Formal Verification of Digital Systems
Part II: Model Checking

Dr. Gitanjali Swamy

Boston Advanced Development Labs

Mentor Graphics Corporation

Mentor Graphics
The Power to Create

Model Checking

- Problem: Is what I designed what I wanted ?
- Model checking solution:
  (Clarke, Emerson, Sistla 86) (Clarke,Emerson82)
  Specify design as an FSM/automaton, and the property (requirements) in a temporal logic.

- Model checking engine: Traverse the design FSM (recursively) marking states that satisfy the sub-parts of the property. Use these in turn to mark states satisfying the property.

TEMPORAL LOGICS

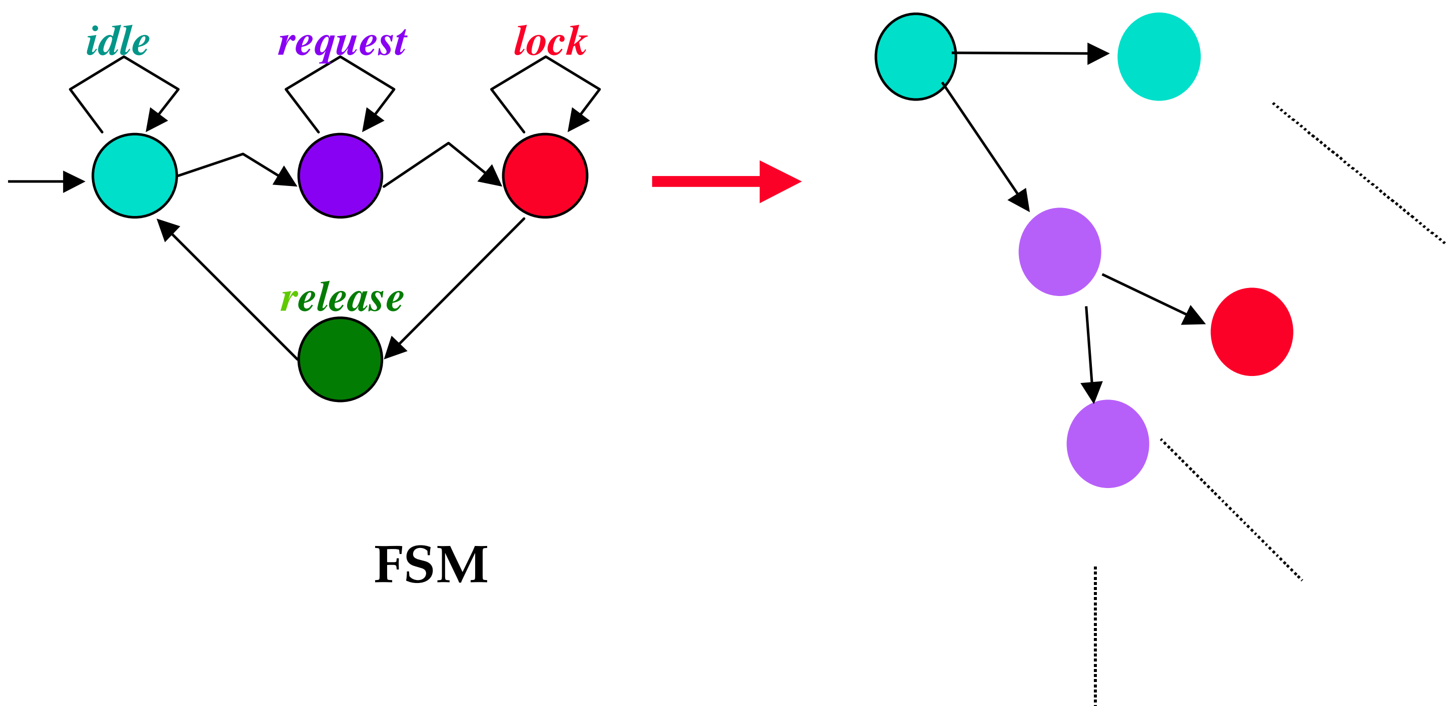
- Classification across a number of axes:
- propositional vs. first-order
- branching time vs. linear
- discrete time vs. continuous
- points vs. intervals
- past vs. future tense
- Concentrate on propositional, discrete, point-based,branching time, future-tense logic

Branching Temporal Logic

- The structure of time is an infinite tree
- Along each path in the tree, the corresponding timeline is isomorphic to N (natural numbers)
- Each node has at least one successor and can have infinitely many successors

Computational Tree Logic

- Finite State Machine (FSM) unrolled to Tree:



FSM

CTL: Computational Tree Logic

- Path Quantifier:
  - A: For All Paths
  - E: Exists A Path
- Temporal Operators:
  - G: Globally
  - F: Finally
  - X: Next State
  - U: Until

CTL

- CTL is defined recursively:
  - All atomic propostitions are formula
  - if p,q are formulae, so are:
    - »EGp, E(pUq),EXp, p+q, not(p)
    - » A(pUq), AXp, AGp, EFp,AFp

- Temporal Structure includes a labelling which associates each state with the set of atomic propositions that are true in that state

Properties in CTL

- It is never the case that two processors simultaneously *lock* the printer (Mutual Exclusion).

  AG !((P1.state = lock) *(P2.state = lock))

- If a processor requests the printer, it eventually gets it (Responsiveness)

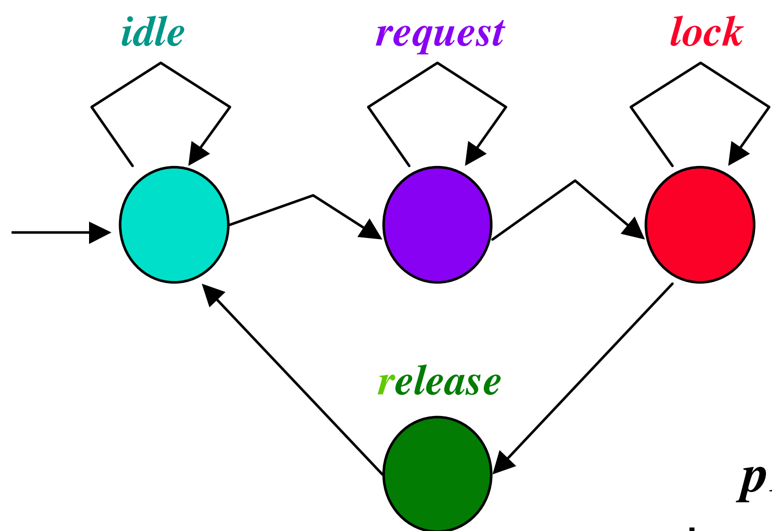
  AG (P1.state = request -> AF(P1.state = lock))

Model Checking Algorithm

- Recursive: Case f:
  - EXp: return $\exists_x T(x,y).p(x)$
  - p+q: return p(x)+q(x)
  - not(p) : return not(p(x))
  - E(pUq) : return LFP(f(Q(x), q(x)), where f(Q(x)) = p(x) + q(x). EX(Q(x))
  - EGp: return GFP(f(Q), p(x)), where f(Q(x)) = Q(x). EX(Q(x))

CTL Model Checking

- P1



$$p1{=}lock \models p1{=}lock$$

$$p1{=}lock \models AF(p1{=}lock)$$

$$0 \models (p1{=}request) \rightarrow AF(p1{=}lock)$$

$$0 \models AG((p1{=}request) \rightarrow AF(p1{=}lock))$$

Fair CTL Model Checking

- Fair CTL = CTL + Fairness.

Complexity

- Complexity:
  - CTL: Linear in size of formula.
  - CTL*: Exponential in length of formula, but linear in size of S.

Which is Better: MC or LC ?

- The Jury is Out !
- Property expressible as CTL not as Buchi: Resetability, i.e. the design can reach a reset state.
- Property expressible as Buchi not as CTL: Some event occurs at odd cycles alone.

Example: Verifying Properties

vis> read_verilog arbiter.v

vis> initialize_verification

vis> model_check arbiter.ctl

#MC: formula passed --- **AG !(P1.state = lock * P2.state = lock)**

#MC: formula failed --- **AG (P1.state = request -> AF (P1.state = lock))**

#MC Error trace ---

initial state:

C0.haveToken:TRUE

C1.haveToken:FALSE

C2:haveToken:FALSE

P1.state:idle

goes to

C0.haveToken:FALSE

C1.haveToken:TRUE

C2:haveToken:FALSE

P1.state:idle

....

Other Aspects

- Binary Decision Diagram data structure.
- Partitioned Transition Relations: How do I efficiently handle systems of interacting FSMs?
- Abstraction: Can I reduce the complexity of representation ?
- Debugging: What witnesses to failure should I return ?

Conclusions

- Still in its infancy: FSM based verification of medium sized industrial designs
  - Can handle 400-500 registers.
  - With abstraction the number may go as high as 5000 registers.
- Compare this to combinational equivalence checking which can handle up to a million gate designs.