Sequential Circuits: Verification

Virendra Singh
Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

CS-226: Digital Logic Design

Lecture 26: 06 April 2021

CADSL

Finite State Machine

$M(I, O, S, S_0, \delta, \lambda)$

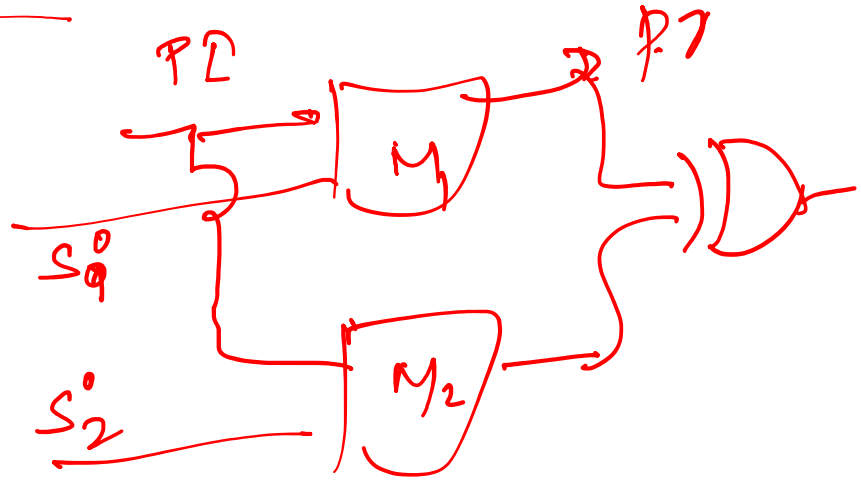
$n_1 = |S_1| =$

$M_1 \; (I, O, S_1, S_1^0, \delta_1, \lambda_1)$

$n_2 = |S_2| =$

$M_2 \; (I, O, S_2, S_2^0, \delta_2, \lambda_2)$

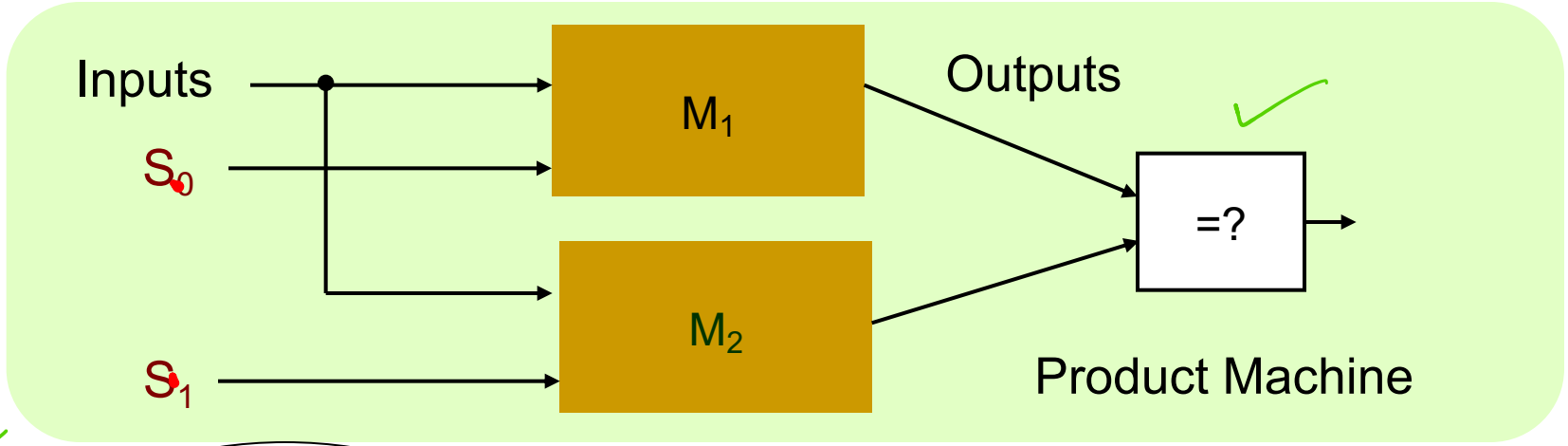
$M_1 \stackrel{?}{=} M_2$

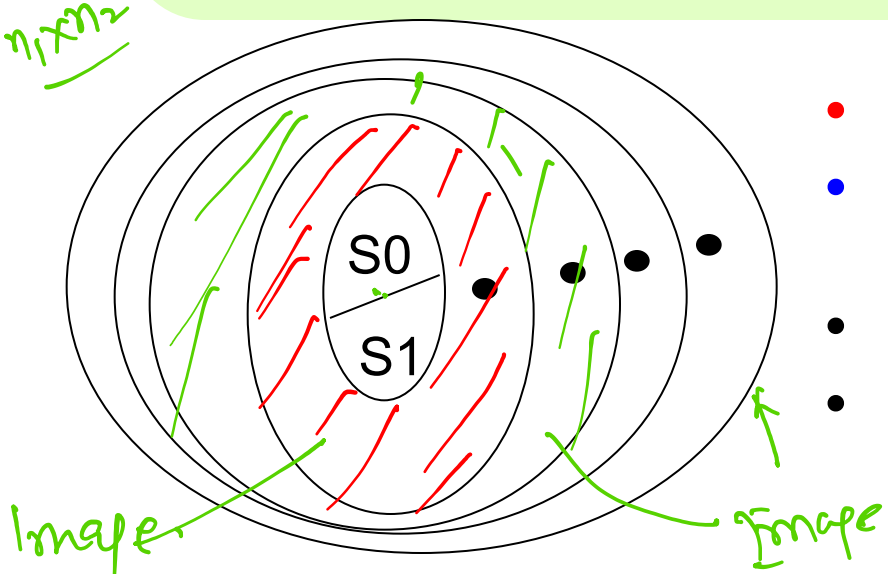


$n_1 \times n_2$.

Reachability-Based Equivalence Checking

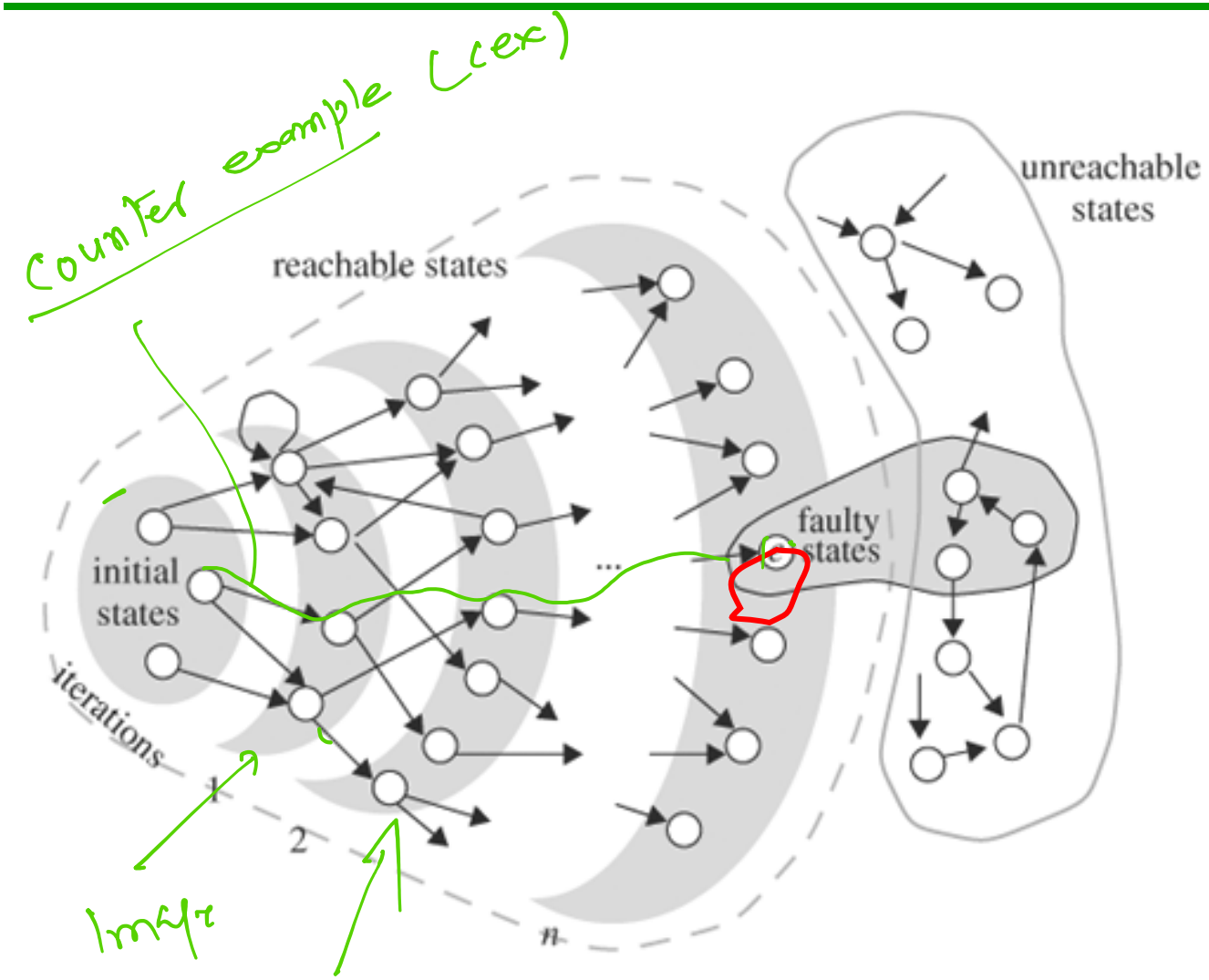
Approach 3: Symbolic Traversal Based Reachability Analysis





- Build product machine of M_1 and M_2
- Traverse state-space of product machine starting from reset states (S_0, S_1)
- Test equivalence of outputs in each state
- Can use any state-space traversal technique

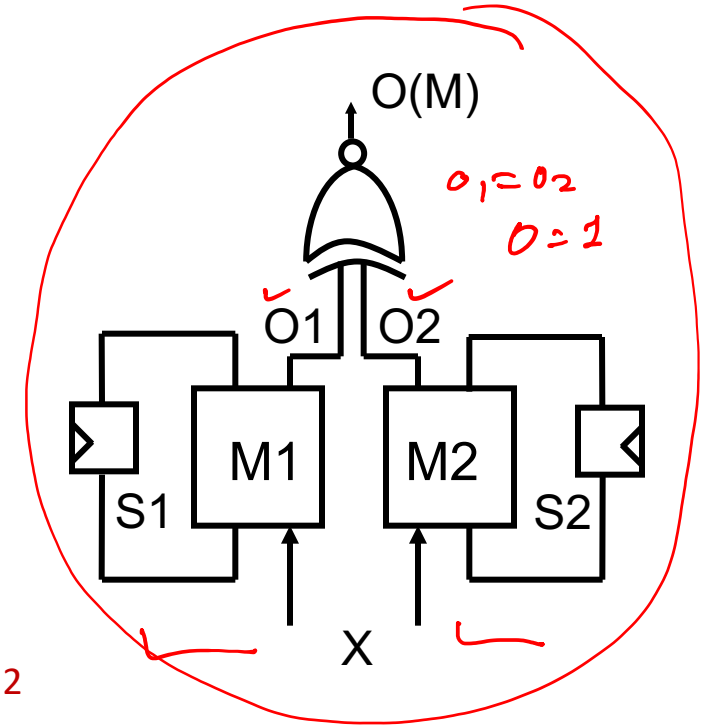
Forward Reachability



Sequential Verification

- Symbolic FSM traversal of the product machine

- Given two FSMs: $M_1(X,S_1, \delta_1, \lambda_1,O_1)$, $M_2(X,S_2, \delta_2, \lambda_2,O_2)$
- Create a product FSM: $M = M_1 \mathbf{x} M_2$
  - traverse the states of M and check its output for each transition
  - the output O(M) =1, if outputs $O_1 = O_2$
  - if all outputs of M are 1, M_1 and M_2 are *equivalent*
  - otherwise, an *error state* is reached
  - *error trace* is produced to show: $M_1 \neq M_2$

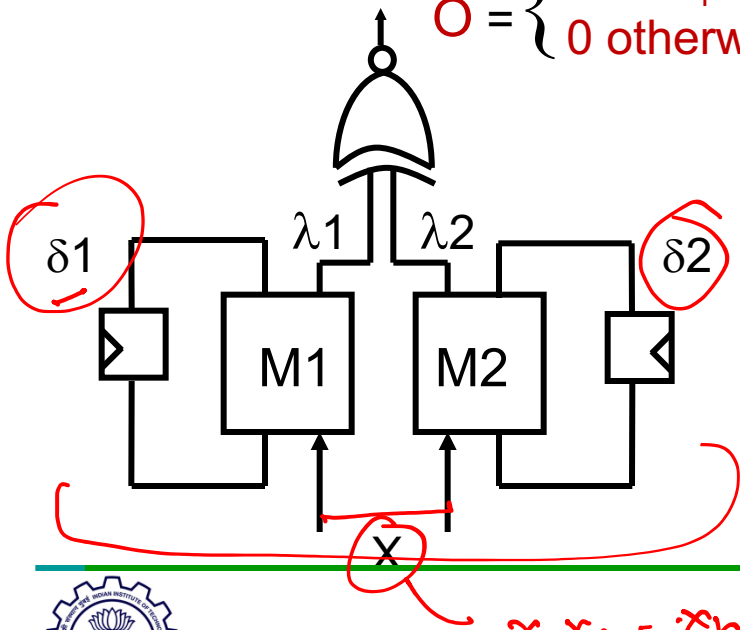


Product Machine - Construction

- Define the product machine $M(X, S, S^0, \delta, \lambda, O)$
  - states, $S = S_1 \times S_2$
  - next state function, $\delta(s, x) : (S_1 \times S_2) \times X \rightarrow (S_1 \times S_2)$
  - output function, $\lambda(s, x) : (S_1 \times S_2) \times X \rightarrow \{0,1\}$

$$O = \begin{cases} 1 & \text{if } O_1 = O_2 \\ 0 & \text{otherwise} \end{cases}$$

$$\lambda(s,x) = \lambda_1(s_1,x) \overline{\oplus} \lambda_2(s_2,x)$$



- Error trace (*distinguishing sequence*) that leads to an error state
  - sequence of inputs which produces 1 at the output of M
  - produces a state in M for which M1 and M2 give different outputs

FSM Traversal - Algorithm

- Traverse the product machine $M(X,S,S_0,\delta,\lambda,O)$
  - start at an initial state S_0
  - iteratively compute symbolic image $Img(S_0,R)$ (set of *next states*):

$$\mathrm{Im}\, g(S_0,R) = \exists_x \exists_s S_0(s).R(x,s,t)$$

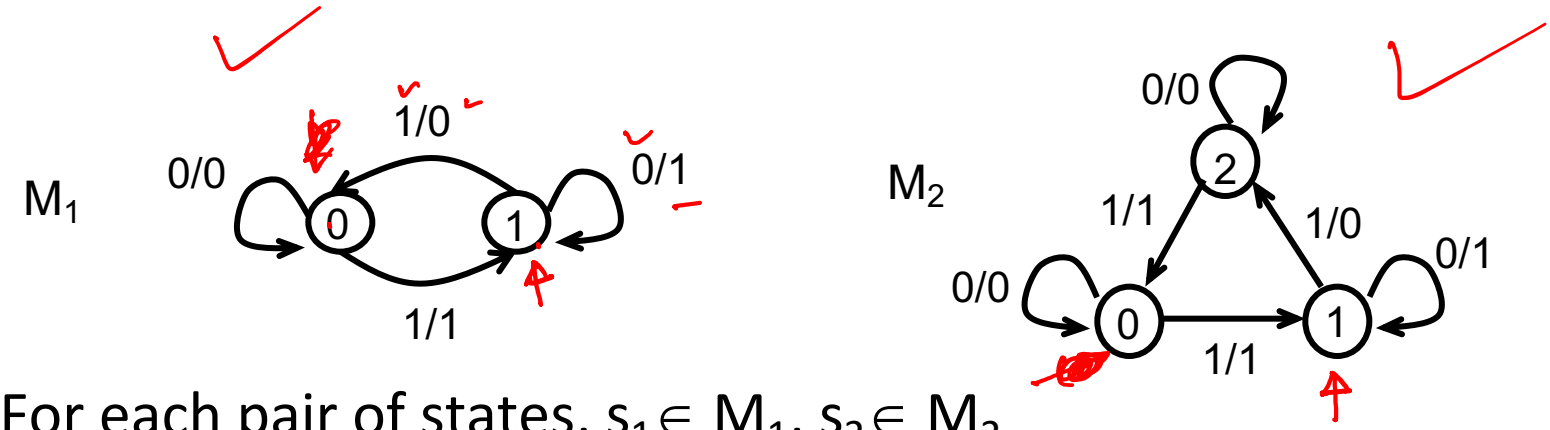
$$R = \prod_i R_i = \prod_i (t_i \equiv \delta_i(s,x))$$

until an *error state* is reached

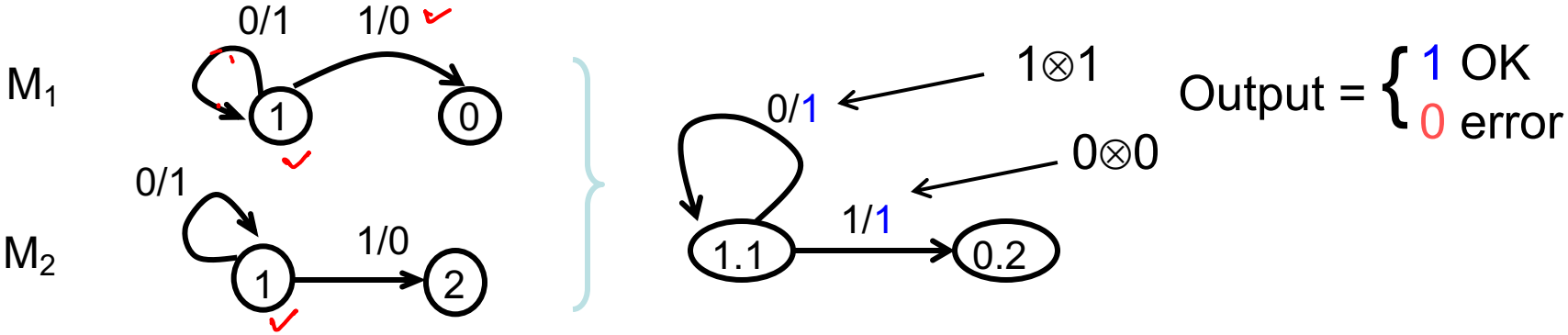
  - transition relation R_i for each next state variable t_i can be computed as $t_i = (t \otimes \delta(s,x))$

(this is an alternative way to compute transition relation, when design is specified at gate level)

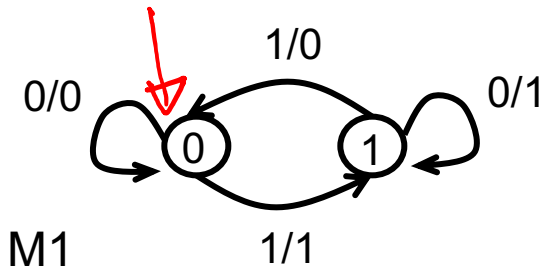
Construction of the Product FSM

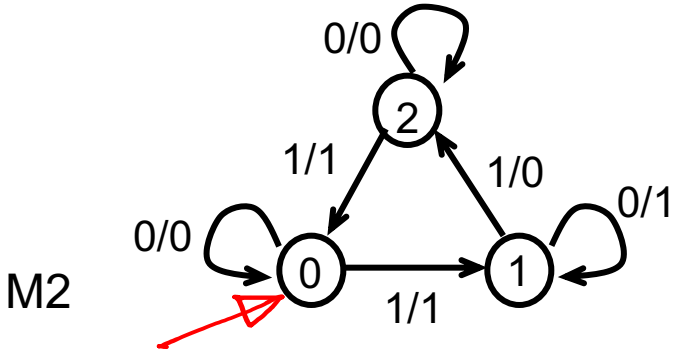


- For each pair of states, $s_1 \in M_1$, $s_2 \in M_2$
  - create a combined state $s = (s_1, s_2)$ of M
  - create transitions out of this state to other states of M
  - label the transitions (*input/output*) accordingly



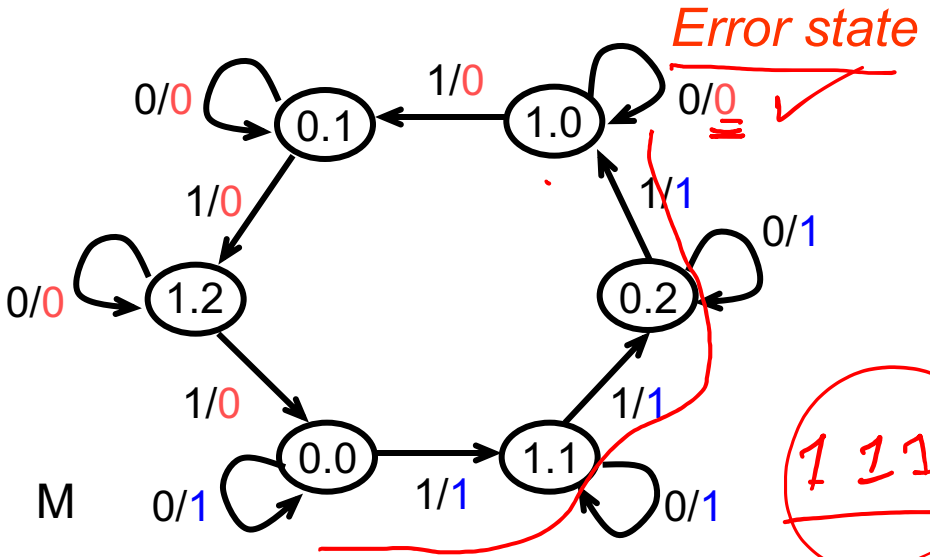
FSM Traversal in Action





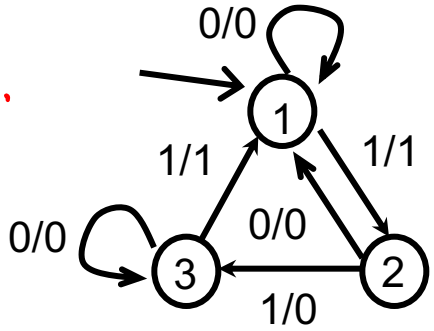
Initiall states: s_1=0, s_2=0, s=(0.0)

State reached	Out(M) x=0	x=1
• *New* ^{0} = (0.0)	1	1
• *New* ^{1} = (1.1)	1	1
• *New* ^{2} = (0.2)	1	1
• *New* ^{3} = (1.0)	0	0

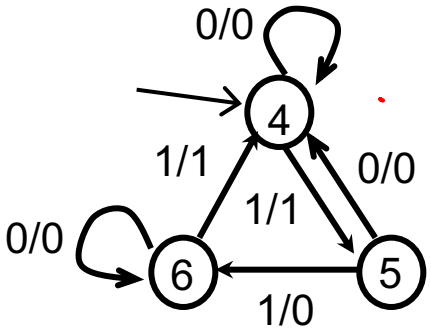


- STOP - backtrack to initial state to get *error trace:* x={1,1,1,0}

FSM Traversal in Action



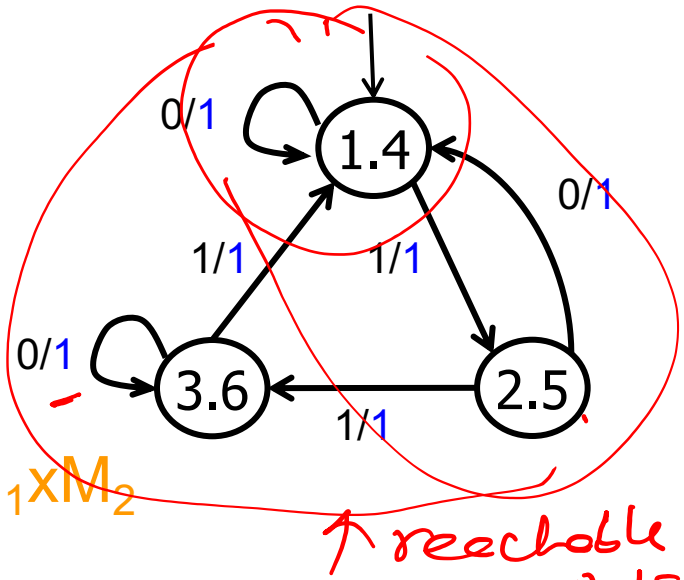
M_1



M_2

Initiall states: s_1=1, s_2=4, s=(1.4)

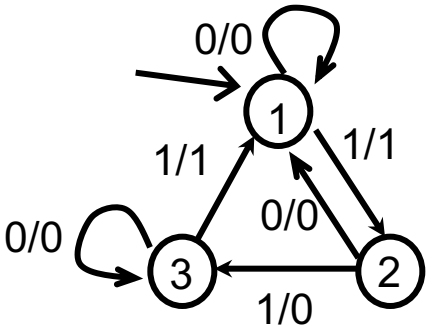
State reached	Out(M) x=0	x=1
• *New*^{0} = (1.4)	1	1
• *New*^{1} = (2.5)	1	1
• *New*^{2} = (3.6)	1	1



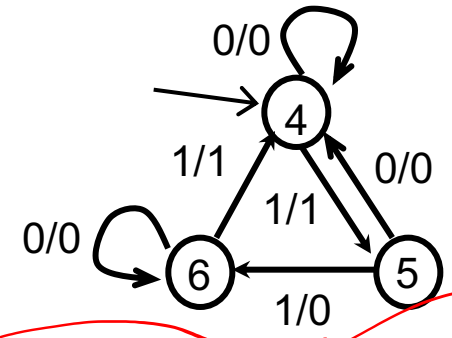
$M = M_1 x M_2$

- STOP: No new reachable state

FSM Traversal in Action



M_1

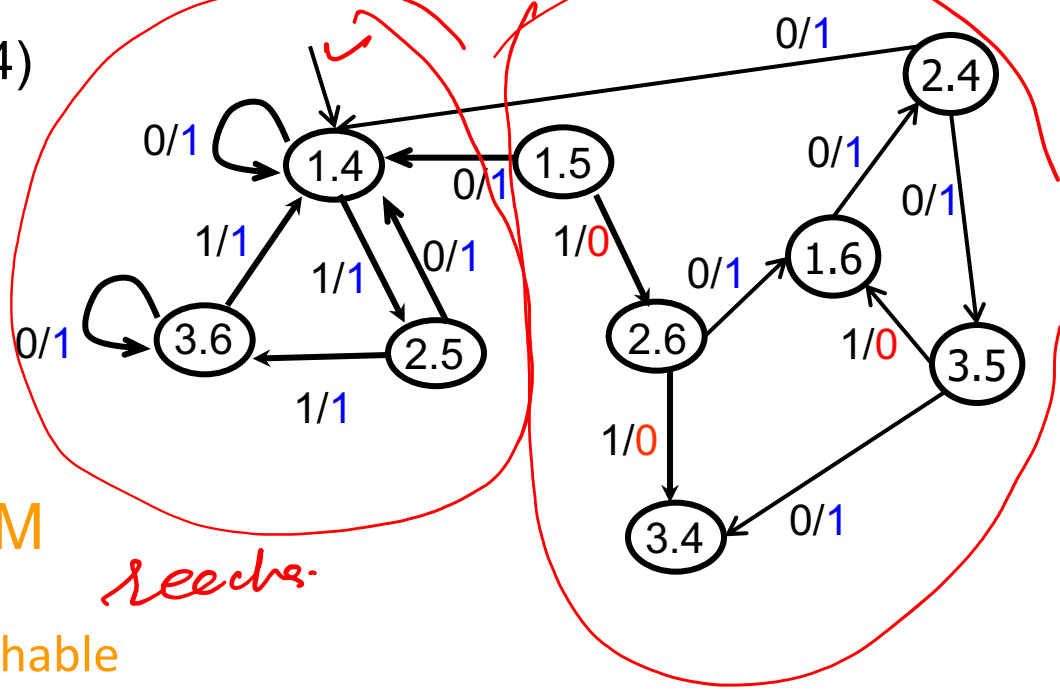


M_2

Unreachable

Initiall states: s_1=1, s_2=4, s=(1.4)

State reached	Out(M) x=0	x=1
• *New*^{0} = (1.4)	1	1
• *New*^{1} = (2.5)	1	1
• *New*^{2} = (3.6)	1	1



M

reach.

- Erroneous states are not reachable

FSM Traversal - Algorithm

- Traverse the product machine M(X,S,δ, λ,O)
  - start at an initial state S_0
  - iteratively compute symbolic image $Img(S_0,R)$ (set of *next states*):

$$\mathrm{Im}\, g(S_0, R) = \exists_x \exists_s S_0(s).R(x,s,t)$$

$$R = \prod_i R_i = \prod_i (t_i \equiv \delta_i(s,x))$$

until an *error state* is reached

  - transition relation R_i for each next state variable t_i can be computed as $t_i = (t \otimes \delta(s,x))$

(this is an alternative way to compute transition relation, when design is specified at gate level)

How to Represent States ?

- Not practical to represent individual states
- Represents set of states symbolically
- ROBDD encodes boolean functions
  - Code elements S
- Represent a subset T as boolean function f_T
- $f_T = \{0,1\}^n \rightarrow \{0.1\}$

How to Represent States ?

SET OF STATES.

{st0}, {st0, st1}, {st0, st1, st3}

~~st0 st1 s2 s3~~

st0, st1, st2, st3

00 01 10 11

explicit way

s_0	s_1	f	f
0	0	1	1
0	1	0	1
1	0	0	0
1	1	0	0

$s_0 s_1$ — 00, 01, 10, 11 — Variable.

$\bar{s}_0 \cdot \bar{s}_1$

ROBDD

ROBDD

$\bar{s}_0 \cdot \bar{s}_1 + \bar{s}_0 s_1 \Rightarrow \bar{s}_0$

{st0, st1}

{st0, st1, st3}

$\bar{s}_0 + s_0 s_1$

$\bar{s}_0 + s_1$

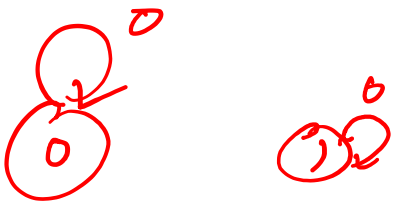
How to Represent States ?

- Computation uses symbolic BFS approach to all reachable states by shortest path
- Key step is image computation
  - Img($\delta(s,x)$, $C(s)$)
- BFS allows to deal multiple states simultaneously
- BDD is used to represent TF
- Let $t_i = \delta_i(s,x)$ i = 1,2,…n
- C(s) is a symbolic state set

How to Represent States ?

x	s	t	fr.
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	
1	0	1	
1	1	0	
1	1	1	

x, s, t



R (x, s, t)

↓

all possible transitions

ROBDD

$T(x,s,t) = R(x,s,t) \cdot C(s)$

$\lambda = \lambda_0 \lambda_1$

$= T(\lambda, s, t)$

How to Represent States ?

δ_1 $\quad$ δ_2

t_1, s_1 $\quad$ t_2, s_2

$t_1 = \delta_1(s_1, x)$ $\quad$ $t_2 = \delta_2(s_2, x)$

$R(x, s, t) = (t_1 \equiv \delta_1(s_1, x)) \cdot (t_2 \equiv \delta_2(s_2, x))$

$\prod_i (t_i \equiv \delta_i(s_i, x))$

How to Represent TF ?

- Given a deterministic transition function (s,x) the corresponding transition relation is defined by
  - $T(s,x,t) = \prod(t_i = \delta_i(s,x))$
- T(s,x,t) = 1 denotes a set of encoded tripples (s,x,t) , each representing a transition in the FST of a given FSM
- Straight forward to compute image
- Need new boolean operation
  - Existential Abstraction
  - $\exists_{xi} . f = f_{xi} + f_{\overline{xi}}$
  - f_{xi}- smallest (fewest minterm) function that contains all minterms of f and independent of xi

How to Represent TF ?

$f(x_1, x_2 \cdots x_i \cdots x_n) = x_i f_{x_i} + \bar{x}_i \cdot f_{\bar{x}_i}$

$f_{x_i} + f_{\bar{x}_i}$

$\exists_{x_i} \cdot f(x_1, x_2 \cdots x_i \cdots x_n) = f_{x_i} + f_{\bar{x}_i}$

$f = x_1 x_2 + \bar{x}_1 x_3$

$f_{x_1} = x_2 + 0 = x_2$

$f_{\bar{x}_1} = x_3$

$\exists_{x_1} f(x_1, x_2, x_3) = x_2 + x_3$

$R(x, s, t) \cdot C(s)$

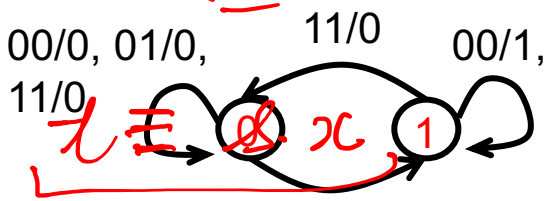
$f(x, s, t)$

How to Represent TF ?

- Given f(s,x) = f(s, .. s_n, x_1, ...x_m) the existential abstraction w.r.t a set of variables is defined as
  - $$\exists_x . f(s,x) = \exists_{x1}(\exists_{x2}(....\exists_{xm}(f(s,x)))$$

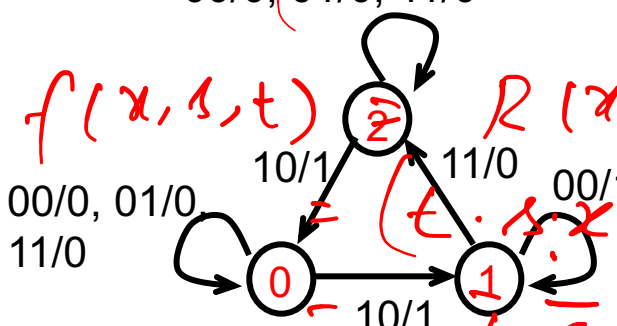
- **Procedure**
  - Compute TF, T(s,x,t)
  - Compute conjunction of T and C
  - Existentially abstract all s variable and all x variable - provides I(t)
  - I(t) is the smallest function independent of s and x which contains all the tripples in f(s,x,t)

State Reachability in Product FSM



M_1

- $t_1 = \delta^1_1 = x_1x_2' + s_1x'_1$
- $\lambda^1 = \delta^1_1$



M_2

- 0=00, 1=01, 2 =10
- $t_2 = \delta^2_2 = s_3x_1x_2 + s_2(x'_1 + x_2)$
- $t_3 = \delta^2_3 = s'_2x_1x'_2 + s_3x'_1$
- $\lambda^2 = s_3x'_1 + x_1x'_2$

$f = \bar{t} \cdot \bar{s}$

$\exists_s f = \bar{t}$

$f_s = 0$

$f_{\bar{s}} = \bar{t}$

next $\bar{t} = 1$ $t=0$

$\exists_t f = \bar{s}$ $\bar{s}=1 \Rightarrow s=0$

$f_t = 0$

$f_{\bar{t}} = \bar{s}$

Thank You