STATE MINIMIZATION

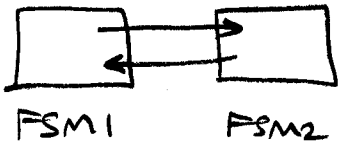
AIM: Implement a FSM with the fewest number of states possible.

Advantage = minimizes the flip-flops you need to use.

[BUT: may not be necessary to do this or may not be what you want.

Examples:

1. FPGA has a fixed # of ffp's. If your current design uses/needs a number of flip-flops less than available number, state minimization not necessary.
2. VLSI chip: trying to minimize the number of flip-flops may significantly increase the wiring complexity to implement next-state equations. (So area of wires ↑ and (total) overall area may ↑.)
3. state minimization may destroy the modularity of the design: e.g.,



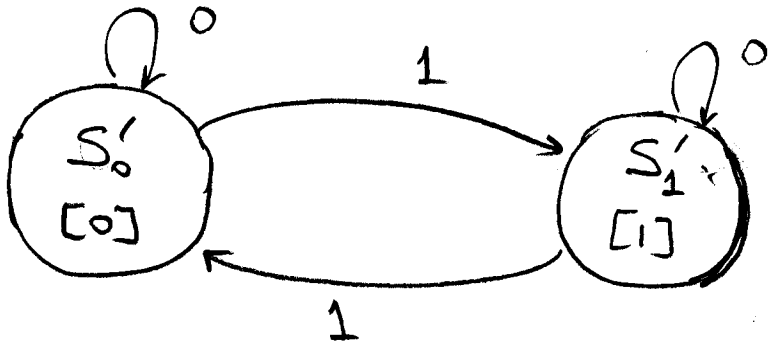
. minimizing the number of states may interleave unrelated parts of FSM1 & FSM2. Can't change FSM1 anymore if need to. ⇒ less flexible, less modular]

With these caveats, let's start.

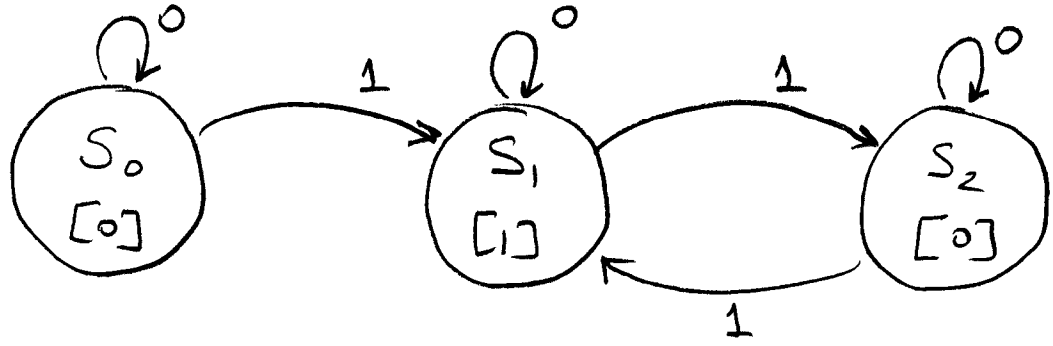
Intuition

Ex1] Examine the following two machines (which are versions of the parity checker without reset.)

Machine 1



Machine 2:

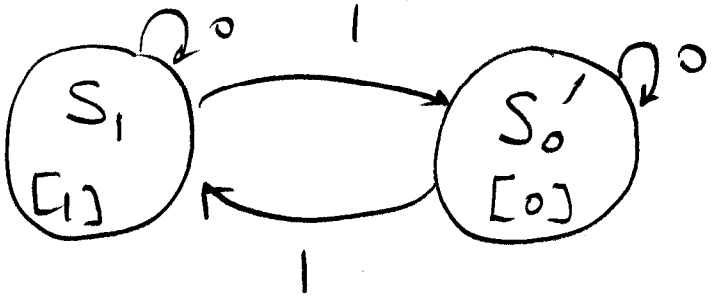


It is not possible to distinguish between states S_0 and S_2 in Machine 2: If I'm sitting at state S_0, how can I tell whether I am in S_0 or S_2? I cannot. Why not? Because

- the outputs are the same. (Both [0].), and
- the transitions <u>from</u> that state are the same. (0 loops back to me, and 1 transitions to S_1.)

[Note that S_2 has an incoming arrow $\xrightarrow{1}$ into it, which S_0 does not. But this does not matter because state (by definition) captures all of the relevant past. Given the state, how I got into that state does not matter in determining the future behavior. This is called the Markov property; namely, that a state is characterized by only its output (in Moore case) and the set of transition arrows emanating from it.]

- In machine 2, S_0 and S_2 are called "equivalent states". (Precise definition will come later.)
- If two states are equivalent, then I can consolidate/reduce/merge them to a single state. In machine 2, reduction of S_0 and S_2 into a single state gives



[Note how we took the union of the incoming arcs into S_0 and S_2 when constructing the new state S_0'.]

— But the above machine is the same as machine 1. We say that Machine 1 and machine 2 are equivalent (Sequential) machines.

FORMAL TECHNIQUES

Main Questions:

1 — How can we reduce a given machine to the minimum number of states?

2 — How can we show (prove) that two machines are equivalent?

—

Main techniques:

1 — Row matching: • Can identify equivalence in some but not all cases.

• Usually used as a first step in reduction before more powerful methods are applied.

2 — Implication chart (a.k.a implication table): can find the min. #

2- Implication table (or implication chart).

- Can find the optimal (i.e. minimum # states) solution.

Example 2:

Build a FSM that takes a 1-bit at a time and recognizes (accepts) 0101 and 1001, but rejects all other 4-bit vectors. (The machine processes quadruples.)

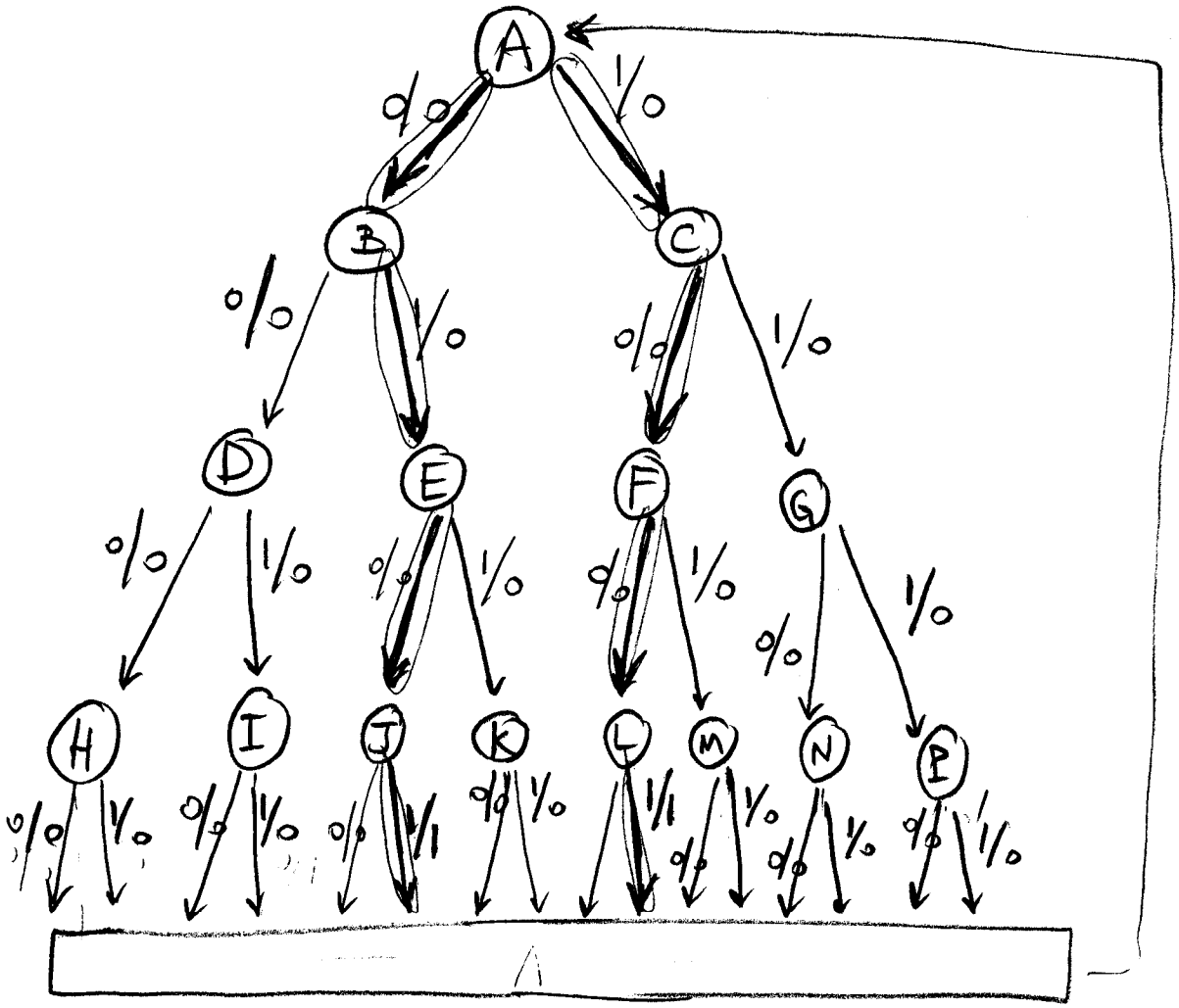
a- Can do this problem intuitively: i.e. build the states while trying to keep the number of states low.

or

b- Write down a state diagram without worrying about the number of states. Then apply tools to reduce the # of states.

We will choose option b to illustrate state reduction/minimization.

Build it as a Mealy machine.



MEALY STATE DIAGRAM

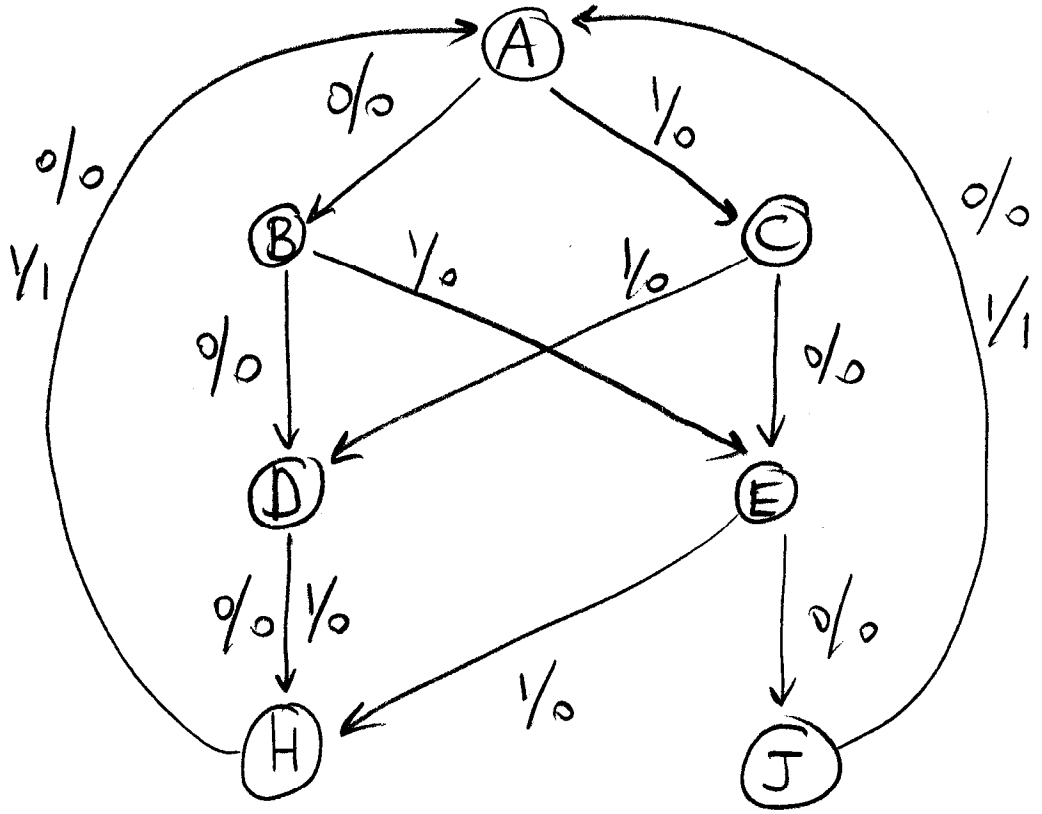
Application of [Row Matching]

MEALY STATE TABLE

Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
A	B	C	0	0
B	D	E	0	0
C	~~F~~ E	~~G~~ D	0	0
D	H	~~I~~ H	0	0
E	J	~~K~~ H	0	0
~~F~~ (E ⑧)	~~L~~ J	~~M~~ H	0	0
~~G~~ (D ⑦)	~~N~~ H	~~P~~ H	0	0
(H)	A	A	0	0
~~I~~ (eliminate ① H)	A	A	0	0
J	A	A	0	1
~~K~~ (H ②)	A	A	0	0
~~L~~ (J ⑥)	A	A	0	1
~~M~~ (H ③)	A	A	0	0
~~N~~ (J ④)	A	A	0	0
~~P~~ (H ⑤ for ...)	A	A	0	0

(no 0 state!)

- can easily draw the final reduced & (Mealy) state diagram from table.
  - with use 7 states.



- Row matching: technique uses equal states.
  - In general this technique cannot find the minimal set of states

Counterexample:

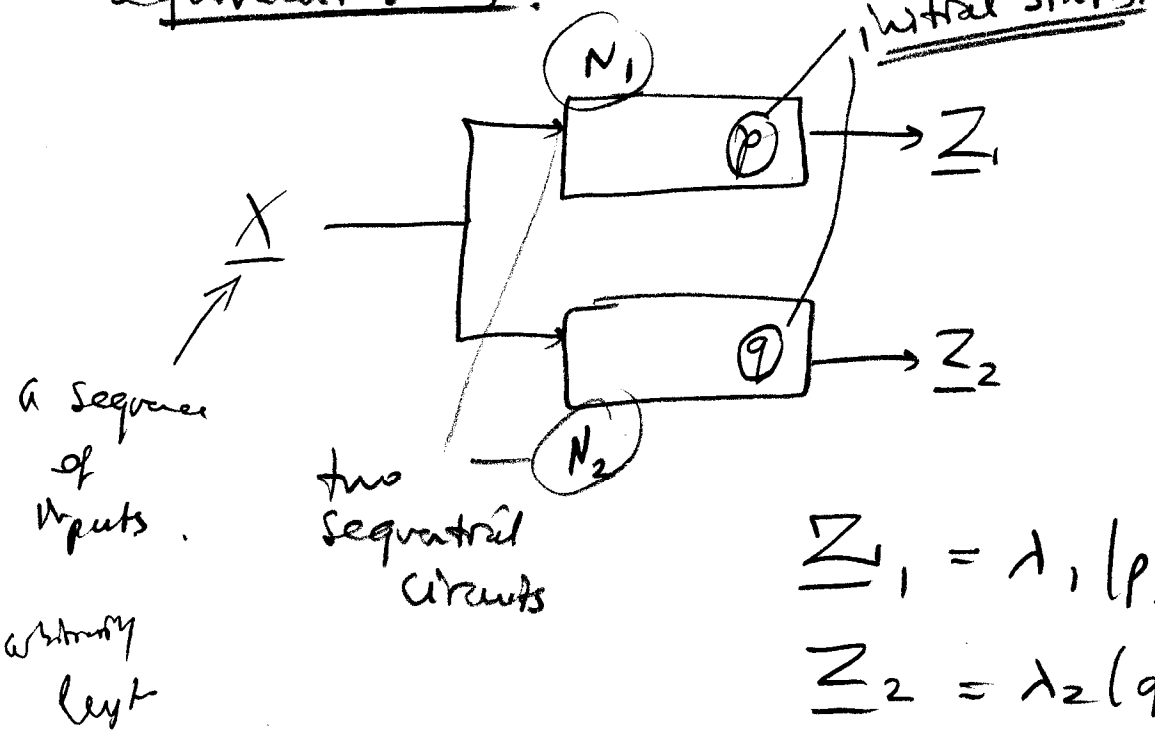
— * Odd-parity checker:

S_0 and S_2 are equivalent but their rows don't match.

∴ Row matching applied to that example does not produce a minimal solution.

∴ need more powerful methods.

— First: we need to develop a precise definition of equivalent states.



$$\underline{Z}_1 = \lambda_1(p, \underline{X})$$

$$\underline{Z}_2 = \lambda_2(q, \underline{X})$$

The state p in N_1 is equivalent to state q in N_2 iff $\lambda_1(p, \underline{X}) = \lambda_2(q, \underline{X})$ $\forall \underline{X}$.

Problem: $\underline{X}$ arbitrarily long

Thm: Two states p and q of a seq. circuit are equivalent iff $\forall$ ~~single input~~ X, the outputs are the same and the next states are equivalent

i.e. $\lambda(p, X) = \lambda(q, X)$ and $\delta(p, X) \equiv \delta(q, X$

* recursive definition!
