set of initial states
 - $S_0 \subseteq S, S_0 = \{s_0\}$
- Set of transitions
 - $R \subseteq S \times S$
 - $R = \{ (s_0, s_1), (s_0, s_2), (s_1, s_2), (s_2, s_0), (s_3, s_2) \}$

Kripke Structures

- Representation: (S, S_0, R, L) over atomic propositions AP
- Set of states S
- Set of initial states $S_0 \subseteq S$
- Set of transitions $R \subseteq S \times S$
- Labeling function $L: S \rightarrow 2^{AP}$

L labels each state with the set of "atomic propositions" that are true in that state.

Concurrent System Model

- Typically, an overall system is specified as a collection of modules and the environment of the system
- Each module is modeled as an automaton
- There are two ways of constructing overall system model
 - Synchronous composition
 - Asynchronous composition

Synchronous Product

- Often used in modeling hardware
- At each step, all modules proceed in lock-step
- Given two structures $M_i = (S_i, S_{i0}, R_i)$, the synchronous product is defined as $M = (S, S_0, R)$ using
 - $S = S_1 \times S_2$
 - $S_0 = S_{10} \times S_{20}$
 - $R((s_1, s_2), (t_1, t_2)) \text{ iff } R_1(s_1, t_1) \wedge R_2(s_2, t_2)$

Synchronous Model Example

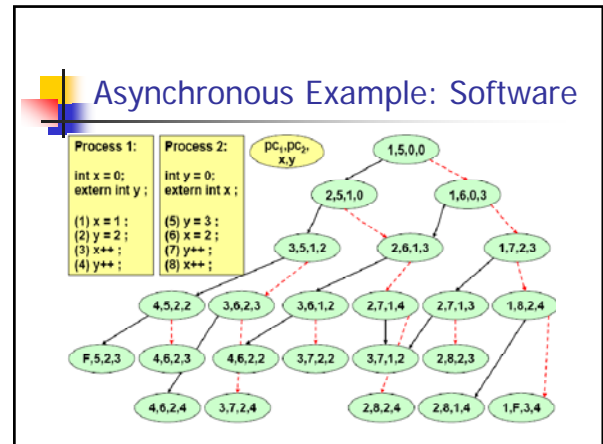
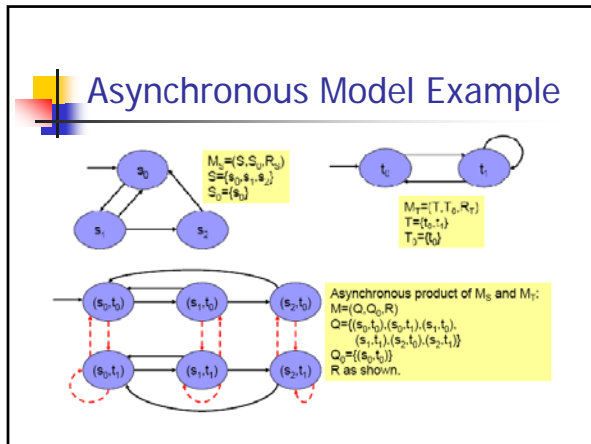
$M_0 = (S_0, C_0, R_0)$
 $S_0 = \{s_0, s_1, s_2\}$
 $S_{00} = \{s_0\}$

$M_1 = (T_0, T_0, R_1)$
 $T = \{t_0, t_1\}$
 $T_0 = \{t_0\}$

Synchronous product of M_0 and M_1
 $M = (Q, Q_0, R)$
 $Q = \{(s_0, t_0), (s_1, t_0), (s_2, t_0), (s_0, t_1), (s_1, t_1), (s_2, t_1)\}$
 $Q_0 = \{(s_0, t_0)\}$
 R as shown.

Asynchronous Product

- Often used to model software (interleaved model)
- At each time step, one module is chosen randomly, which can proceed a single step
- Given two structures $M_i = (S_i, S_{i0}, R_i)$, the asynchronous product is defined as $M = (S, S_0, R)$ using
 - $S = S_1 \times S_2$
 - $S_0 = S_{10} \times S_{20}$
 - $R((s_1, s_2), (t_1, t_2)) \text{ iff } [R_1(s_1, t_1) \wedge s_2 = t_2] \vee [R_2(s_2, t_2) \wedge s_1 = t_1]$



- ### Safety Property
- Safety property
 - "Something bad must not happen"
 - E.g.: System should not crash
 - E.g.: Mutual exclusive use of a shared resource

- ### Liveness Property
- Liveness property
 - "Something good must happen"
 - E.g.: Every packet sent must be received at its destination
 - E.g.: A bus resource allocator will eventually grant the use of the bus

- ### Propositional Logic
- Used to reason about static situations
 - Formulas are built using atomic propositions and propositional operators
 - Atomic proposition $p \in AP$
 - Negation $\neg p$
 - Conjunction $p \wedge q$
 - Disjunction $p \vee q$
 - Implication $p \rightarrow q$

