CH-8: Finite State Machine Optimization

Contemporary Logic Design

YONSEI UNIVERSITY

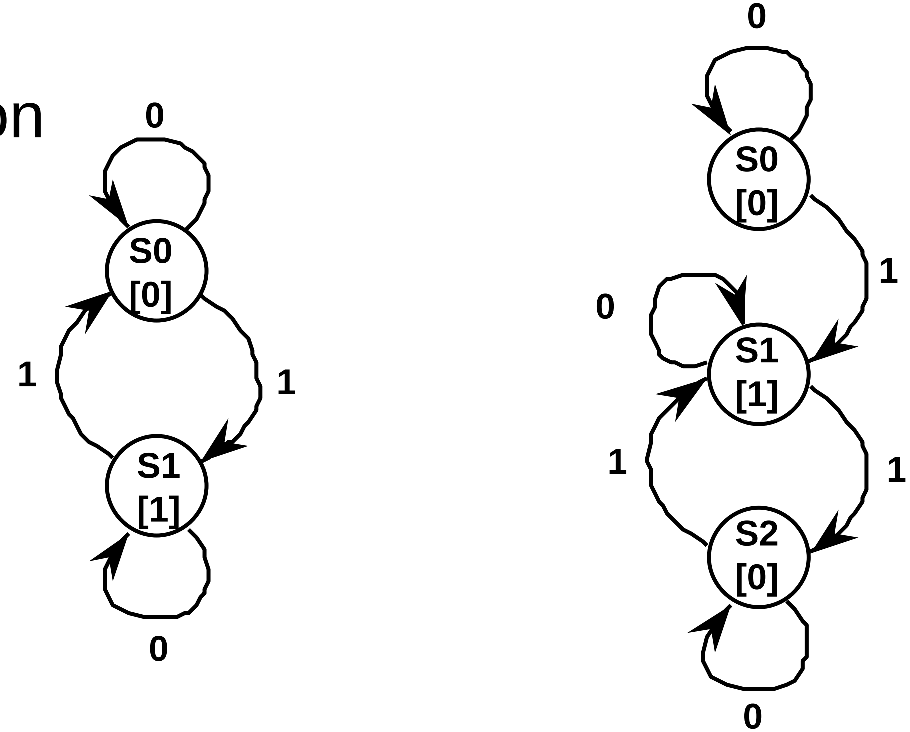
Fall 2016

Finite State Machine Optimization

- State minimization
  - *fewer states* require <u>fewer state bits</u>
  - *fewer bits* require <u>fewer logic equations</u>
- Encodings: state, inputs, outputs
  - state encoding with *fewer bits* has <u>*fewer equations*</u> to implement
    - however, each may be more complex
  - state encoding with *more bits* (e.g., one-hot) has *simpler equations*
    - complexity directly related to complexity of state diagram
  - input/output encoding may or may not be under designer control

Finite State Machine Optimization

- State minimization



- **Odd Parity Checker**: **two alternative state diagrams**
  - **Identical output behavior on all input strings**
  - **FSMs are *equivalent*, but require different implementations**
  - **Design state diagram without any concern for # of states, Reduce later**

Algorithmic Approach to State Minimization

- **Goal** – *identify and combine states* that have equivalent behavior
- *Equivalent states*:
  - same output
  - for all input combinations, states transition to same or equivalent states
- Algorithm sketch to find equivalent states
  - 1. place all states in one set
  - 2. initially partition set based on output behavior
  - 3. successively partition resulting subsets based on next state transitions
  - 4. repeat (3) until no further partitioning is required
    - states left in the same set are equivalent
  - polynomial time procedure

State Minimization: Row Matching Method

- **Sequence detector EX** for 1010 or 0110
  - single input X, output Z
  - taking inputs grouped four at a time, output 1 if last four inputs were the string 1010 or 0110

Example I/O Behavior:

1st bit

X = 0010 0110 1100 1010 0011 . . .
Z = 0000 0001 0000 0001 0000 . . .

Upper bound on FSM complexity:

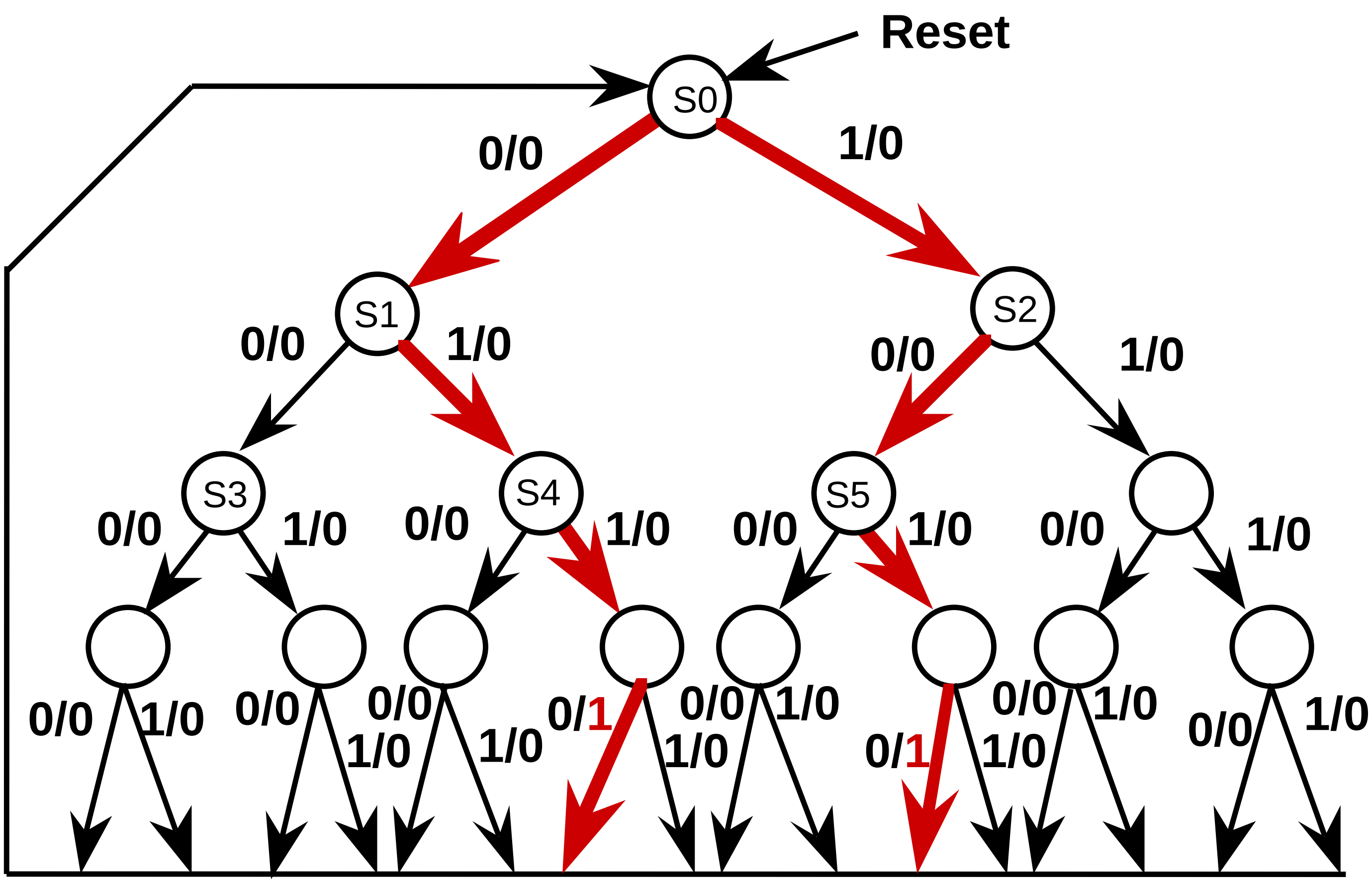
Fifteen states (1 + 2 + 4 + 8)

Thirty transitions (2 + 4 + 8 + 16)

sufficient to recognize any binary string of length four!

State Minimization: Row Matching Method

- **State diagram** for example FSM



State Minimization: Row Matching Method

- **Initial State Transition Table:**

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S_{10}	0	0
10	S_5	S_{11}	S_{12}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011	S_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
101	S_{12}	S_0	S_0	1	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

State Minimization: Row Matching Method

- **Initial State Transition Table:**

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S_{10}	0	0
10	S_5	S_{11}	S_{12}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011	S_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
101	S_{12}	S_0	S_0	1	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

State Minimization: Row Matching Method

- **Initial State Transition Table:**

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S'_{10}	0	0
10	S_5	S_{11}	S'_{10}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011 or 101	S'_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

State Minimization: Row Matching Method

- **Initial State Transition Table:**

		Next State		Output	
Input Sequence	Present State	X=0	X=1	X=0	X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S'_{10}	0	0
10	S_5	S_{11}	S'_{10}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011 or 101	S'_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

State Minimization: Row Matching Method

- **Initial State Transition Table:**

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	$\boxed{S'_7}$	$\boxed{S'_7}$	0	0
01	S_4	$\boxed{S'_7}$	S'_{10}	0	0
10	S_5	$\boxed{S'_7}$	S'_{10}	0	0
11	S_6	$\boxed{S'_7}$	$\boxed{S'_7}$	0	0
not (011 or 101)	S'_7	S_0	S_0	0	0
011 or 101	S'_{10}	S_0	S_0	1	0

State Minimization: Row Matching Method

- **Initial State Transition Table:**

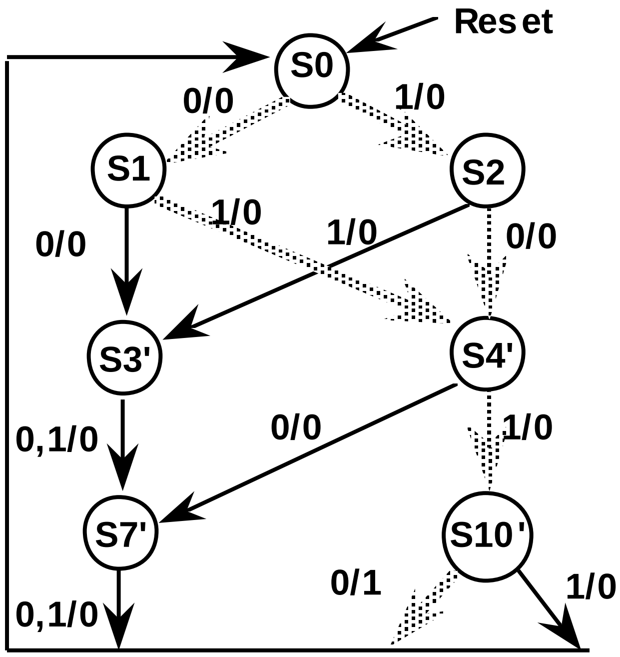
Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S'_7	S'_7	0	0
01	S_4	S'_7	S'_{10}	0	0
10	S_5	S'_7	S'_{10}	0	0
11	S_6	S'_7	S'_7	0	0
not (011 or 101)	S'_7	S_0	S_0	0	0
011 or 101	S'_{10}	S_0	S_0	1	0

State Minimization: Row Matching Method

Final Reduced State Transition Table

		Next State		Output	
Input Sequence	Present State	X=0	X=1	X=0	X=1
Reset	S0	S1	S2	0	0
0	S1	S3'	S4'	0	0
1	S2	S4'	S3'	0	0
00 or 11	S3'	S7'	S7'	0	0
01 or 10	S4'	S7'	S10'	0	0
not (011 or 101)	S7'	S0	S0	0	0
011 or 101	S10'	S0	S0	1	0

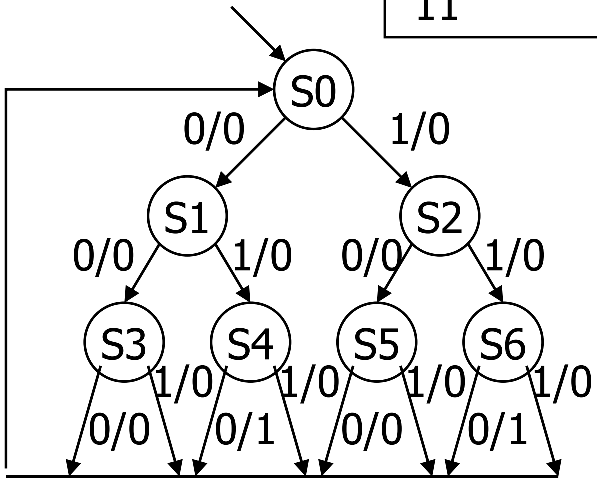
Corresponding State Diagram



State Minimization Example

- Sequence detector for 010 or 110

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0



Method of Successive Partitions

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0

(S0 S1 S2 S3 S4 S5 S6)

(S0 S1 S2 S3 S5) (S4 S6)

(S0 S3 S5) (S1 S2) (S4 S6)

(S0) (S3 S5) (S1 S2) (S4 S6)

S1 is equivalent to S2

S3 is equivalent to S5

S4 is equivalent to S6

Method of Successive Partitions

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4′	0	0
1	S2	S5	S4′	0	0
00	S3	S0	S0	0	0
01	S4′	S0	S0	1	0
10	S5	S0	S0	0	0

(S0 S1 S2 S3 S4 S5 S6)

(S0 S1 S2 S3 S5) (S4 S6)

(S0 S3 S5) (S1 S2) (S4 S6)

(S0) (S3 S5) (S1 S2) (S4 S6)

S1 is equivalent to S2

S3 is equivalent to S5

S4 is equivalent to S6

Method of Successive Partitions

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1	S3'	S4'	0	0
1	S2	S3'	S4'	0	0
00	S3'	S0	S0	0	0
01	S4'	S0	S0	1	0

(S0 S1 S2 S3 S4 S5 S6)

(S0 S1 S2 S3 S5) (S4 S6)

(S0 S3 S5) (S1 S2) (S4 S6)

(S0) (S3 S5) (S1 S2) (S4 S6)

S1 is equivalent to S2

S3 is equivalent to S5

S4 is equivalent to S6

Method of Successive Partitions

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1'	S3'	S4'	0	0
00	S3'	S0	S0	0	0
01	S4'	S0	S0	1	0

(S0 S1 S2 S3 S4 S5 S6)

(S0 S1 S2 S3 S5) (S4 S6)

(S0 S3 S5) (S1 S2) (S4 S6)

(S0) (S3 S5) (S1 S2) (S4 S6)

S1 is equivalent to S2

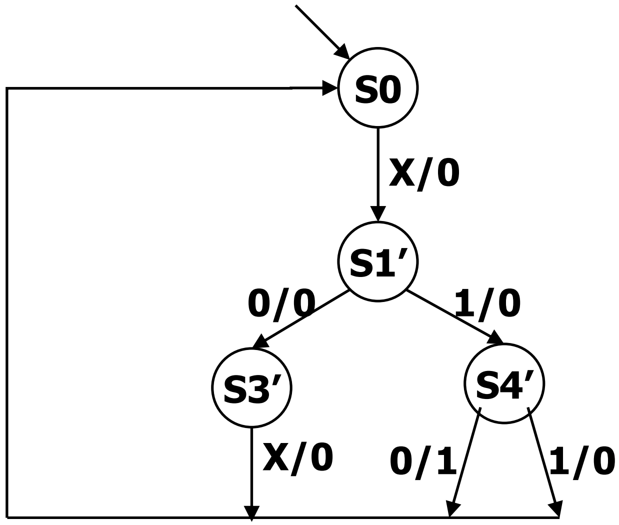
S3 is equivalent to S5

S4 is equivalent to S6

Minimized FSM

- State minimized sequence detector for 010 or 110

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1'	S1'	0	0
0 + 1	S1'	S3'	S4'	0	0
X0	S3'	S0	S0	0	0
X1	S4'	S0	S0	1	0

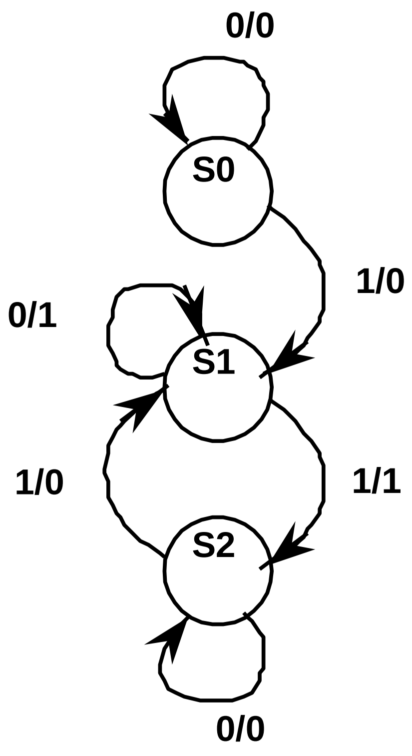


Row Matching Method

- Straightforward to understand and easy to implement
- Problem: does not yield the most reduced state table!

Example: 3 State Odd Parity Checker

Present State	Next State X=0	Next State X=1	Output
S_0	S_0	S_1	0
S_1	S_1	S_2	1
S_2	S_2	S_1	0



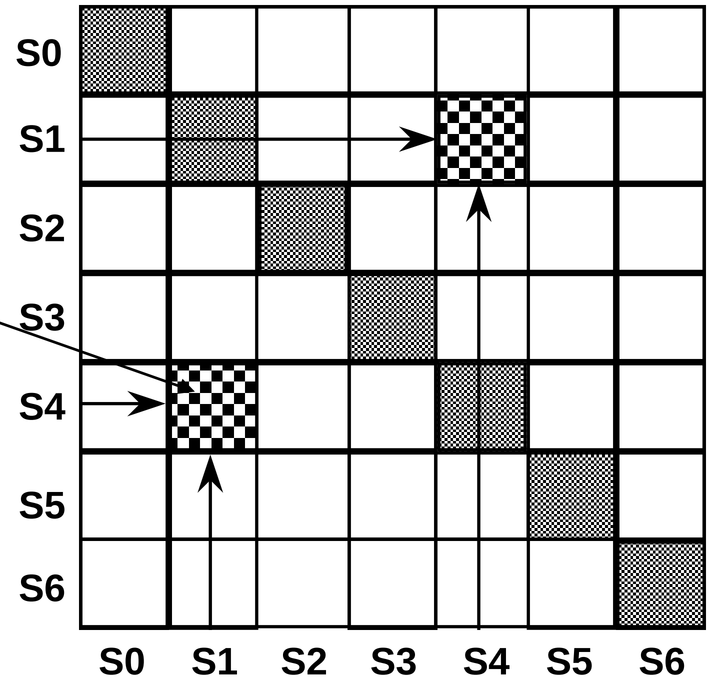
We know S0, S2 can be combined

No way to combine states S0 and S2 based on Next State Criterion!

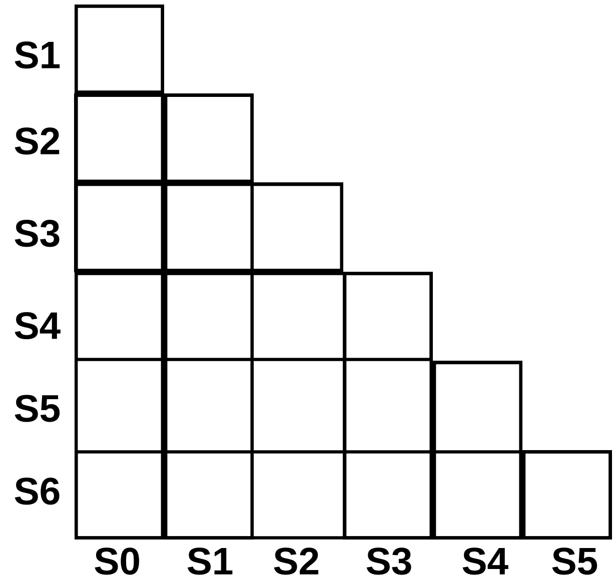
Implication Chart Method

- Enumerate all possible combinations of <u>states taken two at a time</u>

Next States Under all Input Combinations



Naive Data Structure:
Xij will be the same as Xji
Also, can eliminate the diagonal



Implication Chart

Implication Chart Method

Fill in the Implication Chart

Entry Xij : Row is Si, Column is Sj

Si is equivalent to Sj if outputs are the same and next states are equivalent

Xij contains the next states of Si, Sj which must be equivalent if Si and Sj are equivalent

If Si, Sj have different output behavior, then Xij is crossed out

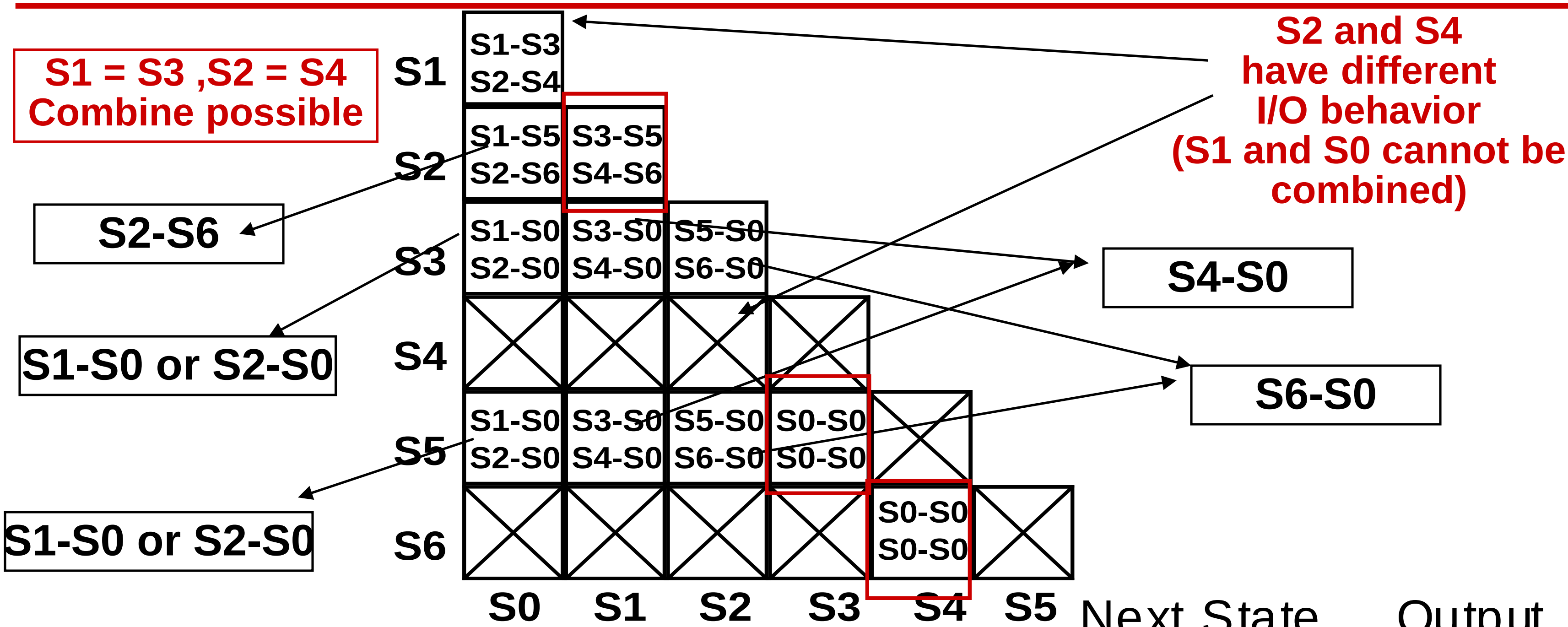
Example:

S0 transitions to S1 on 0, S2 on 1;
S1 transitions to S3 on 0, S4 on 1;

So square X<0,1> contains entries S1-S3 (transition on zero)
S2-S4 (transition on one)

S0	**S1-S3** **S2-S4**
	S1

Implication Chart Method



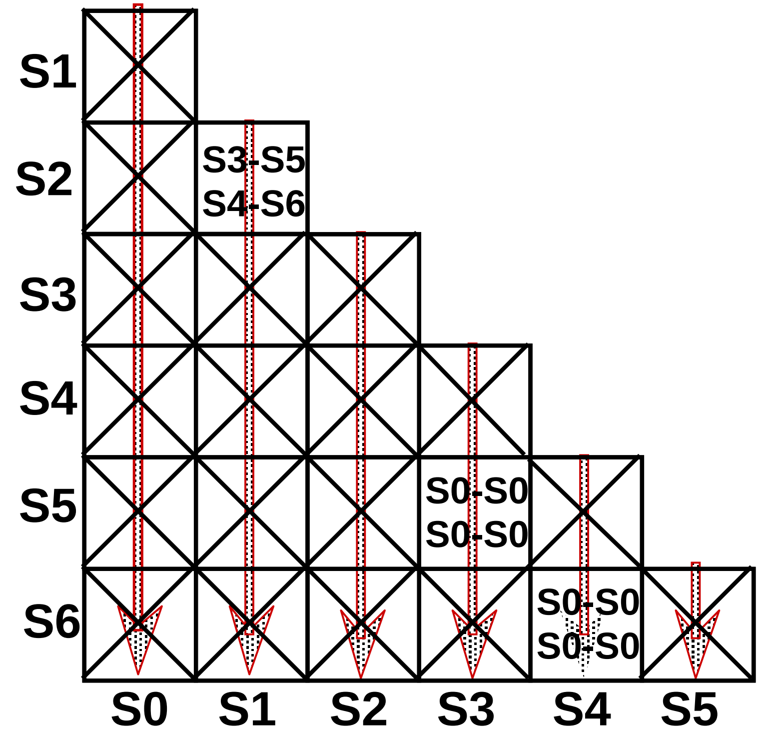
Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_0	S_0	0	0
01	S_4	S_0	S_0	1	0
10	S_5	S_0	S_0	0	0
11	S_6	S_0	S_0	1	0

Implication Chart Method

Results of First Marking Pass

<u>Second Pass (Adds No New Information)</u>

S3 and S5 are equivalent
S4 and S6 are equivalent
This implies that S1 and S2 are too!

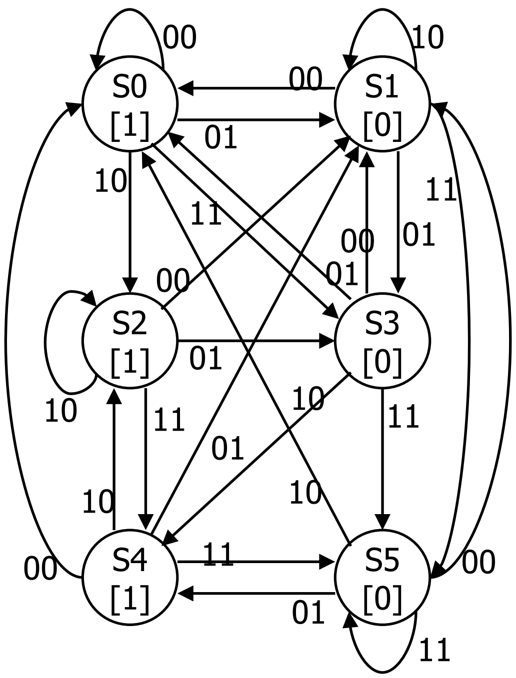


Reduced State Transition Table

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S'_1	S'_1	0	0
0 or 1	S'_1	S'_3	S'_4	0	0
00 or 10	S'_3	S_0	S_0	0	0
01 or 11	S'_4	S_0	S_0	1	0

More Complex State Minimization

- Multiple input example



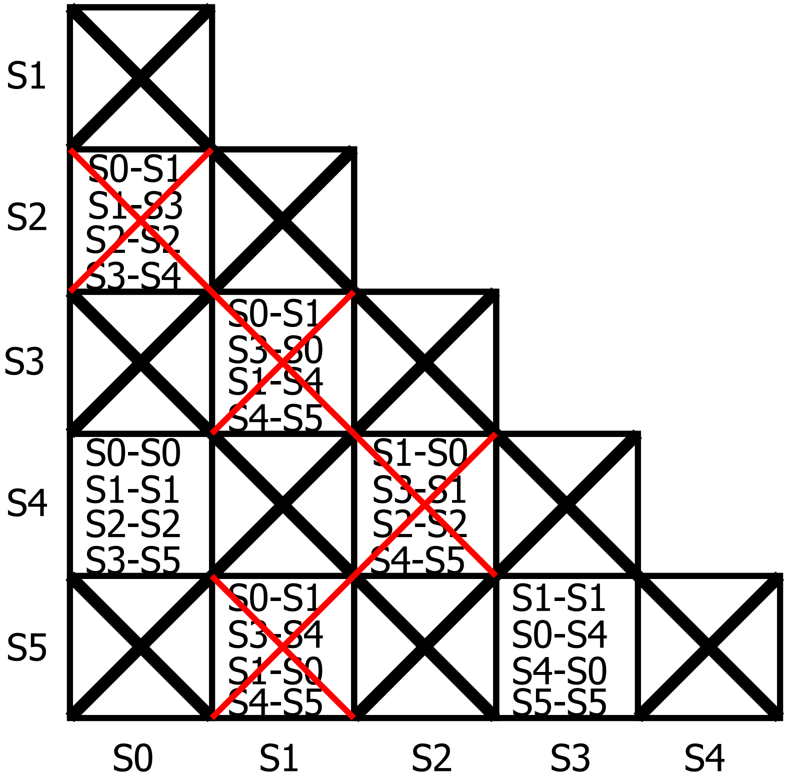
inputs here

present state	next state 00	01	10	11	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S4	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

symbolic state transition table

Minimized FSM

- Implication chart method
  - cross out incompatible states based on outputs
  - then cross out more cells if indexed chart entries are already crossed out



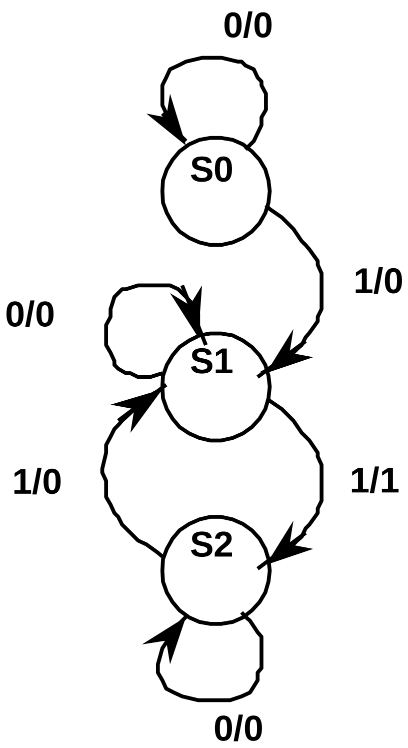
present state	next state 00	01	10	11	output
S0'	S0'	S1	S2	S3'	1
S1	S0'	S3'	S1	S3'	0
S2	S1	S3'	S2	S0'	1
S3'	S1	S0'	S0'	S3'	0

minimized state table
(S0==S4) (S3==S5)

Odd Parity EX

- **Example: 3 State Odd Parity Checker**

Present State	Next State X=0	Next State X=1	Output
S_0	S_0	S_1	0
S_1	S_1	S_2	1
S_2	S_2	S_1	0



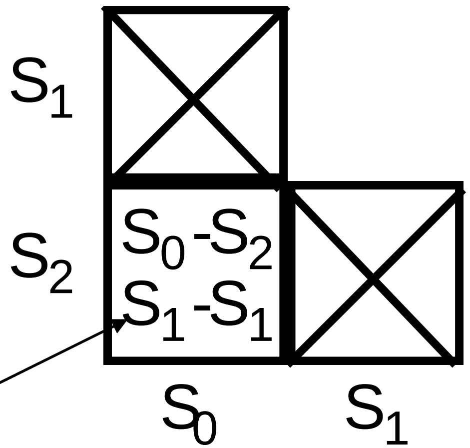
S0, S2 cannot be combined by using row matching method

Implication Chart Method

- **Example: 3 State Odd Parity Checker**

Does the method solve the problem with the odd parity checker·

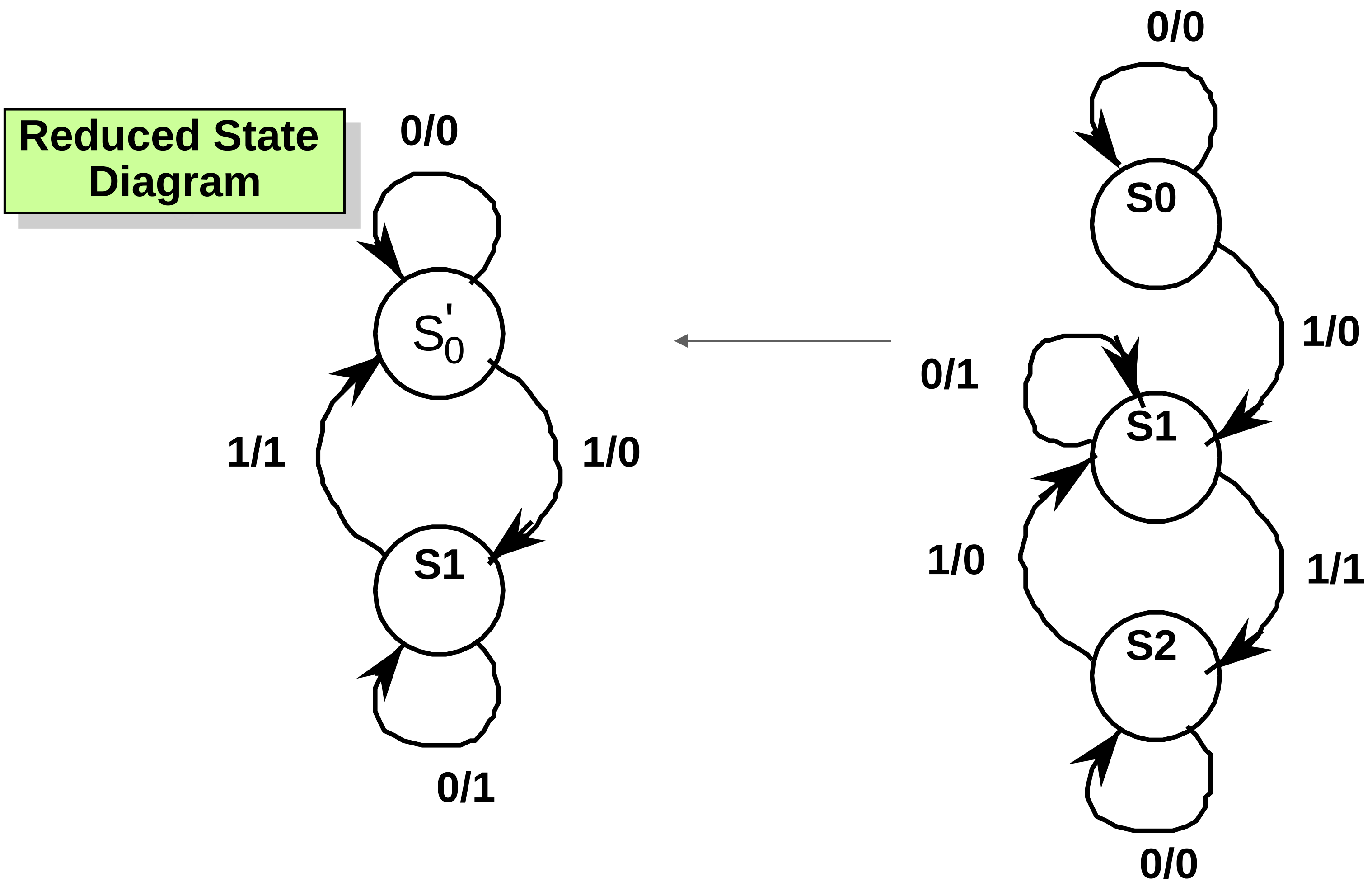
Implication Chart



S0 is equivalent to S2
since nothing contradicts this assertion!

Minimized FSM

- **Example: 3 State Odd Parity Checker**



Implication Chart Summary

1. *Construct implication chart*, one square for each combination of states taken two at a time

2. Square labeled Si, Sj, *if outputs differ, then square gets "X". Otherwise write down implied state pairs* for all input combinations

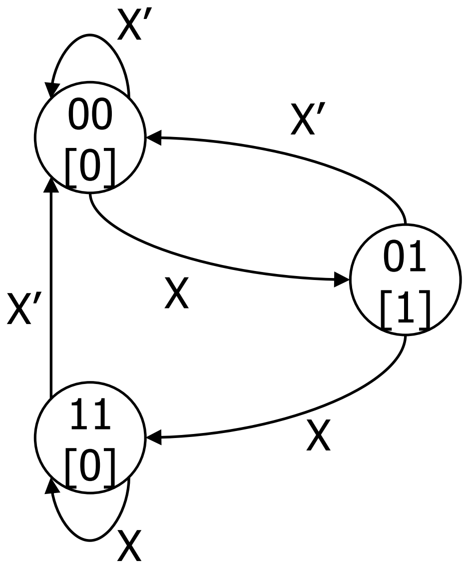
3. Advance through chart *top-to-bottom and left-to-right*. If square *Si, Sj contains next state pair Sm, Sn and that pair labels a square already labeled "X", then Si, Sj is labeled "X".*

4. Continue executing Step 3 until no new squares are marked with "X".

5. For each remaining *unmarked square Si, Sj*, *then Si and Sj are equivalent*.

Minimizing States May Not Yield Best Circuit

- Example: edge detector - outputs 1 when last two input changes from 0 to 1



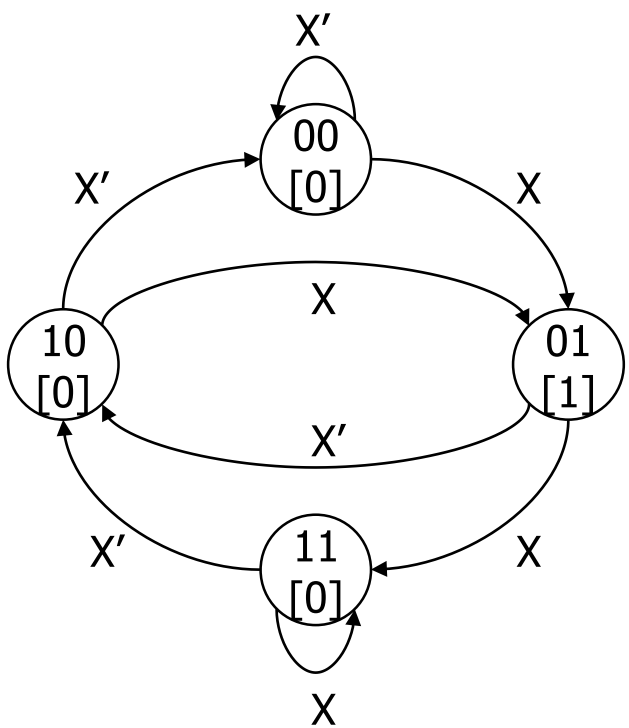
X	Q_1	Q_0	Q_1^+	Q_0^+
0	0	0	0	0
0	0	1	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	1	1	1
–	1	0	0	0

$Q_1^+ = X\ (Q_1 \text{ xor } Q_0)$

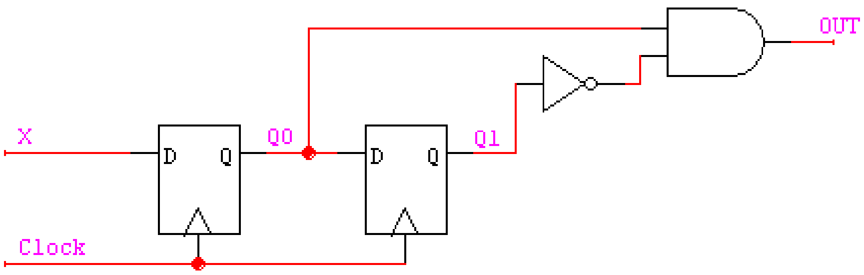
$Q_0^+ = X\ Q_1'\ Q_0'$

Another Implementation of Edge Detector

- "Ad hoc" solution - not minimal but cheap and fast



$Q_1^+ = Q_0$
$Q_0^+ = X$
$O_{ut} = Q_1^{+'} Q_0$



State Assignment

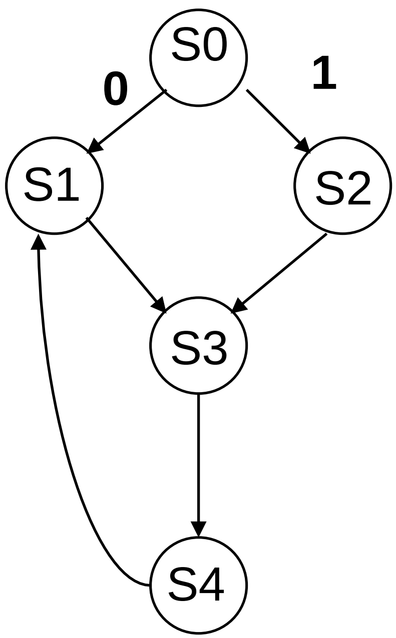
- Choose bit vectors to assign to each “symbolic” state
  - for n state bits for m states there are $2^n! / (2^n – m)!$
    $[\log n <= m <= 2^n]$
  - 2^n codes possible for 1st state, $2^n–1$ for 2nd, $2^n–2$ for 3rd, …
  - huge number even for small values of n and m
    - intractable for state machines of any size
    - <u>heuristics are necessary for practical solutions</u>
  - optimize some metrics for the combinational logic
    - size (amount of logic and number of FFs)
    - speed (depth of logic and fanout)
    - dependencies (decomposition)

State Assignment Strategies

- Possible strategies
  - *sequential* – just number states as they appear in the state table
  - *random* – pick random codes
  - *one-hot* – use as many state bits as there are states (bit=1 –> state)
  - *output* – use outputs to help encode states
  - *heuristic* – rules of thumb that seem to work in most cases
- No guarantee of optimality – another intractable problem

Pencil & Paper Heuristic Methods

- State Maps: similar in concept to K-maps
  If state X transitions to state Y, then assign "close" assignments to X and Y



State Name	Assignment Q_2	Q_1	Q_0
S_0	0	0	0
S_1	1	0	1
S_2	1	1	1
S_3	0	1	0
S_4	0	1	1

Assignment

State Name	Assignment Q_2	Q_1	Q_0
S_0	0	0	0
S_1	0	0	1
S_2	0	1	0
S_3	0	1	1
S_4	1	1	1

Assignment

Q_2 \ $Q_1 Q_0$	00	01	11	10
0	S_0		S_4	S_3
1		S_1	S_2	

State Map

Q_2 \ $Q_1 Q_0$	00	01	11	10
0	S_0	S_1	S_3	S_2
1			S_4	

State Map

Pencil & Paper Heuristic Methods

Minimum Bit Distance Criterion

Transition	*First Assignment Bit Changes*	*Second Assignment Bit Changes*
S0 to S1:	2	1
S0 to S2:	3	1
S1 to S3:	3	1
S2 to S3:	2	1
S3 to S4:	1	1
S4 to S1:	2	2
	13	7

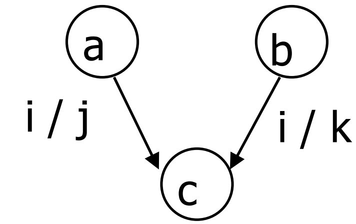
Traffic light controller: HG = 00, HY = 01, FG = 11, FY = 10 yields minimum distance encoding but not best assignment!

Heuristics for State Assignment

- Adjacent codes to states that <u>share a common next state</u>
  - group 1's in next state map

I	Q	Q^+	O
i	a	c	j
i	b	c	k

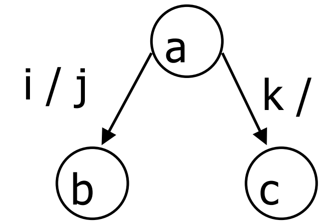
$$c = i * a + i * b$$



- Adjacent codes to states that <u>share a common ancestor state</u>
  - group 1's in next state map

I	Q	Q^+	O
i	a	b	j
k	a	c	l

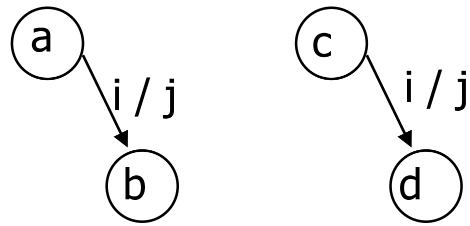
$$b = i * a$$
$$c = k * a$$



- Adjacent codes to states that <u>have a common output behavior</u>
  - group 1's in output map

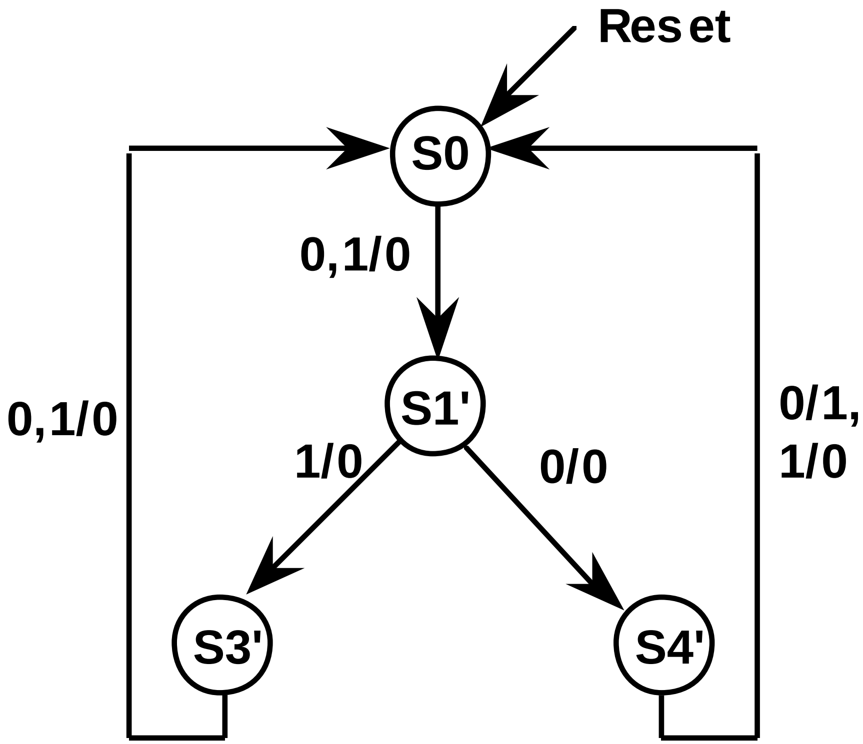
I	Q	Q^+	O
i	a	b	j
i	c	d	j

$$j = i * a + i * c$$
$$b = i * a$$
$$d = i * c$$



Heuristics for State Assignment

- **Example: 3-bit Sequence Detector**



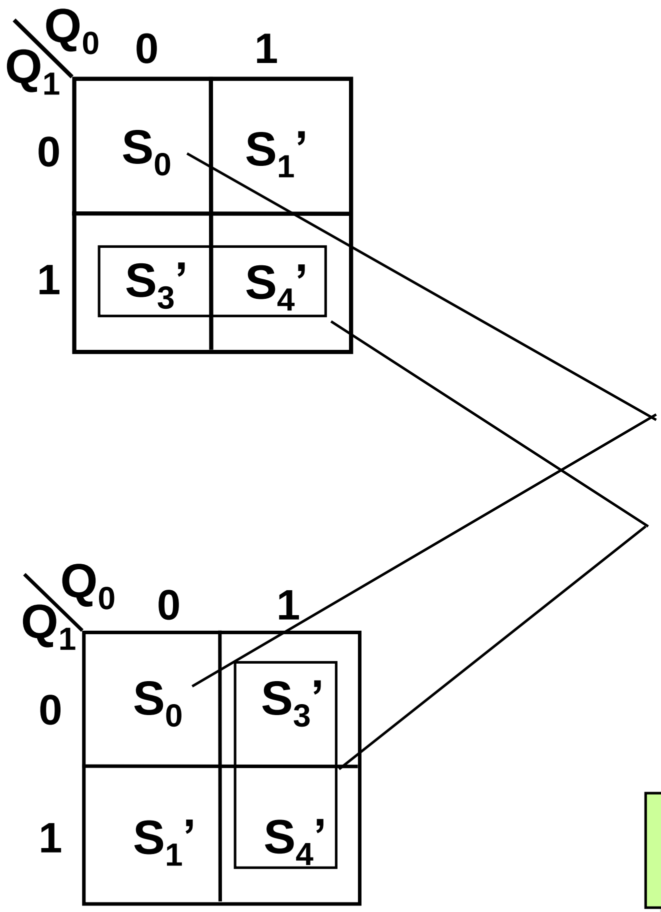
Highest Priority: (S3', S4')

Medium Priority: (S3', S4')

Lowest Priority:
0/0: (S0, S1', S3')
1/0: (S0, S1', S3', S4')

Heuristics for State Assignment

- **Example: 3-bit Sequence Detector**



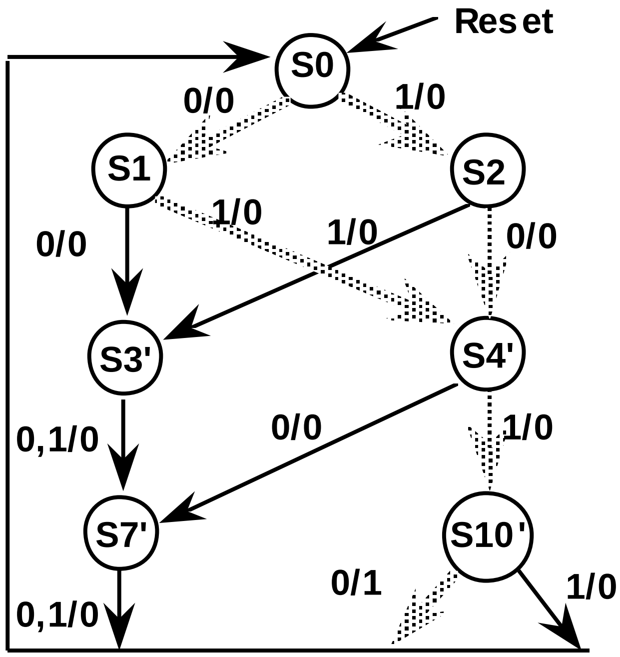
Reset State = 00

Highest Priority Adjacency

Not much difference in these two assignments

Heuristics for State Assignment

- **Another Example: 4 bit String Recognizer**



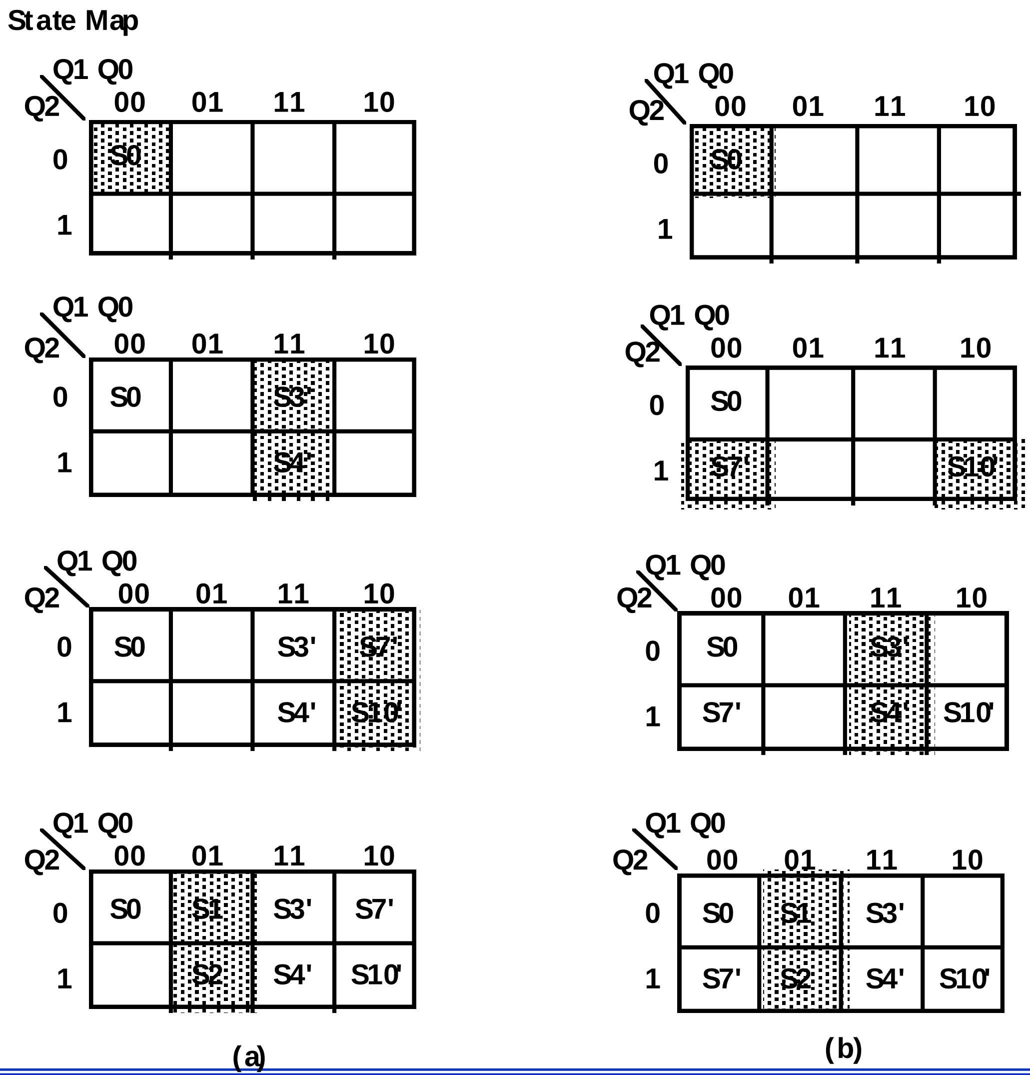
Highest Priority: (S3', S4'), (S7', S10')

Medium Priority:
(S1, S2), 2x(S3', S4'), (S7', S10')

Lowest Priority:
0/0: (S0, S1, S2, S3', S4', S7')
1/0: (S0, S1, S2, S3', S4', S7')

Heuristics for State Assignment

- **Another Example: 4 bit String Recognizer**



00 = Reset = S0

(S3', S4'), (S7', S10') placed adjacently, Then (S1, S2)

Heuristics for State Assignment

Effect of Adjacencies on Next State Map

Current State		Next State X = 0	Next State X = 1
(S_0)	000	001	101
(S_1)	001	011	111
(S_2)	101	111	011
(S'_3)	011	010	010
(S'_4)	111	010	110
(S'_7)	010	000	000
(S'_{10})	110	000	000

Q_0 X \ Q_2 Q_1	00	01	11	10
00	0	0	0	X
01	1	0	0	X
11	1	0	1	0
10	0	0	0	1

P_2

Q_0 X \ Q_2 Q_1	00	01	11	10
00	0	0	0	X
01	0	0	0	X
11	1	1	1	1
10	1	1	1	1

P_1

Q_0 X \ Q_2 Q_1	00	01	11	10
00	1	0	0	X
01	1	0	0	X
11	1	0	0	1
10	1	0	0	1

P_0

Current State		Next State X = 0	Next State X = 1
(S_0)	000	001	010
(S_1)	001	011	100
(S_2)	010	100	011
(S'_3)	011	101	101
(S'_4)	100	101	110
(S'_7)	101	000	000
(S'_{10})	110	000	000

Q_0 X \ Q_2 Q_1	00	01	11	10
00	0	1	0	1
01	0	0	0	1
11	1	1	X	0
10	0	1	X	0

P_2

Q_0 X \ Q_2 Q_1	00	01	11	10
00	0	0	0	0
01	1	1	0	1
11	0	0	X	0
10	1	0	X	0

P_1

Q_0 X \ Q_2 Q_1	00	01	11	10
00	1	0	0	1
01	0	1	0	0
11	0	1	X	0
10	1	1	X	0

P_0

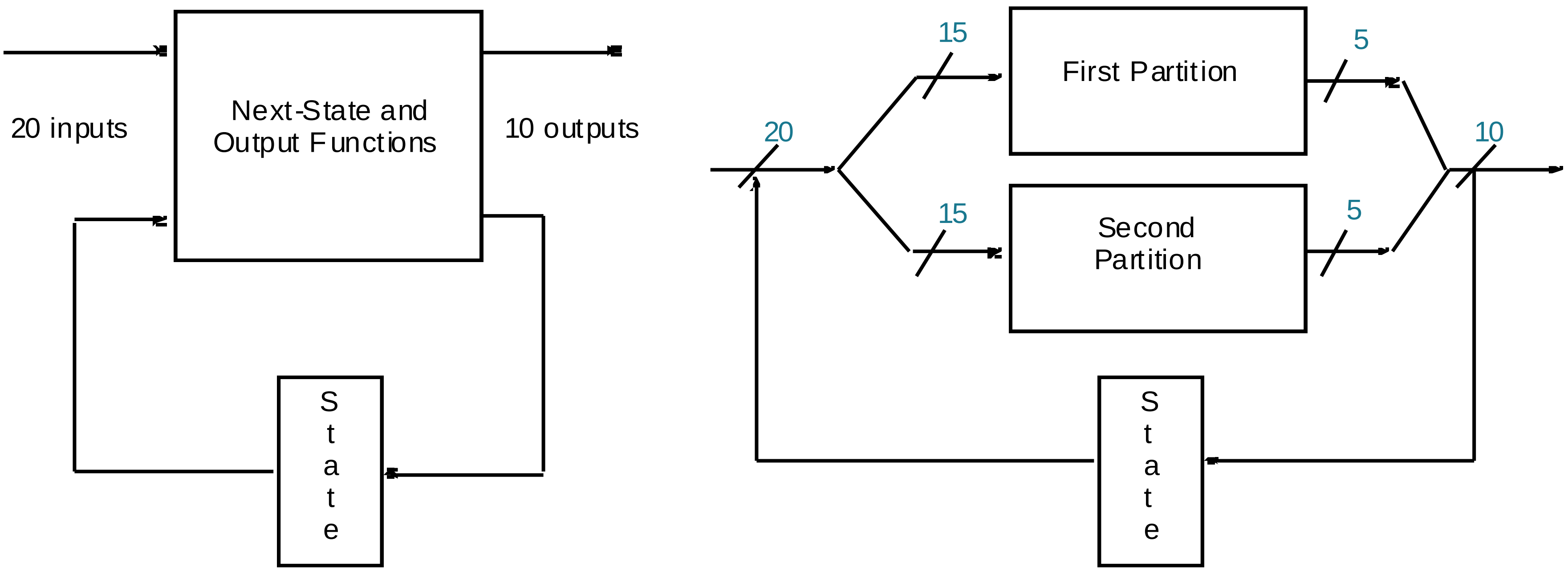
First encoding exhibits a better clustering of 1's in the next state map

General Approach to Heuristic State Assignment

- All current methods are variants of this
  - 1) determine which states “attract” each other (weighted pairs)
  - 2) generate constraints on codes (which should be in same cube)
  - 3) place codes on Boolean cube so as to maximize constraints satisfied (weighted sum)
- Different weights make sense depending on whether we are optimizing for two-level or multi-level forms

Why Partition ?

- Mapping FSMs onto programmable logic components:
  - <u>limited number of input/output pins</u>
  - <u>limited number of product terms</u> or other programmable resources

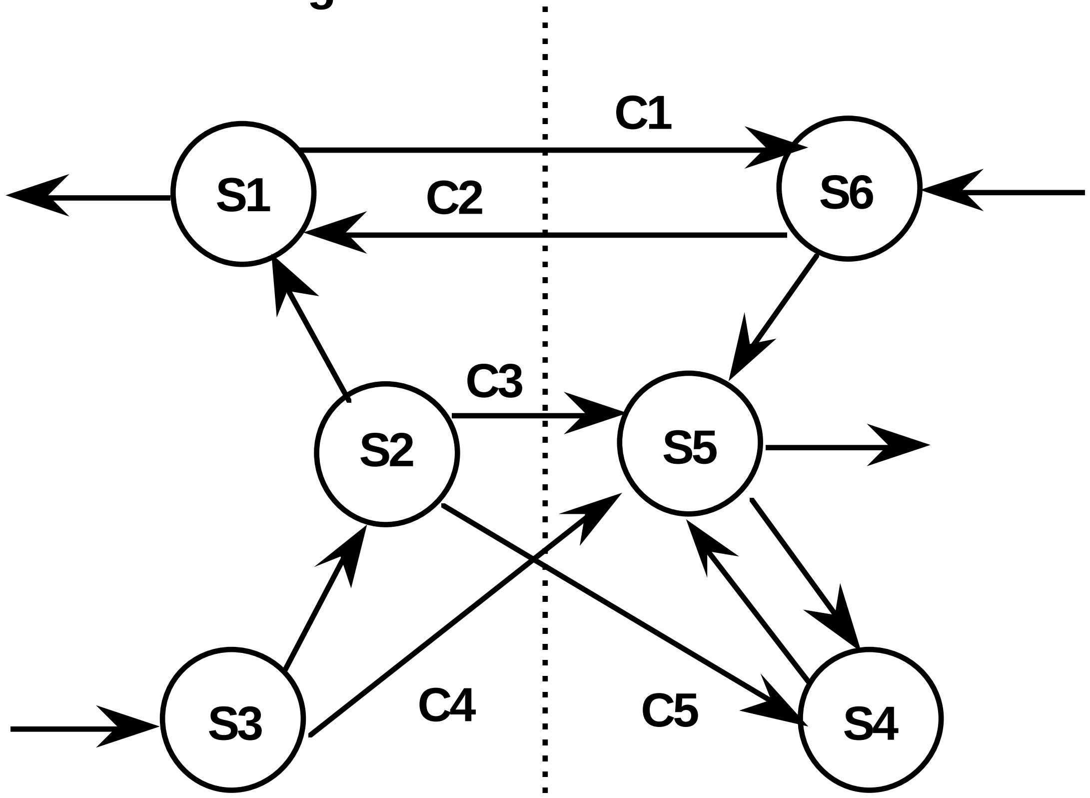


Example of Input/Output Partitioning:
5 outputs depend on 15 inputs
5 outputs depend on different overlapping set of 15 inputs

Finite State Machine Partitioning

Introduction of Idle States

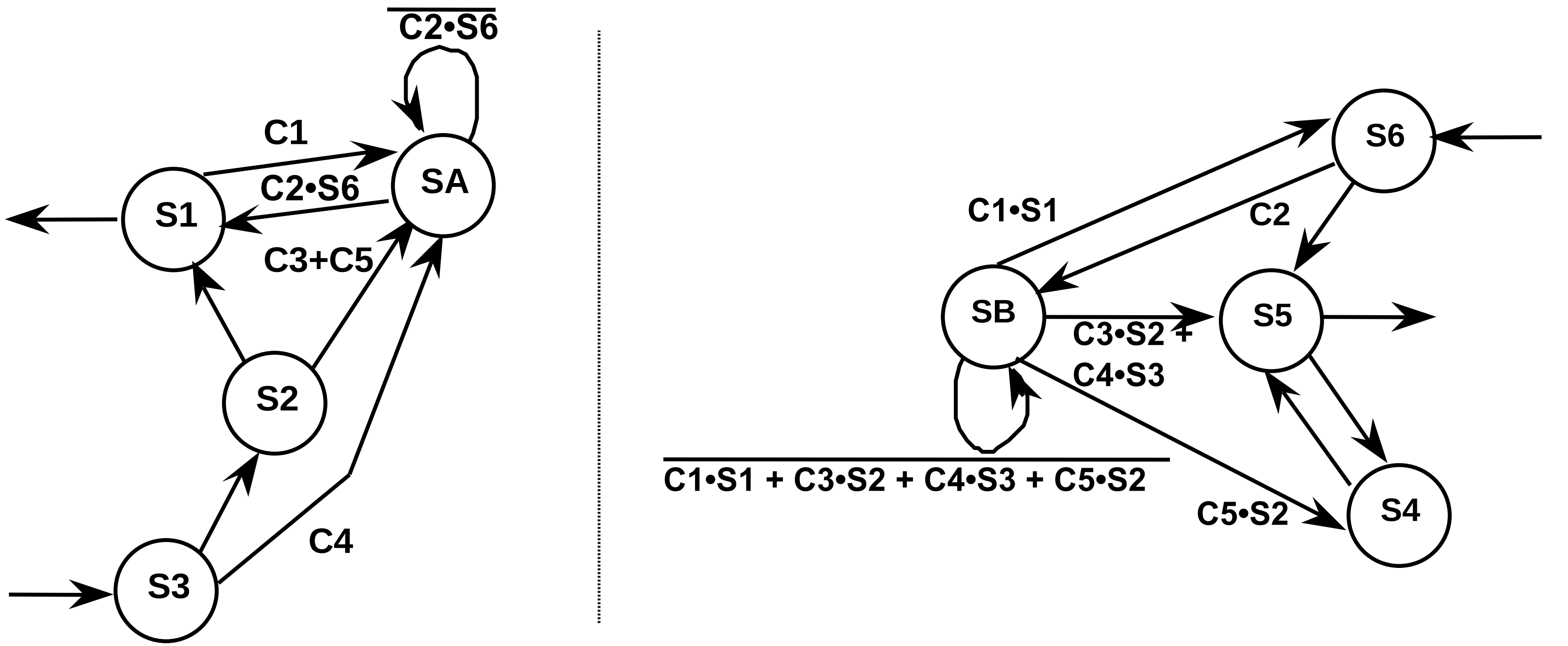
Before Partitioning



Finite State Machine Partitioning

Introduction of Idle States

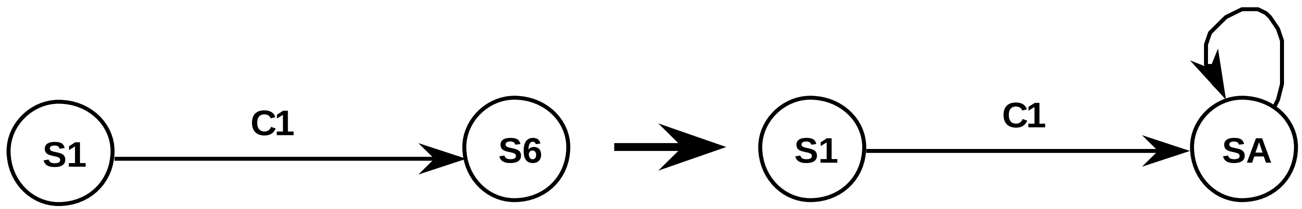
After Partitioning



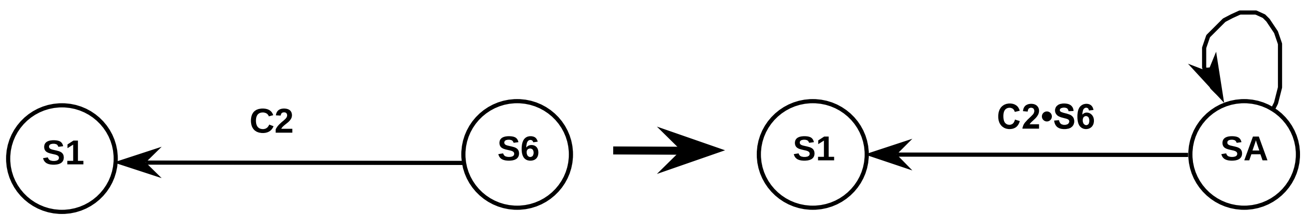
Finite State Machine Partitioning

Rules for Partitioning

Rule #1: Source State Transformation; SA is the Idle State



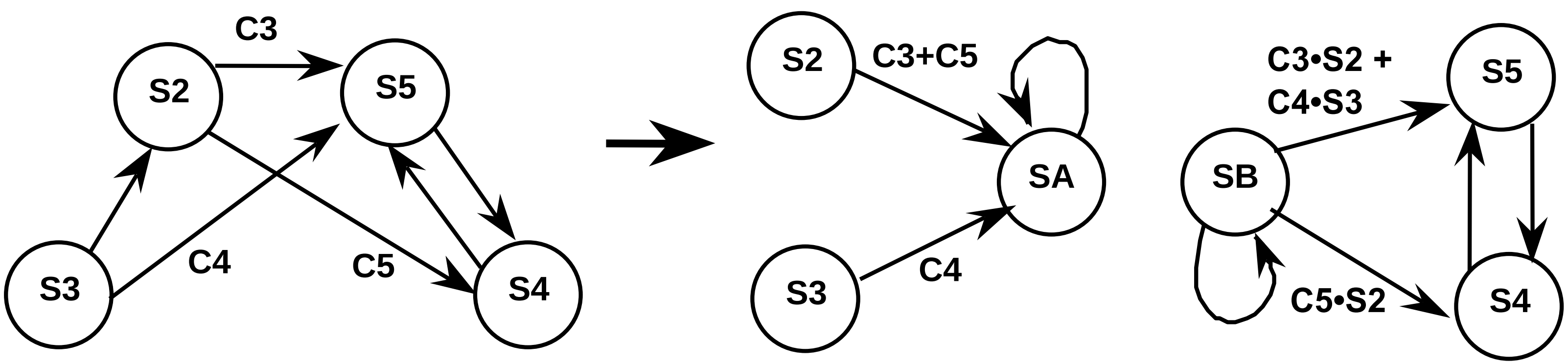
Rule #2: Destination State Transformation



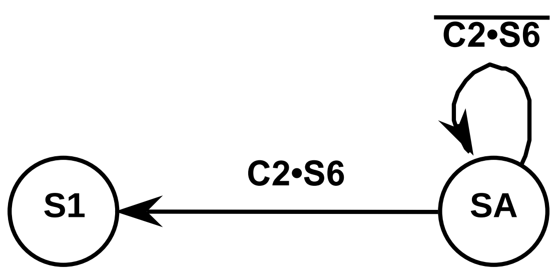
Finite State Machine Partitioning

Rules for Partitioning

Rule #3: Multiple Transitions with Same Source or Destination

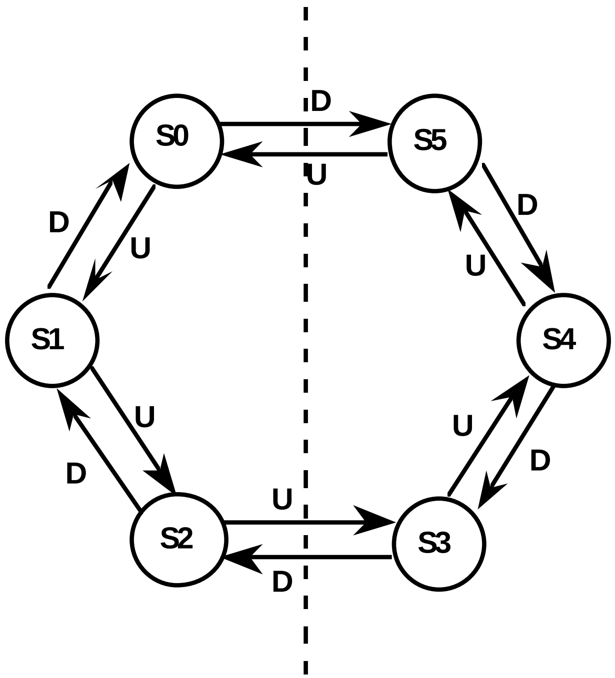


Rule #4: Hold Condition for Idle State



Finite State Machine Partitioning

Another Example

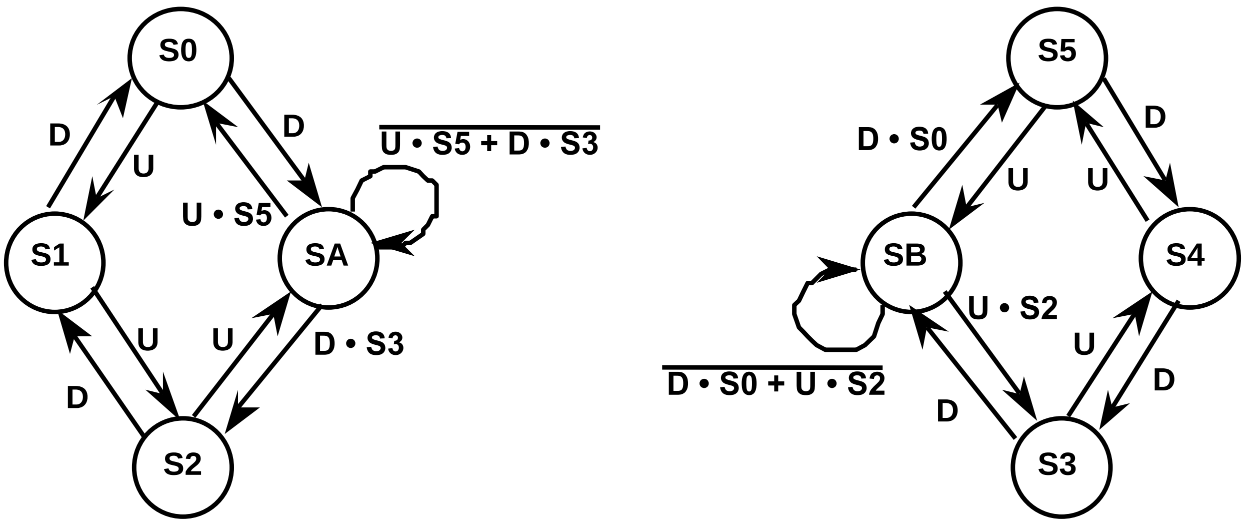


<u>**6 state up/down counter**</u>

building block has 2 FFs + combinational logic

Finite State Machine Partitioning

6 State Up/Down Counter



Introduction of the two idle state SA, SB

Count sequence S0, S1, S2, S3, S4, S5:
S2 goes to SA and holds, leaves after S5
S5 goes to SB and holds, leaves after S2

Down sequence is similar

Sequential Logic Optimization Summary

- State minimization
  - straightforward in fully-specified machines
  - computationally intractable, in general (with don’t cares)
- State assignment
  - many heuristics
  - best-of-10-random just as good or better for most machines
  - output encoding can be attractive (especially for PAL implementations)