# ESE535: Electronic Design Automation

Day 13: March 4, 2009

FSM Equivalence Checking

## Today

- Sequential Verification
  - FSM equivalence
  - Issues
    - Extracting STG
    - Valid state reduction
    - Incomplete Specification

## Motivation

- Write at two levels
  - Java prototype and VHDL implementation
  - VHDL specification and gate-level implementation
- Write at high level and synthesize/optimize
  - Want to verify that synthesis/transforms did not introduce an error

## Cornerstone Result

- Given two FSM's, can test their equivalence in finite time
- *N.B.:*
  - Can visit all states in a FSM with finite input strings
    - No longer than number of states
    - Any string longer must have visited some state more than once (by pigeon-hole principle)
    - Cannot distinguish any prefix longer than number of states from some shorter prefix which eliminates cycle (pumping lemma)

## FSM Equivalence

- Given same sequence of inputs
  - Returns same sequence of outputs
- Observation means can reason about finite sequence prefixes and extend to infinite sequences which FSMs are defined over

## Equivalence

- Brute Force:
  - Generate all strings of length |state|
    - (for larger FSM = most states)
  - Feed to both FSMs with these strings
  - Observe any differences?
- How many such strings?
  - $|\text{Alphabet}|^{\text{states}}$

## Smarter

- Create composite FSM
  - Start with both FSMs
  - Connect common inputs together (Feed both FSMs)
  - XOR together outputs of two FSMs
    - Xor's will be 1 if they disagree, 0 otherwise
- Ask if the new machine ever generate a 1 on an xor output (signal disagreement)
  - Anything it accepts is a proof of non-equivalence
  - Accepts nothing → equivalent

## Creating Composite FSM

- Assume know start state for each FSM
- Each state in composite is labeled by the pair $\{S1_i, S2_j\}$
  - At most product of states
- Start in $\{S1_0, S2_0\}$
- For each symbol *a*, create a new edge:
  - $T(a,\{S1_0, S2_0\}) \rightarrow \{S1_i, S2_j\}$
    - If $T_1(a, S1_0) \rightarrow S1_i$, and $T_2(a, S2_0) \rightarrow S2_j$
- Repeat for each composite state reached

## Composite DFA

- At most |alphabet|*|State1|*|State2| edges == work
- Can group together original edges
  - *i.e.* in each state compute intersections of outgoing edges
  - Really at most $|E_1|*|E_2|$

## Acceptance

- State $\{S1_i, S2_j\}$ is an accepting state iff
  - On some input, State $S1_i$ and $S2_j$ produce different outputs
- If $S1_i$ and $S2_j$ have the same outputs for all composite states, it is impossible to distinguish the machines
  - They are equivalent
- A reachable state with differing outputs
  - Implies the machines are not identical

## Empty Language

- Now that we have a composite state machine, with this acceptance
- **Question**: does this composite state machine accept anything?
  - Is there a reachable state has differing outputs?

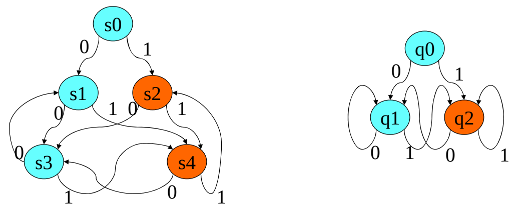
## Answering Empty Language

- Start at composite start state $\{S1_0, S2_0\}$
- Search for path to an Accepting state
- Use any search (BFS, DFS)
- End when find accepting state
  - Not equivalent
- OR when have explored entire reachable graph w/out finding
  - Are equivalent

## Reachability Search

- Worst: explore all edges at most once
  - $O(|E|)=O(|E_1|*|E_2|)$
- Should be able to combine composition construction and search
  - *i.e.* only follow edges which fill-in as search

## Example



● = accept state

## Issues to Address

- Obtaining State Transition Graph from Logic
- Incompletely specified FSM?
- Know valid (possible) states?

## Getting STG from Logic

- Brute Force
  - For each state
    - For each input minterm
      - Simulate/compute output
      - Add edges
  - Compute set of states will transition to
- Smarter
  - Exploit cube grouping, search pruning
    - Cover sets of inputs together
  - Coming attraction: PODEM

## Incomplete State Specification

- Add edge for unspecified transition to
  - Single, new, terminal state
- Reachability of this state may indicate problem
  - Actually, if both transition to this new state for same cases
    - Might say are equivalent
    - Just need to distinguish one machine in this state and other not

## Valid States

- Composite state construction and reachability further show what’s reachable
- So, end up finding set of valid states
  - Not all possible states from state bits

## Summary

- Finite state means
  - Can test with finite input strings
- Composition
  - Turn it into a question about a single FSM
- Reachability
  - Allows us to use poly-time search on FSM to **prove** equivalence

## Admin

- Next Week: Spring Break
- Next Lecture: Monday March 16
- Reading: Hardcopy handout today

## Big Ideas

- Equivalence
  - Same observable behavior
  - Internal implementation irrelevant
    - Number/organization of states, encoding of state bits…
- Exploit structure
  - Finite DFA … necessity of reconvergent paths
  - Pruning Search – group together cubes
  - Limit to valid/reachable states
- Proving invariants vs. empirical verification