

CS137: Electronic Design Automation

Day 9: February 10, 2006
FSM Equivalence Checking



CALTECH CS137 Winter2006 -- DeHon

Today

- Sequential Verification
 - DFA equivalence
 - Issues
 - Extracting STG
 - Valid state reduction
 - Incomplete Specification
 - Solutions
 - State PODEM
 - State/path exploration

2

CALTECH CS137 Winter2006 -- DeHon

Cornerstone Result

- Given two DFA's, can test their equivalence in finite time
- *N.B.:*
 - Can visit all states in a DFA 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)

3

CALTECH CS137 Winter2006 -- DeHon

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 DFAs (FSMs) are defined over

4

CALTECH CS137 Winter2006 -- DeHon

Equivalence

- Brute Force:
 - Generate all strings of length $|state|$
 - (for larger DFA)
 - Feed to both DFAs
 - Observe any differences?
 - $|Alphabet|^{|states|}$

5

CALTECH CS137 Winter2006 -- DeHon

Smarter

- Create composite DFA
- XOR together acceptance of two DFAs in each composite state
- Ask if the new machine accepts anything
 - Anything it accepts is a proof of non-equivalence
 - Accepts nothing \rightarrow equivalent

6

CALTECH CS137 Winter2006 -- DeHon

Composite DFA

- Assume know start state for each DFA
- 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

CALTECH CS137 Winter2006 -- DeHon

7

Composite DFA

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

CALTECH CS137 Winter2006 -- DeHon

8

Acceptance

- State $\{S1_i, S2_j\}$ is an accepting state iff
 - State $S1_i$ accepts and $S2_j$ does not accept
 - State $S1_i$ does not accept and $S2_j$ accepts
- If $S1_i$ and $S2_j$ have the same acceptance for all composite states, it is impossible to distinguish the machines
 - They are equivalent
- A state with differing acceptance
 - Implies a string which is accepted by one machine but not the other

CALTECH CS137 Winter2006 -- DeHon

9

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 which accepts the input?

CALTECH CS137 Winter2006 -- DeHon

10

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

CALTECH CS137 Winter2006 -- DeHon

11

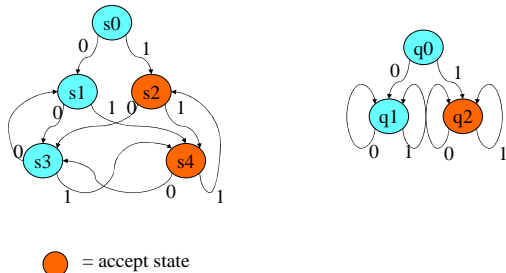
Reachability Search

- Worst: explore all edges at most once
 - $O(|E|) = O(|E_1|*|E_2|)$
- Actually, should be able to find during composite construction
 - If only follow edges, fill-in as search

CALTECH CS137 Winter2006 -- DeHon

12

Example



CALTECH CS137 Winter2006 -- DeHon

13

Issues to Address

- Get State-Transition Graph from
 - RTL, Logic
- Incompletely specified FSM?
- Know valid (possible) states?
- Know start State for Logic?
- Computing the composite FSM may be large

CALTECH CS137 Winter2006 -- DeHon

14

Getting STG Verilog/VHDL

- Gather up logic to **wait** statement
 - Make one state
- Split states (add edges) on **if/else**, **select**
- Backedges with **while/for**
 - Branching edges on loop conditions
- Start state is first state at beginning of code.

CALTECH CS137 Winter2006 -- DeHon

15

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
 - Use modified PODEM to justify outputs and next state
 - Exploit cube grouping, search pruning

CALTECH CS137 Winter2006 -- DeHon

16

PODEM state extraction

- Search for all reachable states
 - Don't stop once find one output
 - Keep enumerating and generating possible outputs

CALTECH CS137 Winter2006 -- DeHon

17

CS137a: Day6

Delay Computation

- Modification of a testing routine
 - used to justify an output value for a circuit
- PODEM
 - backtracking search to find a suitable input vector associated with some target output
 - Simply a branching search with implication pruning
 - Heuristic for smart variable ordering

CALTECH CS137 Winter2006 -- DeHon

18

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

CALTECH CS137 Winter2006 -- DeHon

19

Valid States

- PODEM justification finds set of possibly reachable 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

CALTECH CS137 Winter2006 -- DeHon

20

Start State for Logic

- Start states should output same thing between two FSMs
- Start search with state set $\{S1_0, S2_i\}$ for all $S2_i$ with same output as $S1_0$
- Use these for acceptance (contradiction) reachability search

CALTECH CS137 Winter2006 -- DeHon

21

Memory?

- Concern for size of search space
 - Product set of states
 - Nodes in search space
- Combine
 - Generation
 - Reachability
 - State justification/enumeration

CALTECH CS137 Winter2006 -- DeHon

22

Composite Algorithm

- PathEnumerate(st, path, ValStates)
 - // st is a state of M1
 - ValStates+=st
 - While !(st.enumerated)
 - Edge=EnumerateStateFanout(st) // PODEM
 - Simulate Edge on M2
 - Equivalent result? If not return(FAIL)
 - If (Edge.FaninState(M1), Edge.FaninState(M2) in Path.Spairs)
 - Return(PATH_OK) ;; already visisted/expanded that state
 - Else
 - ValStates+=Edge.FaninState(M1)
 - Path=Path+Edge; Update Path.Spairs
 - PathEnumerate(Edge.FaninState(M1), Path, ValStates)3

CALTECH CS137 Winter2006 -- DeHon

Start Composite Algorithm

- PathEnumerate(Start(M1), empty, empty)
- Succeed if complete path search and not fail
 - Not encounter contradiction

CALTECH CS137 Winter2006 -- DeHon

24

Admin

- Class Monday: Processor Verification
- No Class Wednesday
- Friday: 10:30am—noon + 1:30—3pm
 - Proposed Plan of Attack
 - Sequential baseline also due

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