ESE535:
Electronic Design Automation

Day 21: April 13, 2015
FSM Equivalence Checking

Penn

Today

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

Behavioral
(C, MATLAB, …)
↓ Arch. Select, Schedule
RTL
↓ FSM assign, Two-level, Multilevel opt., Covering, Retiming
Gate Netlist
↓ Placement, Routing
Layout
↓
Masks

FSM Equivalence

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

---

Question

- Given a state machine with N states:
- How long of an input sequence do I need to visit any of the N states?
  - (i.e. if someone picks a state, how long of an input sequence might you need to select a path to that state?)

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 = the one with the most states)
  - Feed to both FSMs with these strings
  - Observe any differences?
- How many such strings?
  - $|\text{Alphabet}|^{\text{states}}$

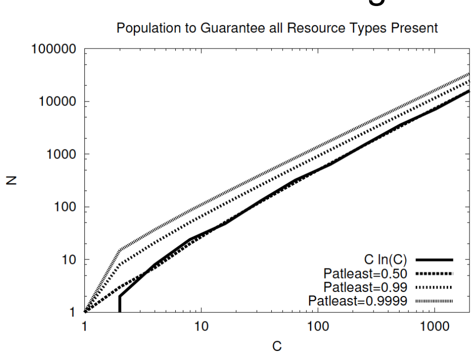
Random Testing

- What does this say about random testing?
- P(generate string)=$1/|\text{alphabet}|^{|\text{states}|}$
- P(generate string)=$|\text{alphabet}|^{-|\text{states}|}$
- P(miss string) = 1-P(generate string)
- P(miss string, n tests)=P(miss string)^n
- P(gen str, n test)=$1-(1-|\text{alphabet}|^{-|\text{states}|})^n$

Random Testing

- Instance of "Coupon Collector" Problem
  - If there are C unique "Coupons" that can be selected uniformly at random
  - How many coupons will a collector need to get to have a full set of C?
- Need C ln (C) to have a 50% chance of a full set

Random Testing



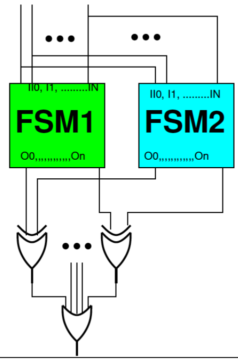
[DeHon, LLNSD Chapter, Kluwer 2004]

Random Testing

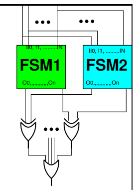
- Random testing
  - Powerful
  - Not an efficient way to guarantee finds all behaviors
- How can we do better?

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

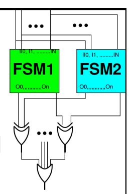


Smarter



- 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)
  - Any 1 is a proof of non-equivalence
  - Never produce a 1 → equivalent

Creating Composite FSM



- Assume know start state for each FSM
- Each state in composite is labeled by the pair $\{S1_i, S2_j\}$
  - How many such states?
  - Compare to number of strings of length #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 FSM

- How much work?

  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 $|E_1|*|E_2|$

Non-Equivalence

- State $\{S1_i, S2_j\}$ demonstrates non-equivalence iff
  - $\{S1_i, S2_j\}$ reachable
  - 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 construction
- **Question**: does this composite state machine ever produce a 1?
  - Is there a reachable state that has differing outputs?

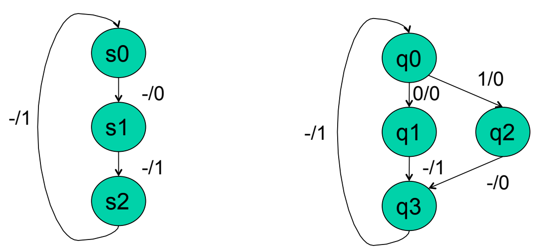
Answering Empty Language

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

Reachability Search

- Worst: explore all edges at most once
  - $O(|E|)=O(|E_1|*|E_2|)$
- When we know the start states, we can combine composition construction and search
  - *i.e.* only follow edges which fill-in as search
  - (way described)

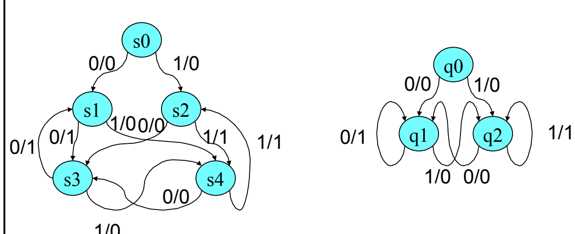
Example



Creating Composite FSM

- Assume know start state for each FSM
- Each state in composite is labeled by the pair $\{S1_i, S2_j\}$
- 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 $T_2(a, S2_0) \rightarrow S2_j$
    - Check that both state machines produce same outputs on input symbol *a*
- Repeat for each composite state reached

Example



Issues to Address

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

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

Start State?

- Worst-case:
  - Try verifying for all possible start state pairs
  - Identify start state pairs that lead to equivalence
    - Candidate start pairs
- More likely have one (specification) where know start state
  - Only need to test with all possible start states for the other FSM

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
    - Or find differentiating input sequence

Big Ideas

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

Admin

- Reading for next two lectures on blackboard