Equivalence Checking of Sequential Circuits

Implicit State Enumeration, Image Computations, Traversal of Product Machines

Priyank Kalla

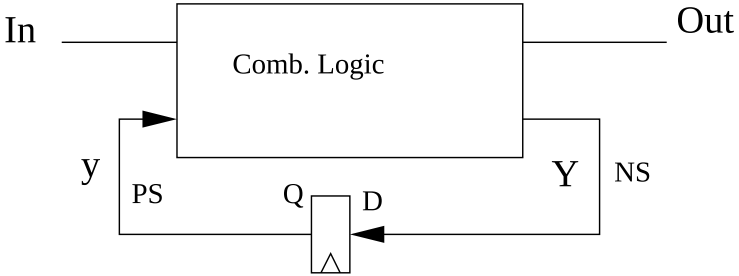
Associate Professor
Electrical and Computer Engineering, University of Utah
kalla@ece.utah.edu
`http://www.ece.utah.edu/~kalla`

Dec 1-3, 2014

The FSM Model

Define a Mealy Machine as an *n*-tuple:

- $\mathcal{M} = (\sum, O, S, S^0, \Delta, \Lambda)$
- $\sum$: Input label; O: output label
- S: Set of States; $S^0 \subset S$: set of initial states
- $\Delta : S \times \sum \rightarrow S$: Next State Transition function
- $\Lambda : S \times \sum \rightarrow O$: Output function



FSM Equivalence

Two Machines, $\mathcal{M}_1$ and $\mathcal{M}_2$, are equivalent if

- They are identical; or
- They have identical states but different encoding; or
- $\mathcal{M}_1 \subseteq \mathcal{M}_2$ or vice-versa; or
- They have different reachable states but same distinguishable states (same condition as above); or
- Different unreachable states, and unreachable states are a *don't care* condition

Prove that two machines (sequential circuits) produce the same output response on application of all possible input **sequences**

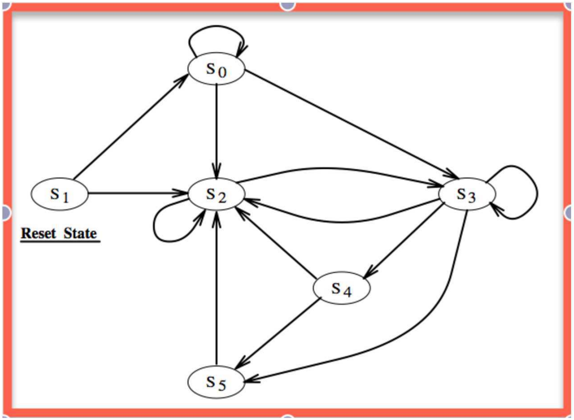
Approach

- Build a product machine, and traverse the product machine
- Machine traversal is the core computational engine
- At every step of traversal, see observe of the output responses are the same
- Explicit traversal (DFS) is infeasible
- We use Implicit State Enumeration (BFS traversal) based on Boolean formulas
- Implementation is BDD based (can also use Gröbner)
- First, we will study Implicit State Enumeration, then apply it to product machine

BFS Traversal Algorithm

```
**Input**: Transition functions $\Delta$, initial state $S^0$
$from^0 = reached = S^0$;
$i = 0$;
repeat
    $i \leftarrow i + 1$;
    $to^i \leftarrow \mathrm{Img}(\Delta, from^{i-1})$;
    $new^i \leftarrow to^i \cap \overline{reached}$;
    $reached \leftarrow reached \cup new^i$;
    $from^i \leftarrow new^i$;
until $new^i == 0$;
return $reached$
```

- The main computation: $to^i \leftarrow \mathrm{Img}(\Delta, from^{i-1})$
- $\mathrm{Img}(\Delta, from^{i-1})$: The "forward" image of the set $from^{i-1}$ under the transition function Δ
- Let us apply this to a FSM


s_0
s_1
s_2
s_3
s_4
s_5
Reset State

Algorithm run on the FSM

$from^0 = reached = S_1$. At iteration $i = 1$:

- $to^1 \leftarrow Img(\Delta, S_1) = \{S_0, S_2\}$
- $new^1 = \{S_0, S_2\} \cap \{S_0, S_2, \ldots S_5\} = \{S_0, S_2\}$
- $reached = \{S_1, S_0, S_2\}$, and $from^1 = \{S_0, S_2\}$

At iteration $i = 2$ and $i = 3$:

- $to^2 = \{S_0, S_2, S_3\}, new^2 = \{S_3\} = from^2, reached = \{S_1, S_0, S_2, S_3\}$
- $to^3 = \{S_3, S_4, S_5\}, new^3 = \{S_4, S_5\} = from^3, reached = \{S_0, \ldots, S_5\}$
- $to^4 = \{S_2, S_5\}, new^4 = \emptyset$, algorithm terminates!

BFS traversal without computing *new*^{i} states

```
**Input**: Transition functions $\Delta$, initial state $S^0$
$from^0 = reached^0 = S^0$;
$i = 0$;
**repeat**
    $i \leftarrow i + 1$;
    $to^i \leftarrow \mathrm{Img}(\Delta, from^{i-1})$;
    $reached^i \leftarrow reached^{i-1} \cup to^i$;
    $from^i \leftarrow reached^i$;
**until** $reached^i == reached^{i-1}$;
**return** $reached^i$
```

This basic approach can be easily implemented using the Gröbner Basis algorithm, details follow in the next few slides.

Traverse a circuit?

If a sequential circuit is given:

- The **transition function** Δ is given by Boolean equations of the flip-flops of the circuit: $t_i = \Delta_i(s, x)$
  - t_i is the next state variable, s represents the present state variables and x represents the input variables
- The **Transition relation of the FSM**: $T(s, x, t) = \prod_{i=1}^{n} (t_i \overline{\oplus} \Delta_i)$
  - n is the number of flip flops
  - $\overline{\oplus}$ is XNOR operation
- Let $C(s)$ denote the set of initial (or current) states
- Image $\text{Img}(\Delta, C) = \exists_s \exists x [T(s, x, t) \cdot C(s)]$
  - Remember Smoothing: $\exists_x f = f_x \vee f_{\overline{x}}$
  - $\exists_x f$ is the smallest function that contains f and makes f independent of x

It is not that exotic....

Image of a function

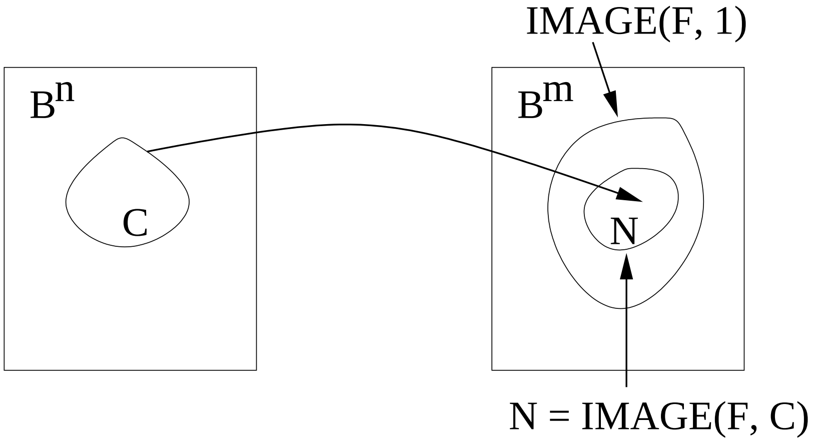
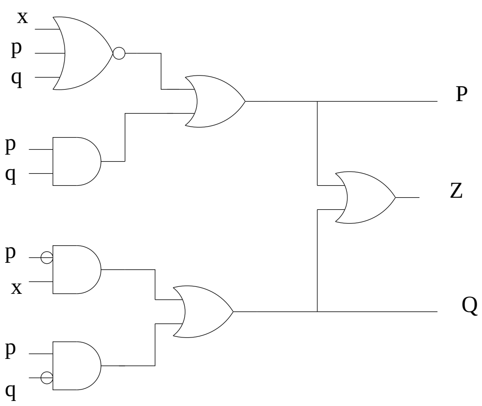


Image Computation on a Circuit



P = x'p'q' + pq
Q = p'x + pq'

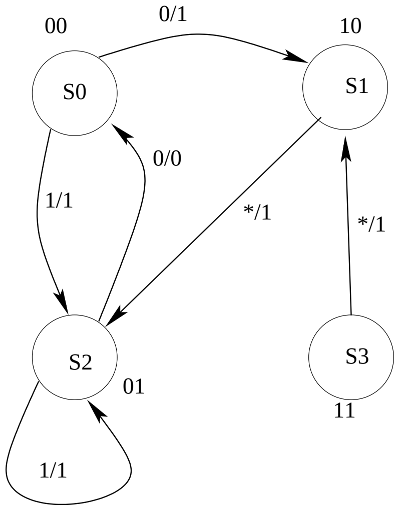


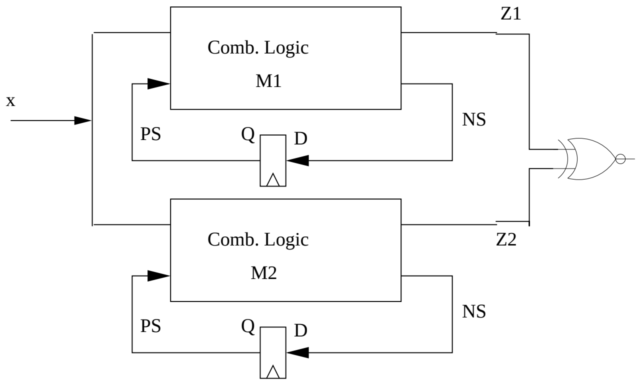
Image Computation

- Init State: $C(s) = p'q' = \{00\}$
- $t_1 \overline{\oplus} \Delta_1 = P \overline{\oplus} (x'p'q' + pq)$
- $t_2 \overline{\oplus} \Delta_2 = Q \overline{\oplus} (xp' + pq')$
- $T(s, x, t) = [P \overline{\oplus} (x'p'q' + pq)] \wedge [Q \overline{\oplus} (xp' + pq')]$
- Starting from initial state 00, what is the **set** of next states?

$$\begin{aligned}
to^1 = Img(\Delta, from^0 = C(s)) &= \exists p, q, x \;\; T(p, q, x, P, Q) \wedge C(s) \\
&= PQ' + P'Q \\
new^1 &= to^1 \cap \overline{reached} \\
&= PQ' + P'Q \\
reached = reached \cup new &= P'Q' + PQ' + P'Q = P' + Q'
\end{aligned}$$

In the next iteration: $C(s) = from^1 = new^1 = PQ' + P'Q$, continue...

BFS Traversal is the core of Sequential Verification



Given $\mathcal{M}_1 = (\sum, O, S^1, S_0^1, \Delta_1, \Lambda_1)$; $\mathcal{M}_2 = (\sum, O, S^2, S_0^2, \Delta_2, \Lambda_2)$

- Build a product machine $\mathcal{M}_1 \times \mathcal{M}_2 = (\sum, O^{12}, S^{12}, S_0^{12}, \Delta_{12}, \Lambda_{12})$

Product Machine

Given $\mathcal{M}_1 = (\sum, O, S^1, S^1_0, \Delta_1, \Lambda_1)$; $\mathcal{M}_2 = (\sum, O, S^2, S^2_0, \Delta_2, \Lambda_2)$

- Build a product machine $\mathcal{M}_1 \times \mathcal{M}_2 = (\sum, O^{12}, S^{12}, S^{12}_0, \Delta_{12}, \Lambda^{12})$, where:
- $s^{12} = (s^1, s^2) \in S^{12}$: Concatenation of states
- $$\begin{aligned} \Delta_{12} \equiv \Delta_{12}(s^{12}, x) :& (S^1 \times S^2) \times \sum \rightarrow (S^1 \times S^2) \\ &\Longrightarrow ((S^1 \times \sum) \rightarrow S^1) \wedge ((S^2 \times \sum) \rightarrow S^2) \\ &\Longrightarrow (\Delta_1(s^1, x), \Delta_2(s^2, x)) \end{aligned}$$
- $z^{12} = \Lambda_{12}(s^{12}, x) : (S^1 \times S^2) \times \sum \rightarrow \{0, 1\}$, where:
- $\Lambda_{12}(s^{12}, x) = \begin{cases} 1 & \text{if } \Lambda_1 = \Lambda 2 \\ 0 & \text{otherwise} \end{cases}$

In other words...

The product machine has:

- States that are concatenation of the states of the original machine
- Transition relation is also a concatenation (or conjunction) of the transition relations of the original machines

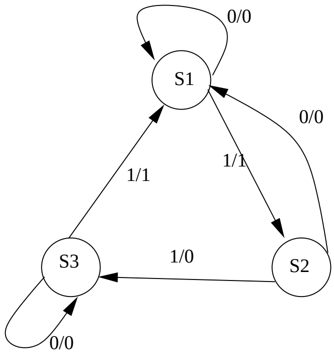
Verification Problem

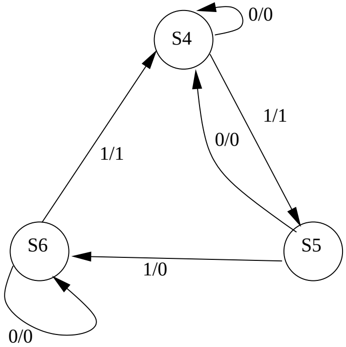
Find a sequence of inputs that distinguishes the initial states of the individual machines.

Verification Approach

Compute Next States independently and concatenate the results to form the next state of the product. The outputs specified for the two corresponding transitions are then compared. If equal, then the product machine outputs a 1, otherwise 0 (BUG!)

FSM Verify Example





Build the Product Machine.....

FSM_VERIFY

```
Input: $\sum, \Delta_1, \Delta_2, \Lambda_1, \Lambda_2$, initial states $S_0^1, S_0^2$
$from^0 = reached = new^0 = (S_0^1, S_0^2)$; $i = 0$;
repeat
    $i \leftarrow i + 1$;
    $to^i \leftarrow \mathrm{Img}(\Delta_{12}, from^{i-1})$;
    $new^i \leftarrow to^i \cap \overline{reached}$;
    for each $S^{12} \in new^i$ do
        for each $x \in \sum$ do
            if $\Lambda_{12}(s, x) = 0$ then
                return FALSE;
            end
        end
    end
    $from^i \leftarrow new^i$;
    $reached \leftarrow reached \cup new^i$;
until $new^i \neq 0$;
```