Testing Finite State Machine

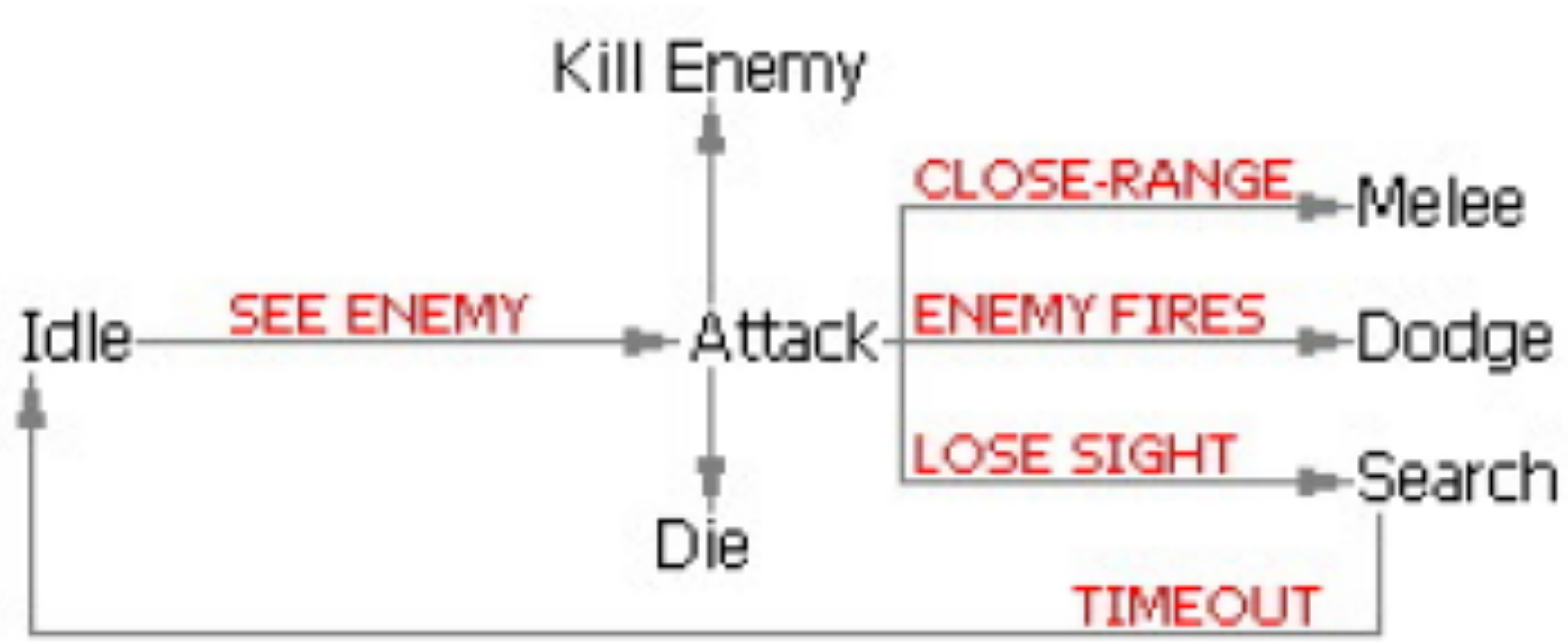
CS453 Automated Software Testing

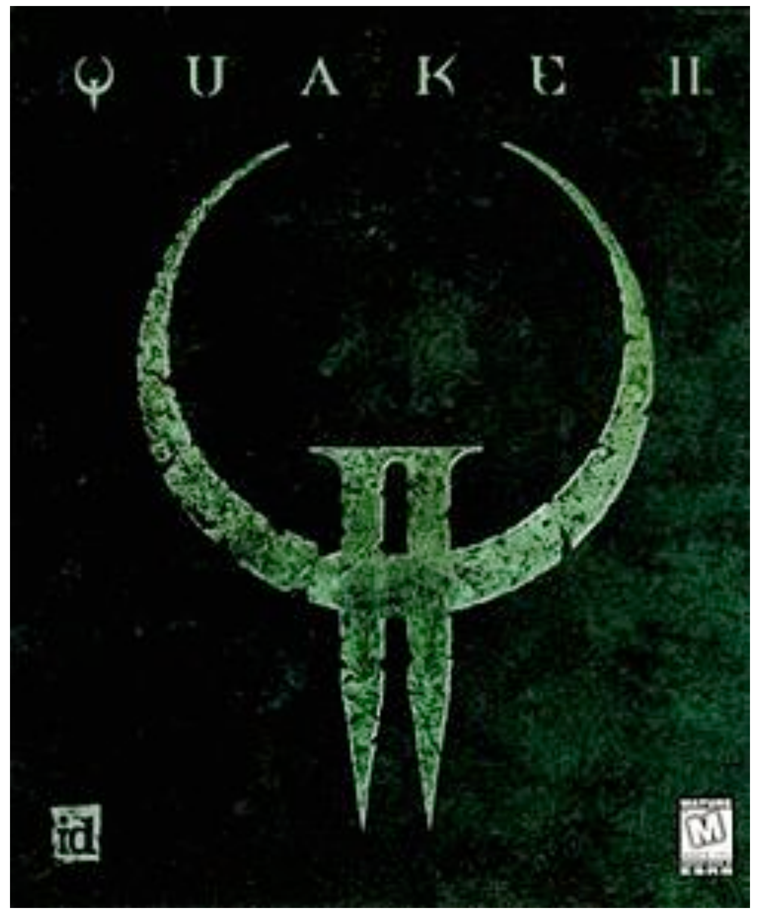
Shin Yoo | COINSE@KAIST

State-based Models

- Many real systems have some internal states. For example:
  - Embedded Control Systems
  - Communication Protocols
  - Video Games
  - …
- These systems might be specified in state-based models using e.g. Statecharts, SDL or FSM.

Quake II





Examples from Finite State Machine for Games, UW CG group

States and transitions

- A system may be modelled by:
  - a set of logical states
  - transitions between these states
- Then:
  - each state will normally represent some set of values for the state variables
  - each transition will represent the use of some operation to the state

Finite State Machine

- A (deterministic) finite state machine is defined by tuple $(S, s_1, X, Y, \delta, \lambda)$in which:
  - S is a finite set of states and s_1 is the initial state
  - X is the finite input alphabet/set
  - Y is the finite output alphabet/set
  - function δ is the state transfer function
  - function λ is the output function
- We can extend δ and λ to take sequences, giving $\delta*$ and $\lambda*$.

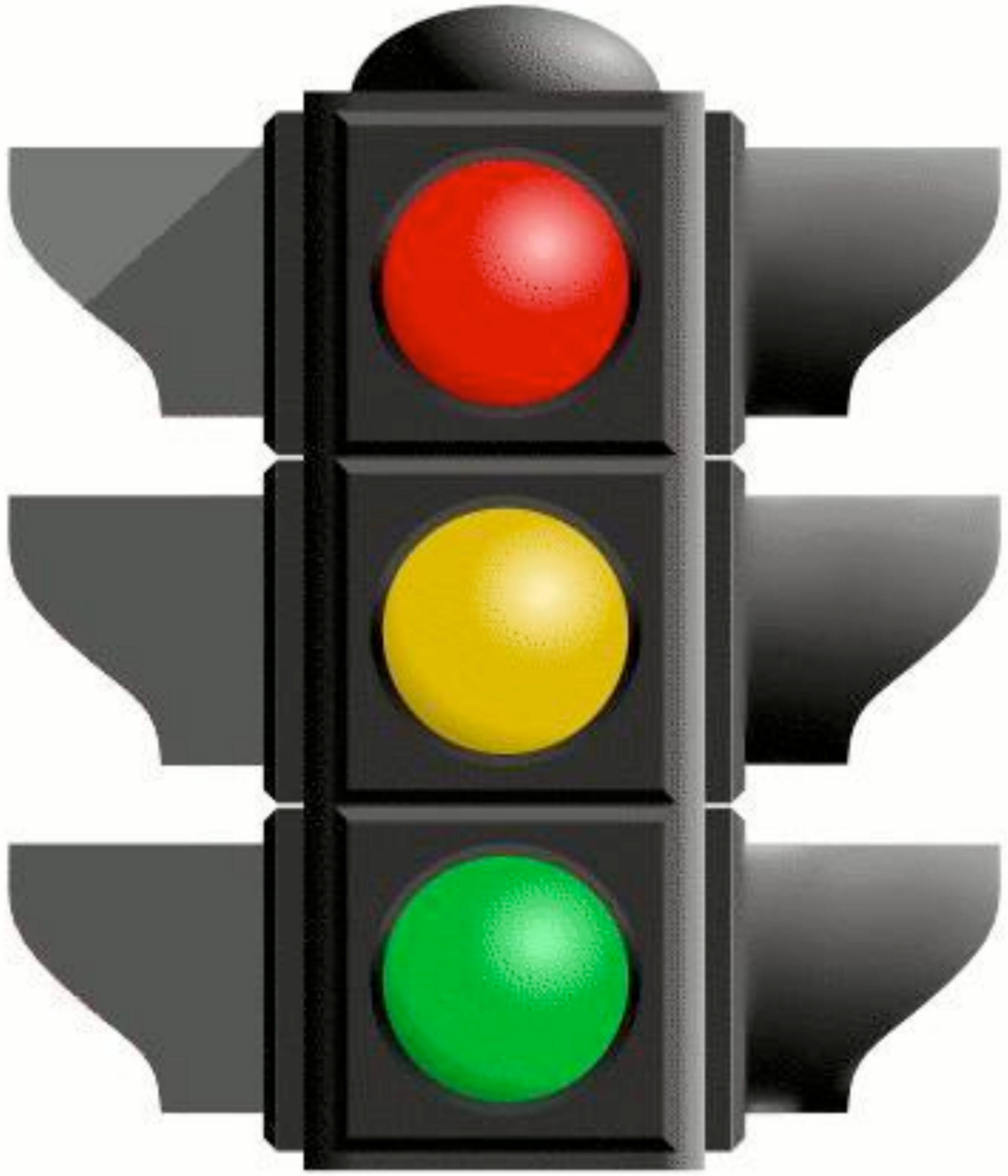
Behaviour of an FSM

- If we input a sequence x when M is in its initial state we get output sequence $\lambda^*(s_1, x)$ and M moves to state $\delta^*(s_1, x)$.

- If we input a sequence x when M is in state s we get output sequence $\lambda^*(s, x)$ and M moves to state $\delta^*(s, x)$.

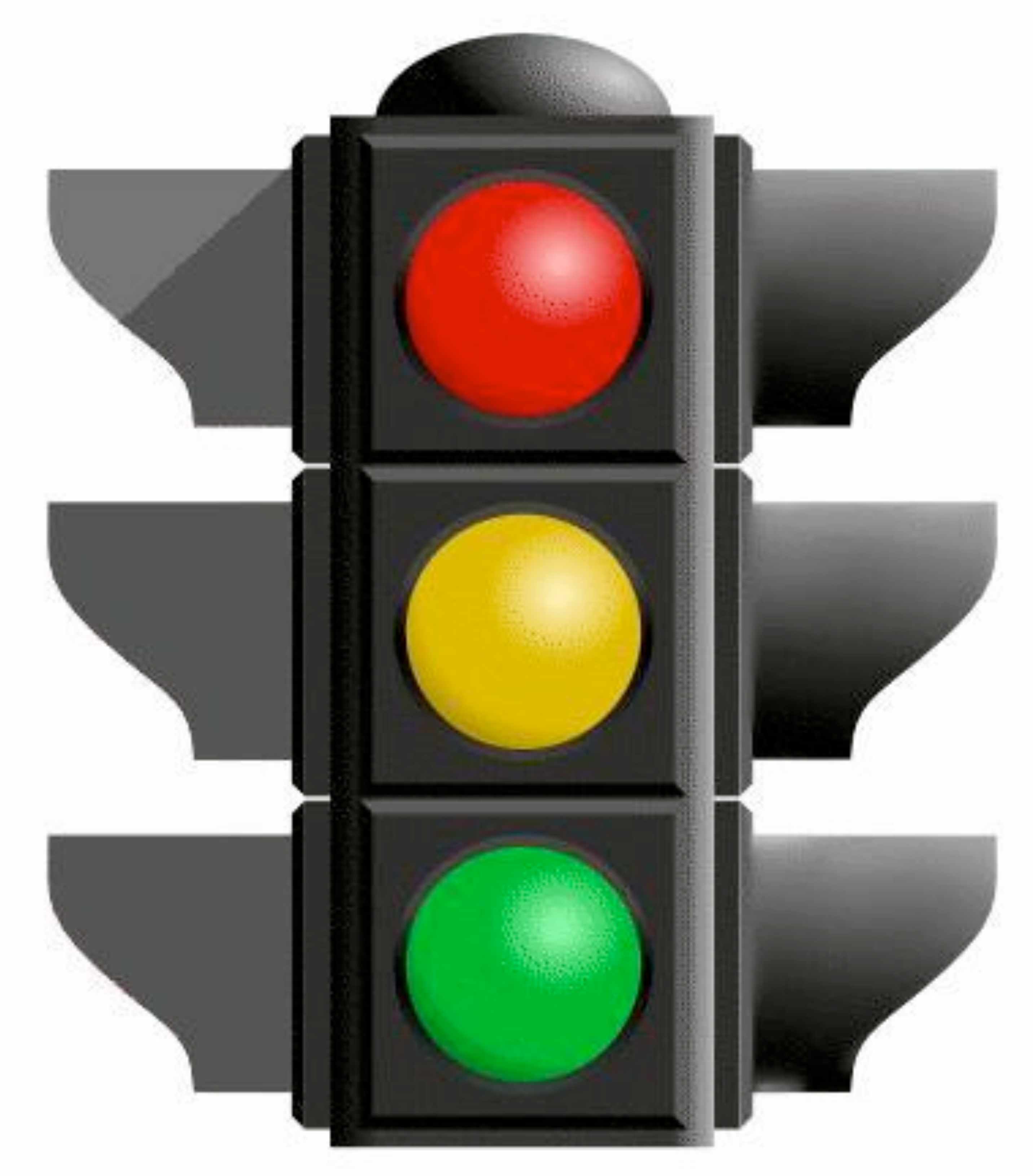
Example: Traffic Lights

- We will consider the following system:
- There are three colours for the lights: red, amber, and green
- The control system receives a message `ch` indicating when it should change the colour.
- It changes state and outputs a value to the lights telling them what the colour should be.



Example: Traffic Lights

- The FSM `MT` is defined by:
- State set `{Red,Green,Amber1,Amber2}`
- Initial state: `Green`
- Input alphabet `{ch}`
- Output alphabet `{green, red, amber}`
- State transfer function: `δ(Green,ch)=Amber1, δ(Amber1,ch)=Red, δ(Red,ch)=Amber2, δ(Amber2,ch)=Green`
- Output function: `λ(Green,ch)=amber , λ(Amber1,ch)=red , λ(Red,ch)=amber , λ(Amber2,ch)=green`

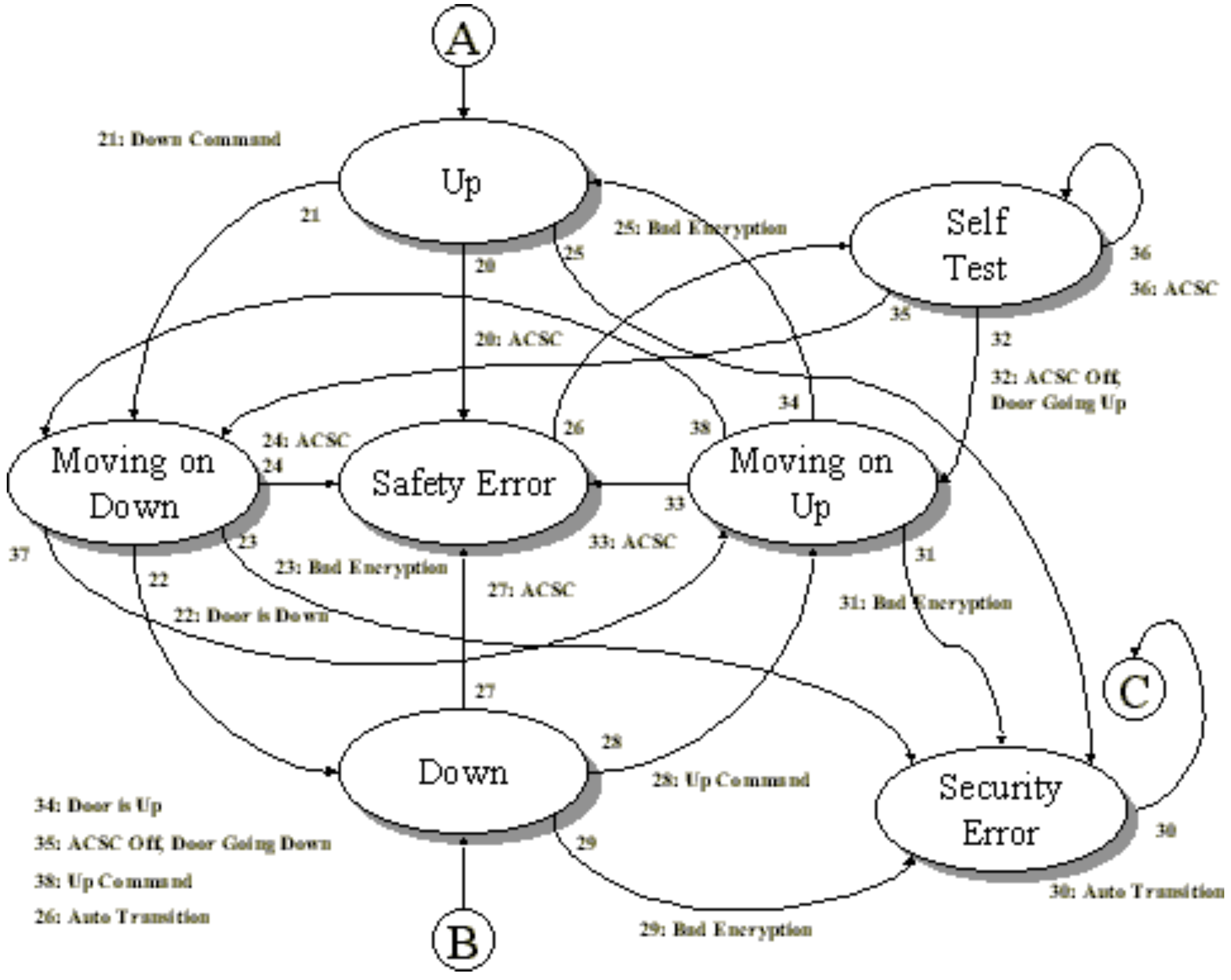


FSMs and Directed graphs

- FSM M can be represented by a directed graph (digraph) $G = (V, E)$ in which:
  - A state s_i is represented by a vertex v_i
  - If input x can move M from state s_i to state s_j with output y we add an edge $(s_i, s_j, x/y)$: an edge from s_i to s_j with label x/y.
- Then the paths (from v_1) in G represent the input/output sequences of M.

State Diagram

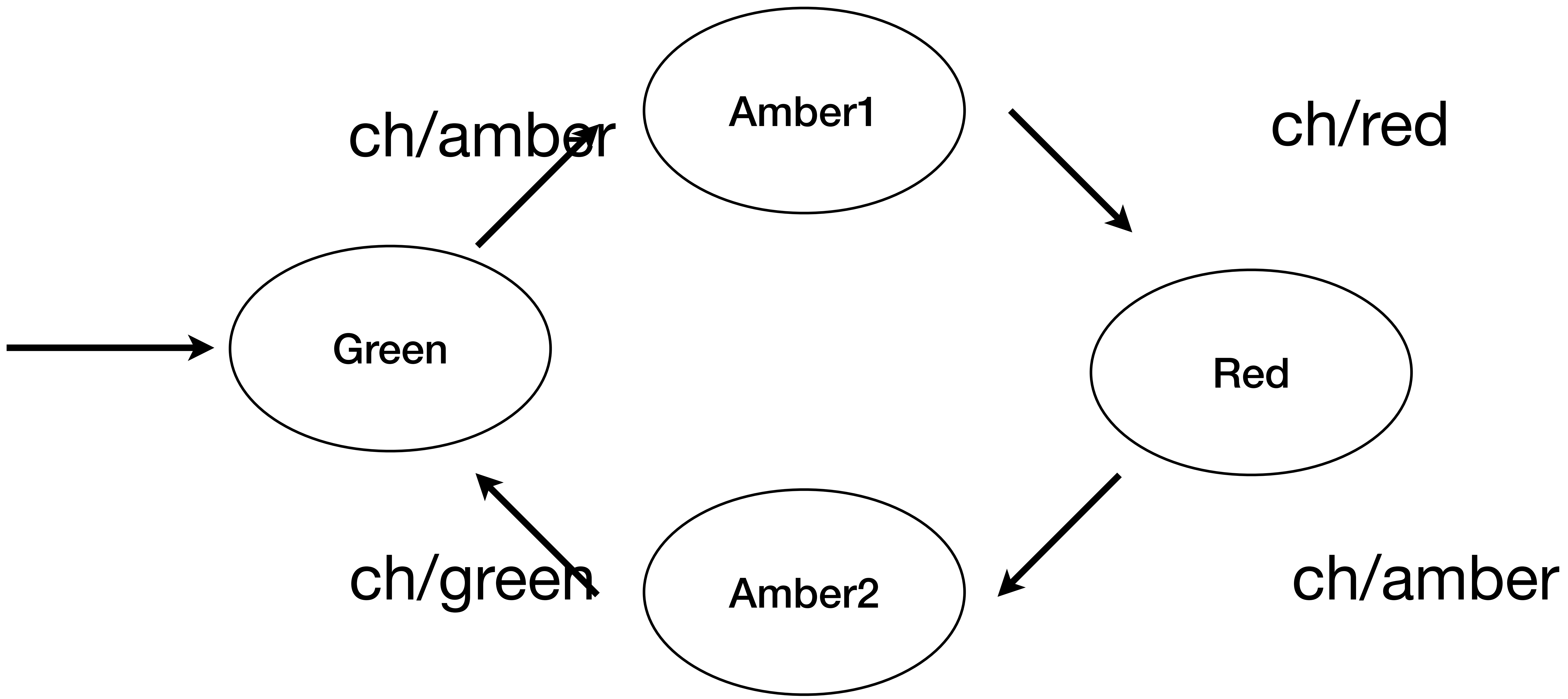
- A state-based system can be represented by a state diagram.
  - Each state is represented by a node.
  - The transitions are represented by arcs between nodes



Finite **S**tate **M**achine

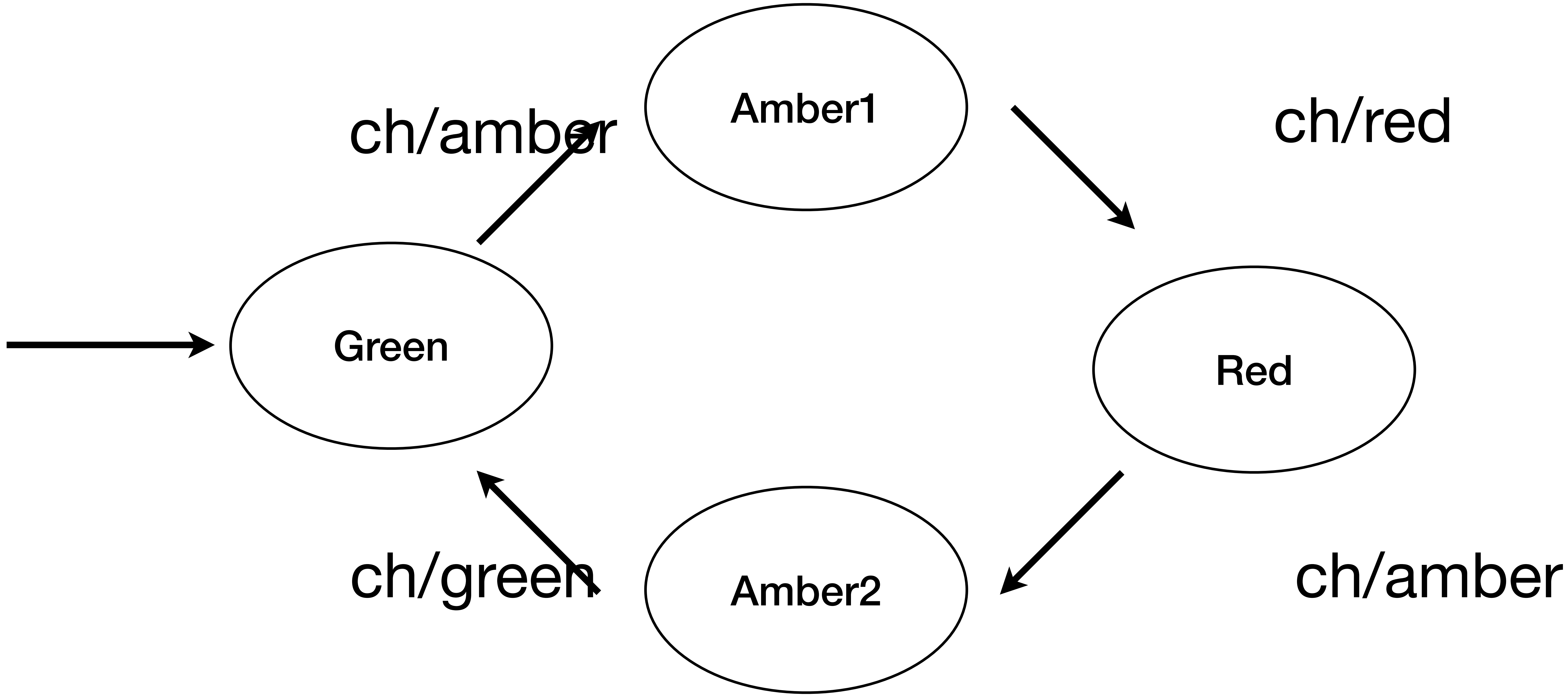
Flying **S**paghetti **M**onster

State Diagram for FSM `MT`



Actions in `MT`

Suppose we input sequence `<ch, ch>` when `MT` is in state `Amber2`.



We have that λ_*(`Amber2,chch) = green,amber` and δ_*(`Amber2,chch) = Amber1`.

Actions in `MT`

- Suppose we input sequence <ch, ch> when MT is in state Amber2.
  - The first ch moves MT to state Green and produces output green.
  - The second ch move MT from state Green to state Amber1 and leads to output amber.
  - We have that $\lambda*$(Amber2,chch) = green,amber and $\delta*$(Amber2,chch) = Amber1.

Initially and Strongly connected FSMs

- M is initially connected if:
  - Every state can be reached from the initial state – i.e. if for each state s there is some sequence of edges from the initial state to s.
- M is strongly connected if:
  - For every ordered pair of states (s, s') there is some input sequence that takes M from s to s' – i.e., if for each s, s' there is some sequence of edges from s to s'.

FSM Equivalence

- Two FSMs M and M' with the same input alphabets are equivalent if, for each input sequence they produce the same output sequence.

Minimal FSMs

- An FSM is minimal if there is no equivalent FSM with fewer states.
- If M is not minimal, it can be rewritten to form an equivalent minimal FSM.

Reset Operations

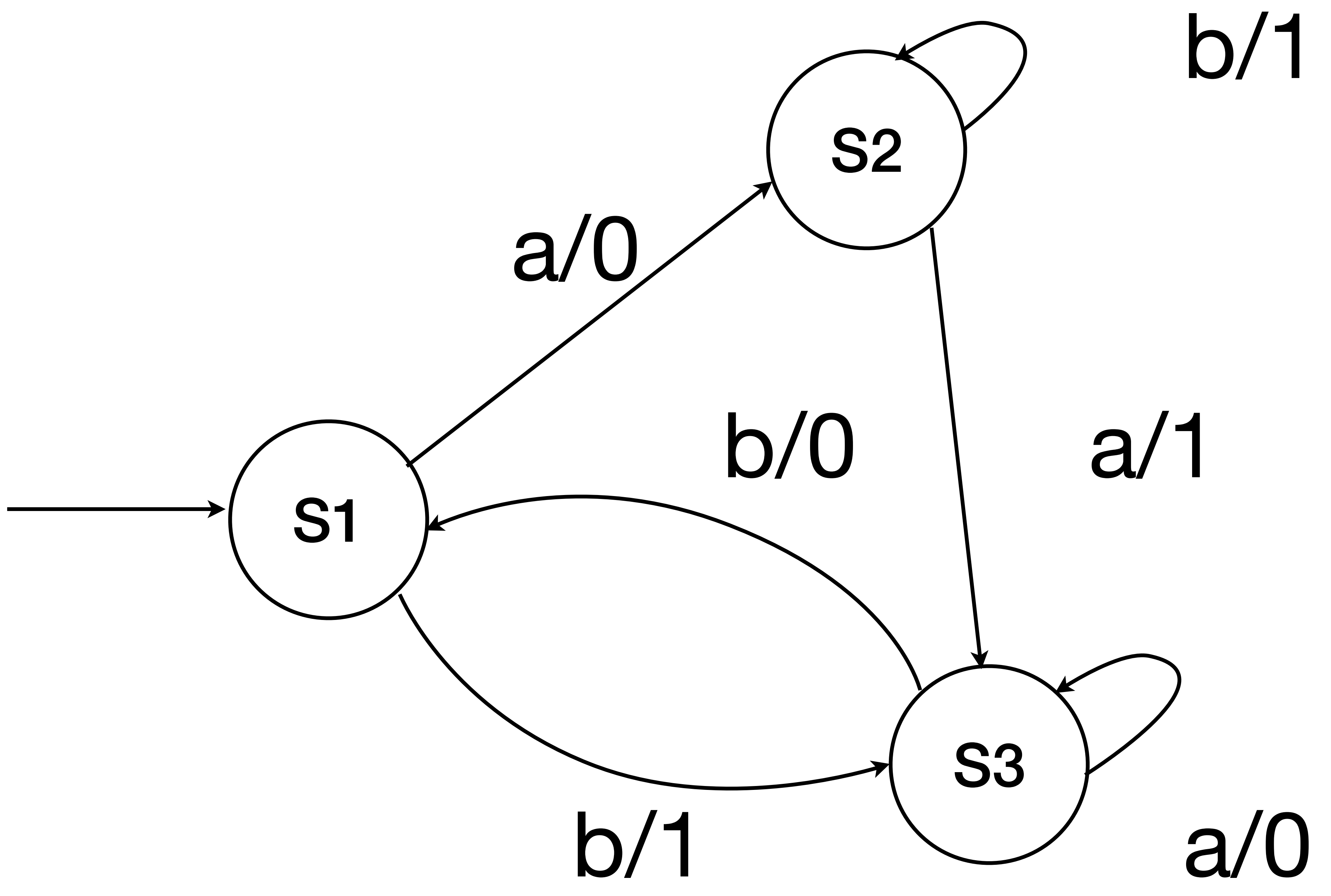
- A reset operation is one that always takes the FSM to the initial state.
- Sometimes we assume that there is a reliable reset operation: there is some reset operation that we know is correct.
- This helps in testing: we can use it to separate test sequences
- It may involve switching the machine off and then on again.

Further Assumptions

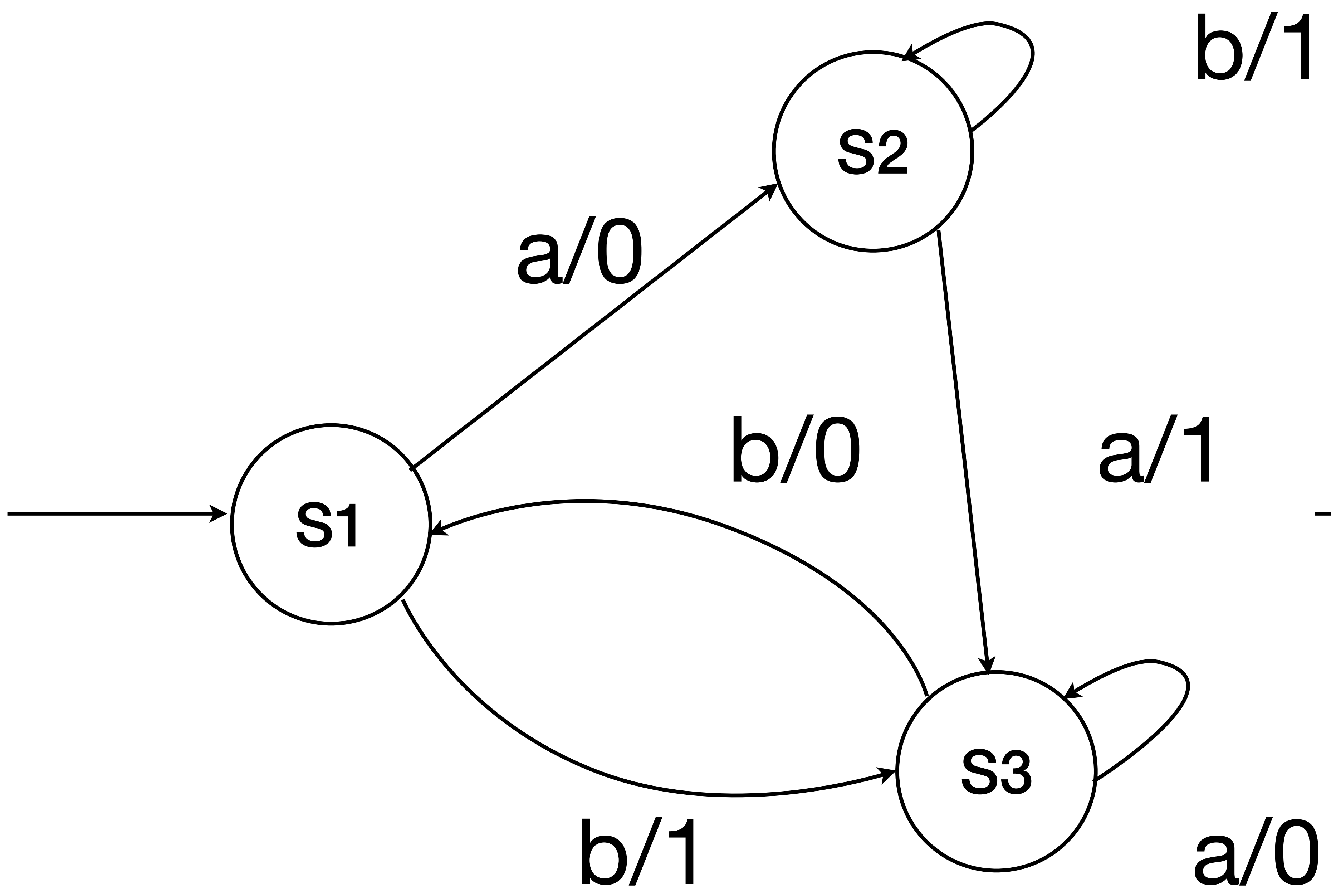
- It is normal to assume that M is minimal, strongly connected and completely specified.
- Often also we assume that there is some reset operation.
- These simplify test generation.

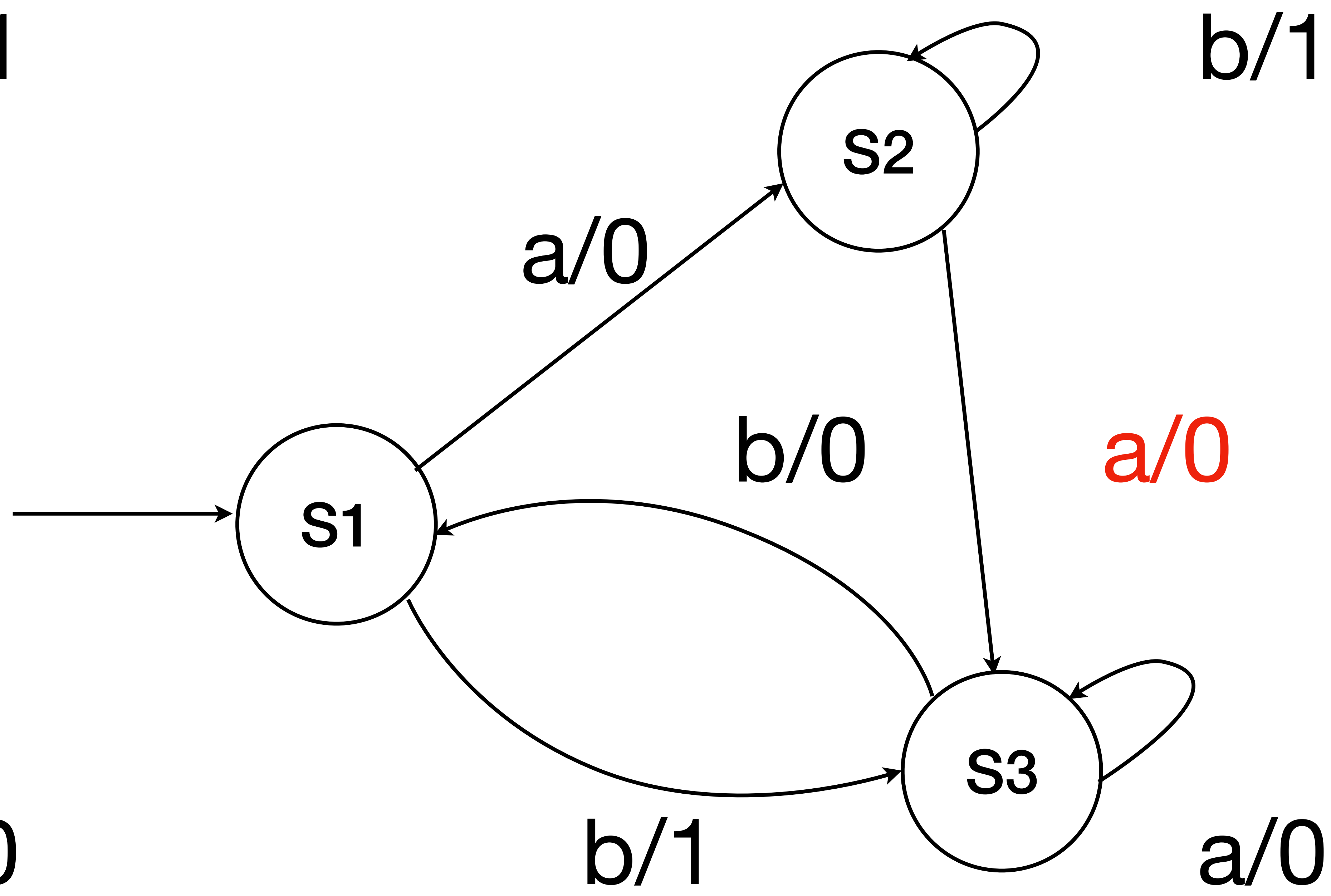
Faults and FSM

- There are two main classes of fault:
  - Output fault: a transition has the wrong output
  - State transfer fault: a transition goes to the wrong state
- Note: state transfer faults may lead to M’ having more states that M.

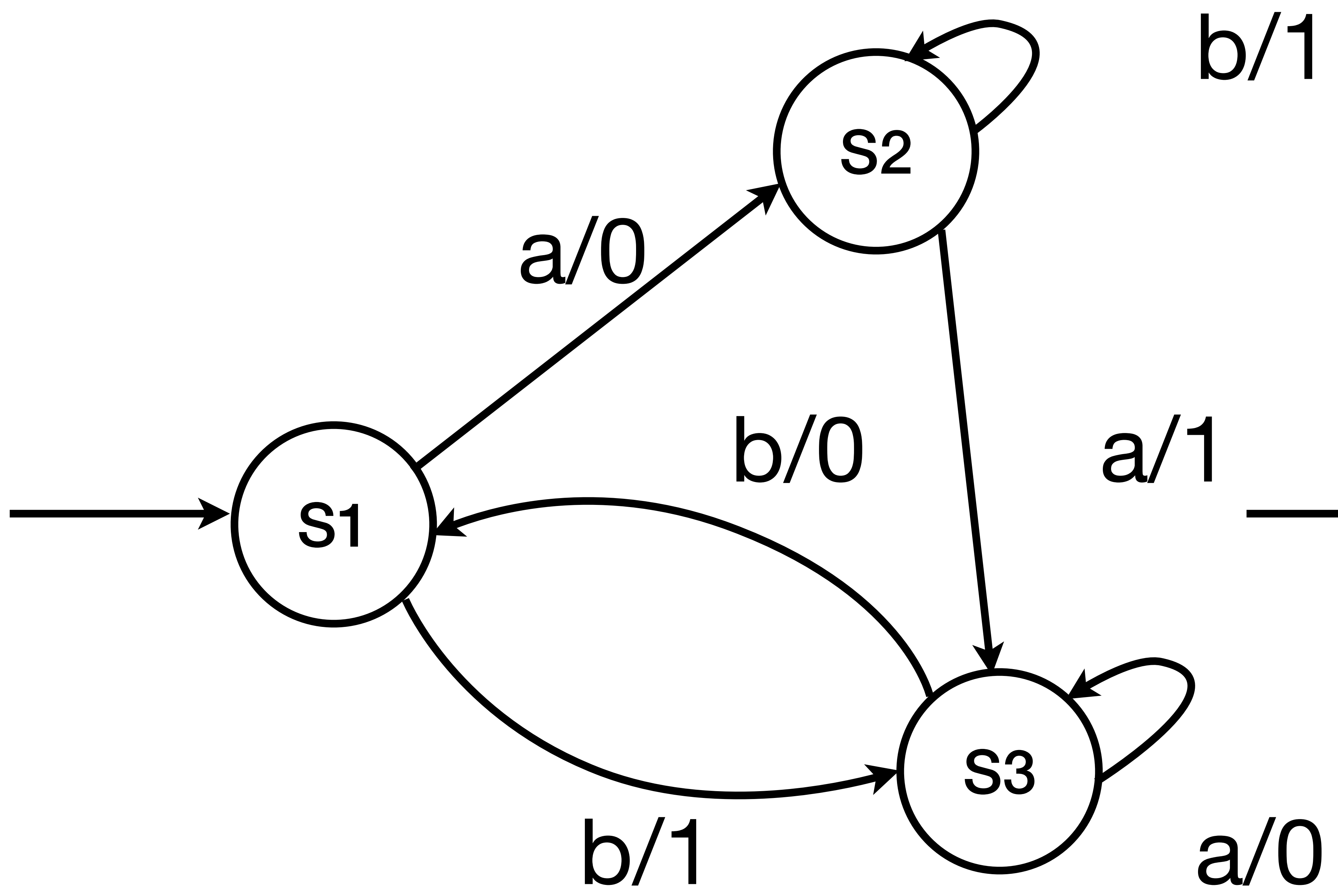


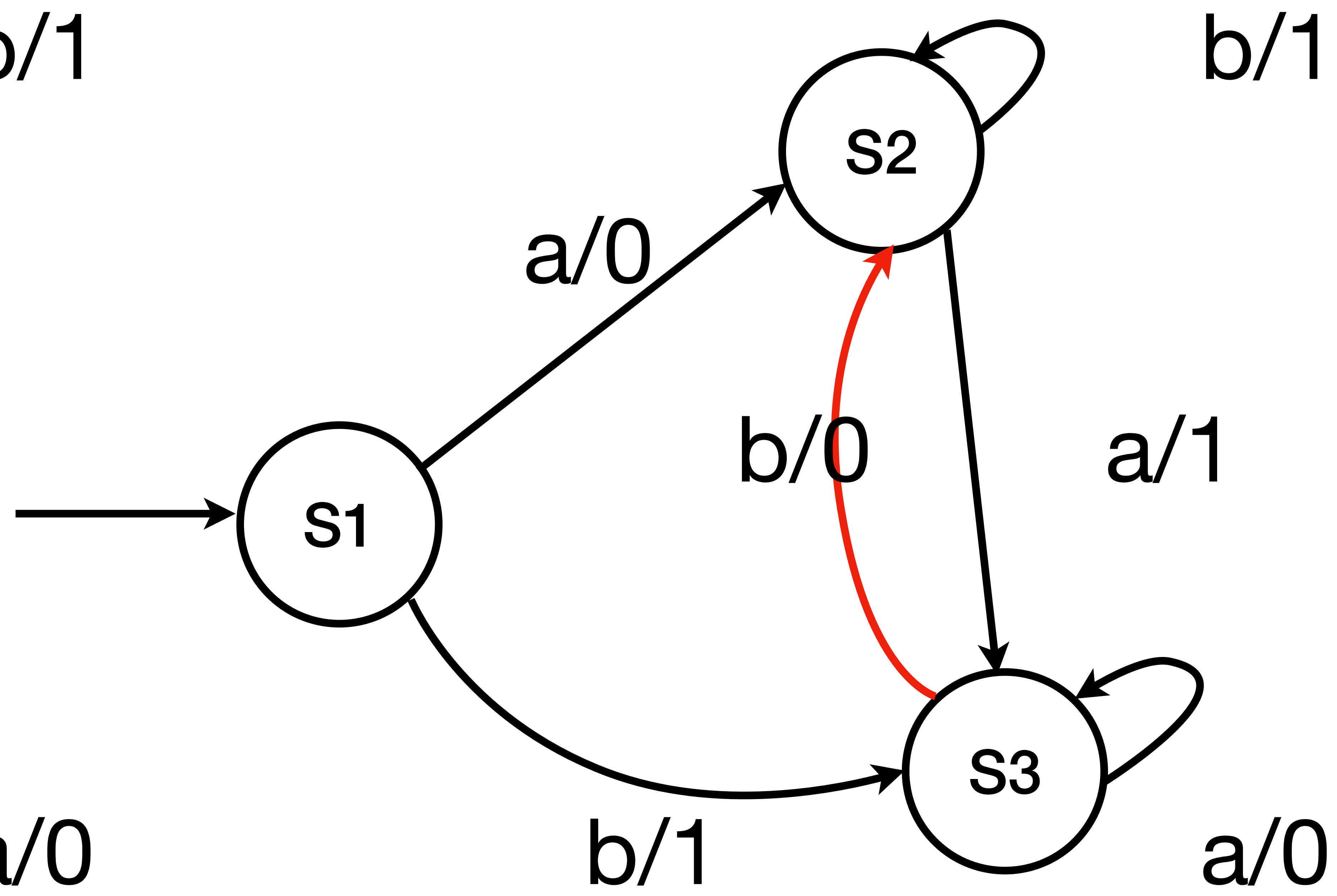
Output Faults



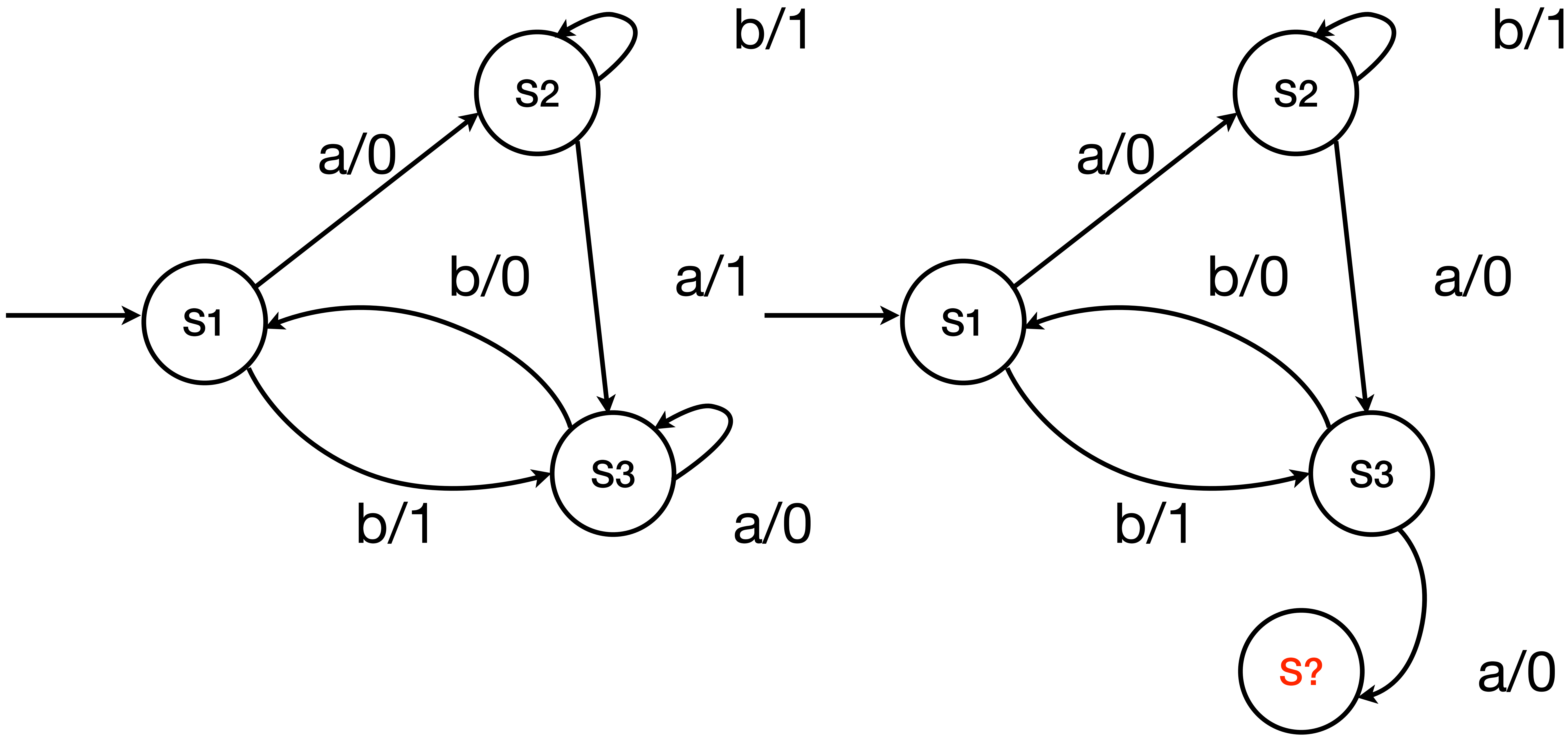


State Transfer Faults





State Transfer Faults



What do we need to do(know) to detect each type of fault?

Finding Output Faults

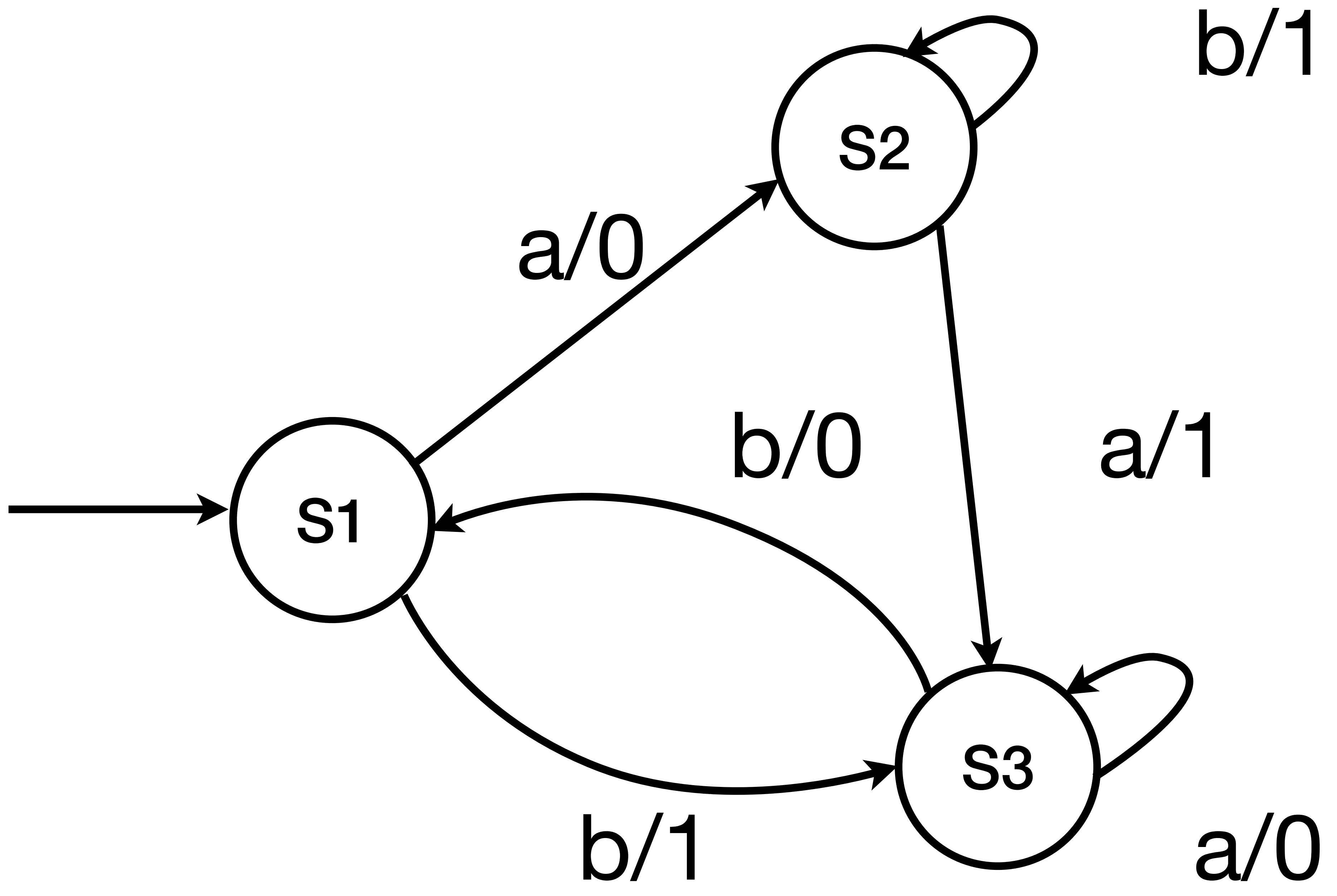
- To find output faults we just need to execute transitions.
- Transition tour method: generate a single sequence (a transition tour) that covers each transition.
- FSM is assumed to be fully specified: we just compare observed output to the specification!
- What do we need?
    - An input sequence that will take us through ALL transitions in the FSM

Transition Tour Method

- In the transition tour method we:
  - Find some path/walk, from the initial state, that covers every edge/transition.
  - Our test is the input sequence defined by following this sequence.
- This detects all output errors. However, there is no guarantee that all transfer errors can be detected.

Transition Tour Example

- We could follow the path with edges:
  - `a/0, b/1, a/1, a/0, b/0, b/1`
- This gives test sequence:
  - `abaabb`



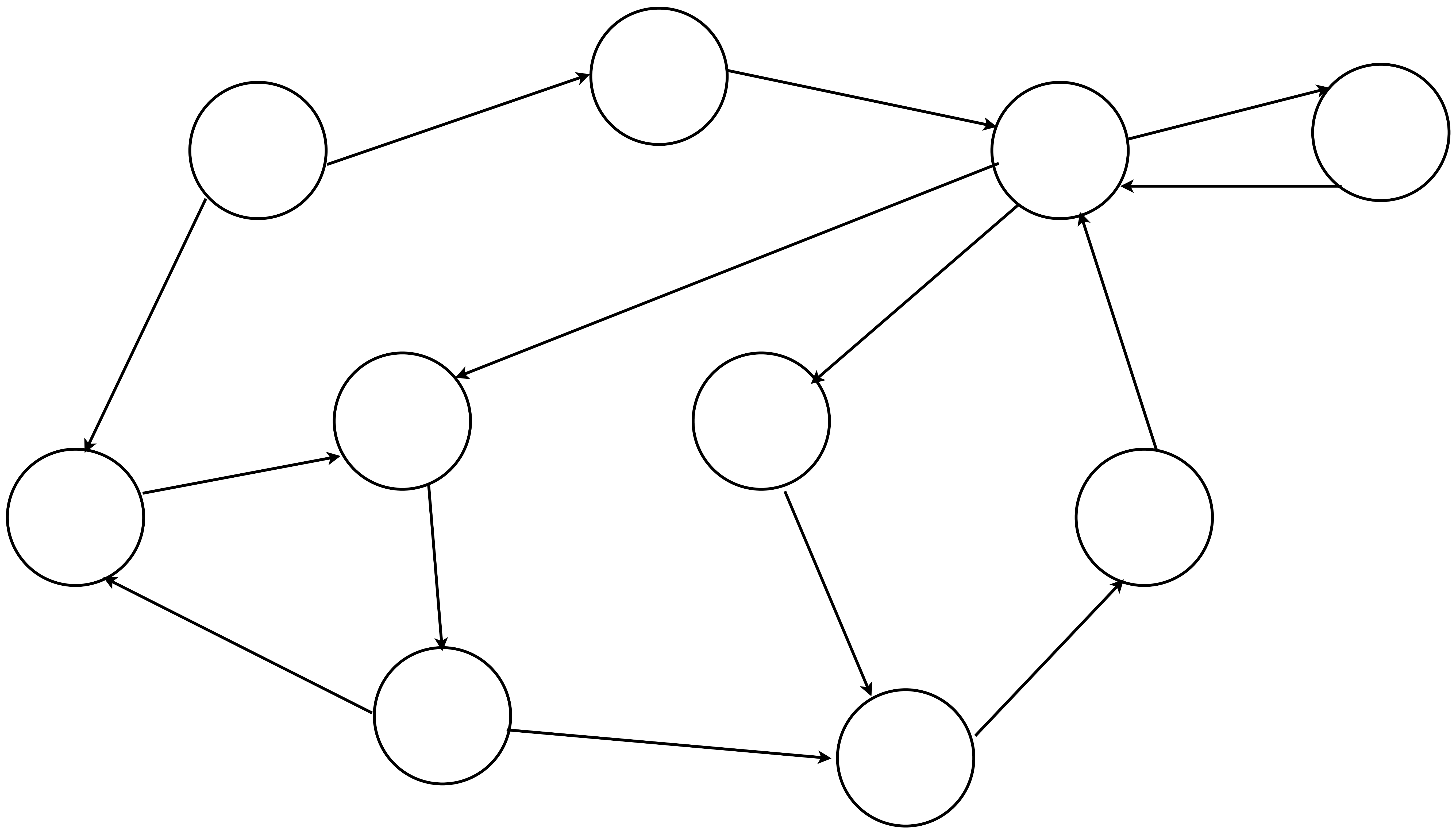
Generating a Transition Tour

- We can simply follow a path, at each step extending it by:
  - 1. Choosing an edge we have yet to take
  - 2. Adding a path from where we are to the source node of this edge
  - 3. Adding the edge (i.e. move to the target node of this edge)
- Note: there are also algorithms that produce minimal length transition tours.

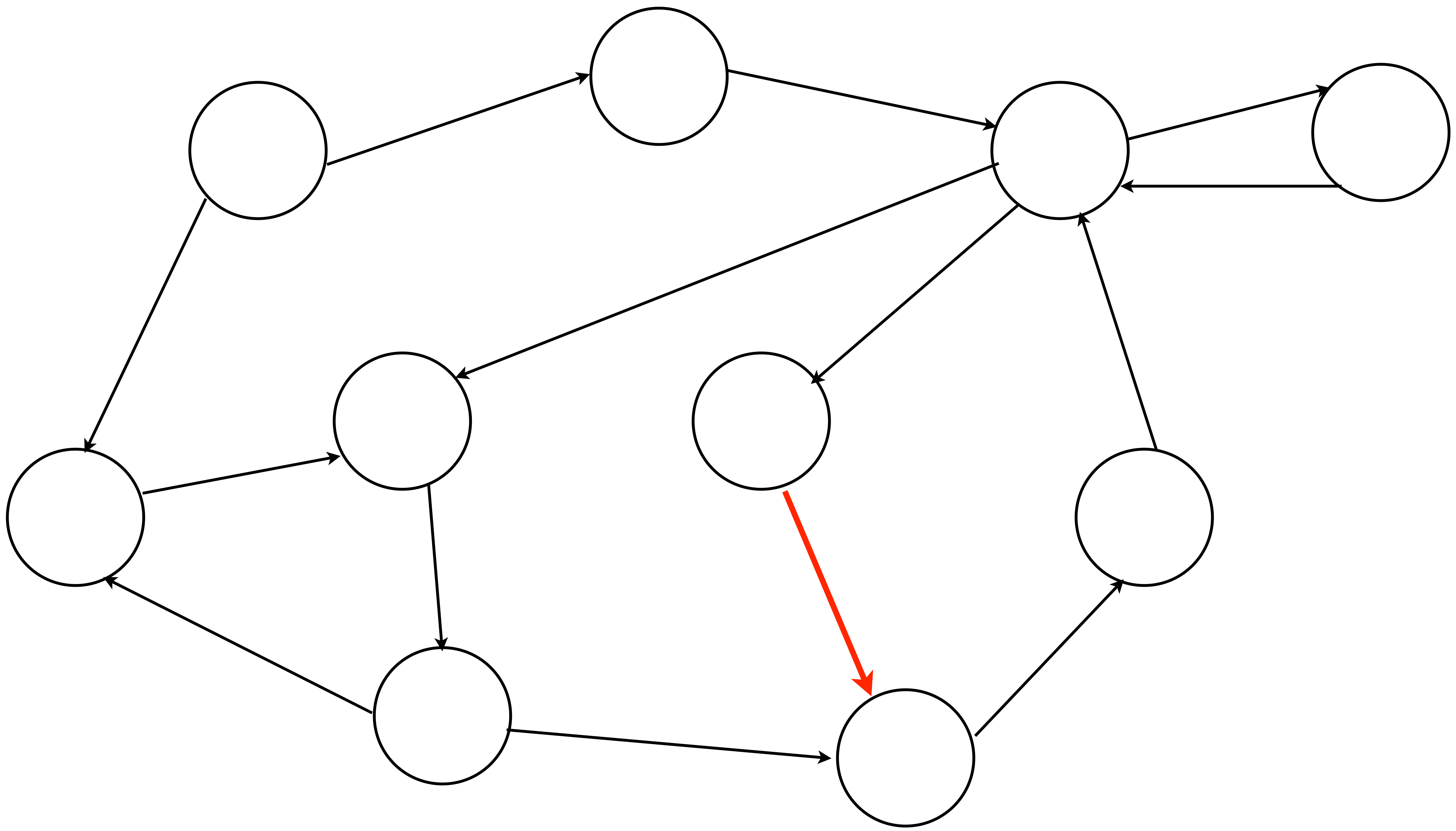
Finding State Transfer Faults

- We want to check whether a correct transition is followed
- What do we need to do in order to check this?
  - 1. Get to the start state of the transition
  - 2. Execute the transition
  - 3. Check the end state is the right one.

Example

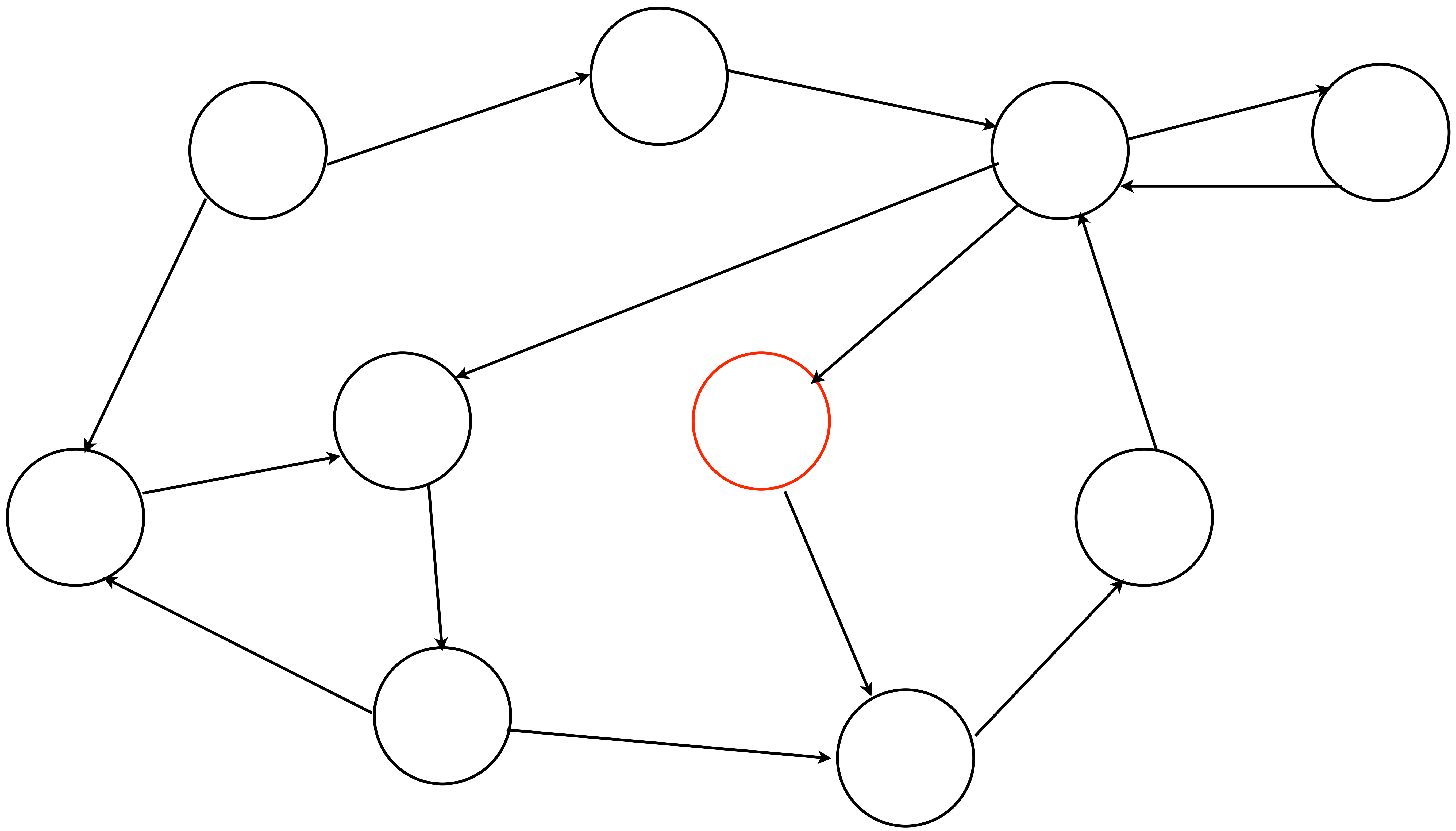


Example



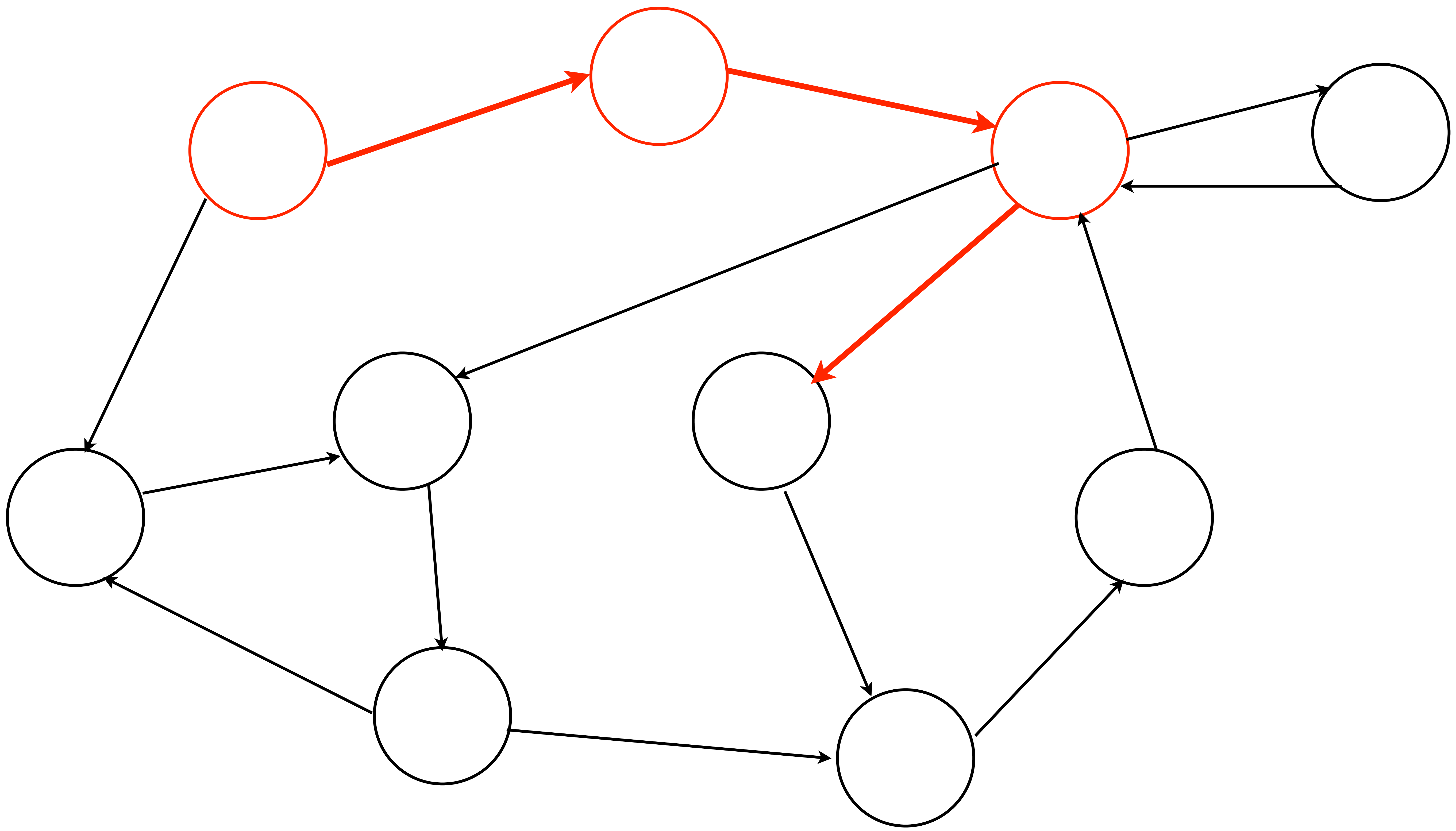
We want to test this transition

Example



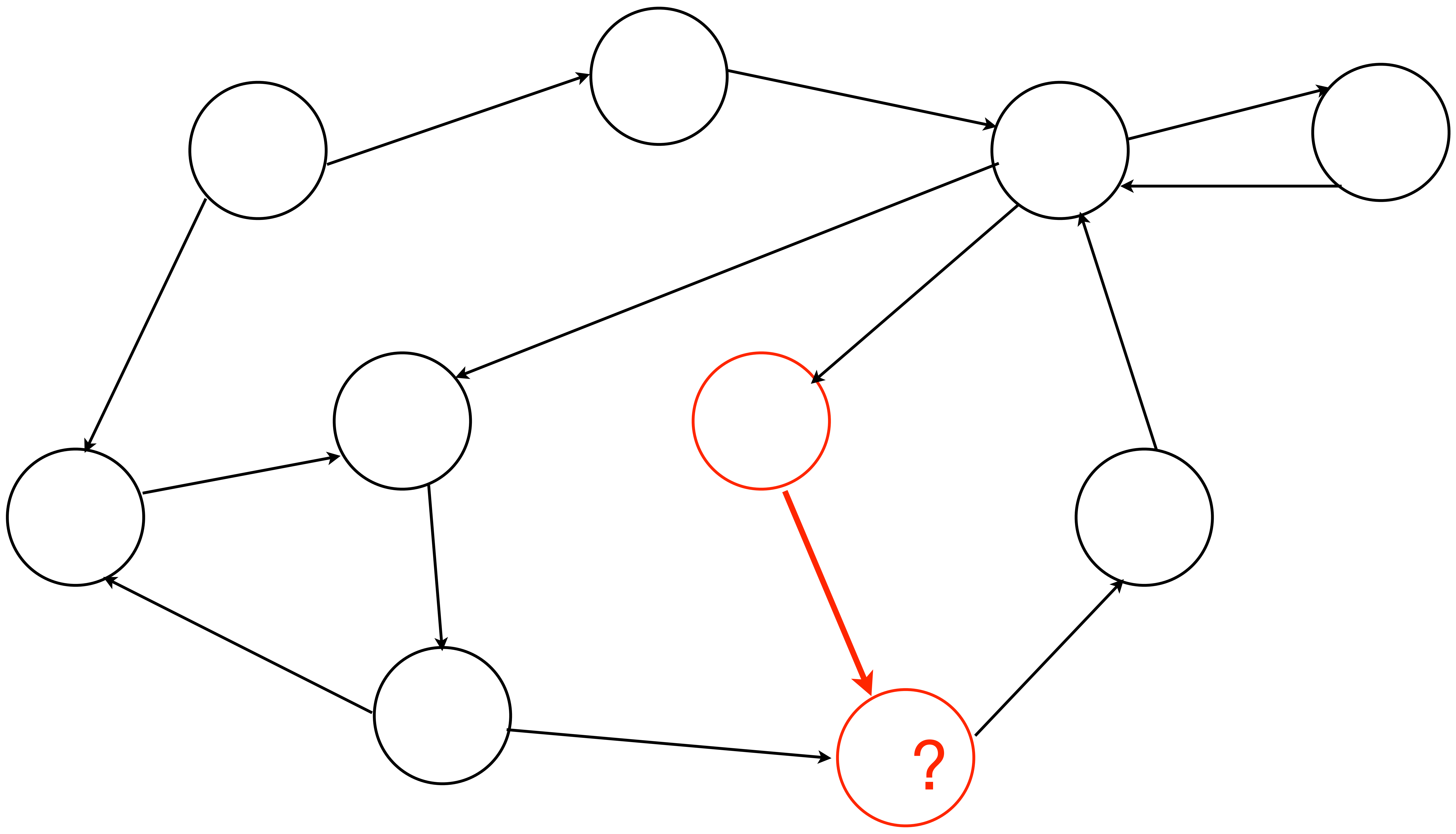
First get here.

Example



In other words, find the input sequence that does this

Example



Then we do this. The question is, are we at the right target node?

Checking State

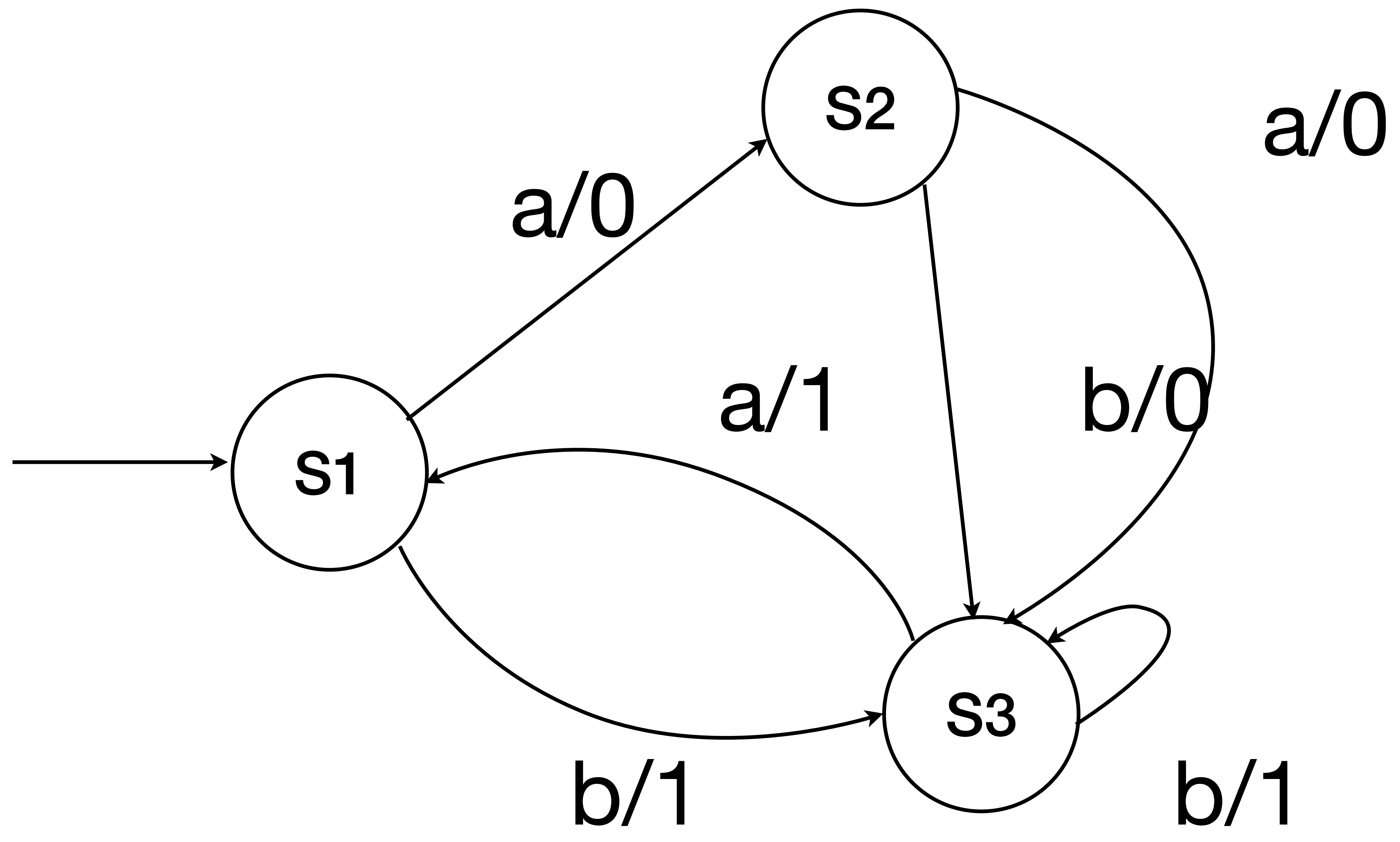
- It is crucial that we can determine the “current” state that we are in, simply based on the outputs of the FSM
- There are multiple techniques. In the increasing order of strength, we will learn about:
  - A distinguishing sequence
  - Unique Input/Output (UIO) sequences
  - A characterising set

Distinguishing Sequences

- An input sequence D is a distinguishing sequence if:
  - for every pair of states s, s' of M such that $s \neq s'$ we have that $\lambda^*(s, D) \neq \lambda^*(s', D)$.
  - That is, all states produce unique outputs in response to D: therefore we can identify the state.
- One sequence distinguishes all states; certain FSMs will NOT have a distinguishing sequence

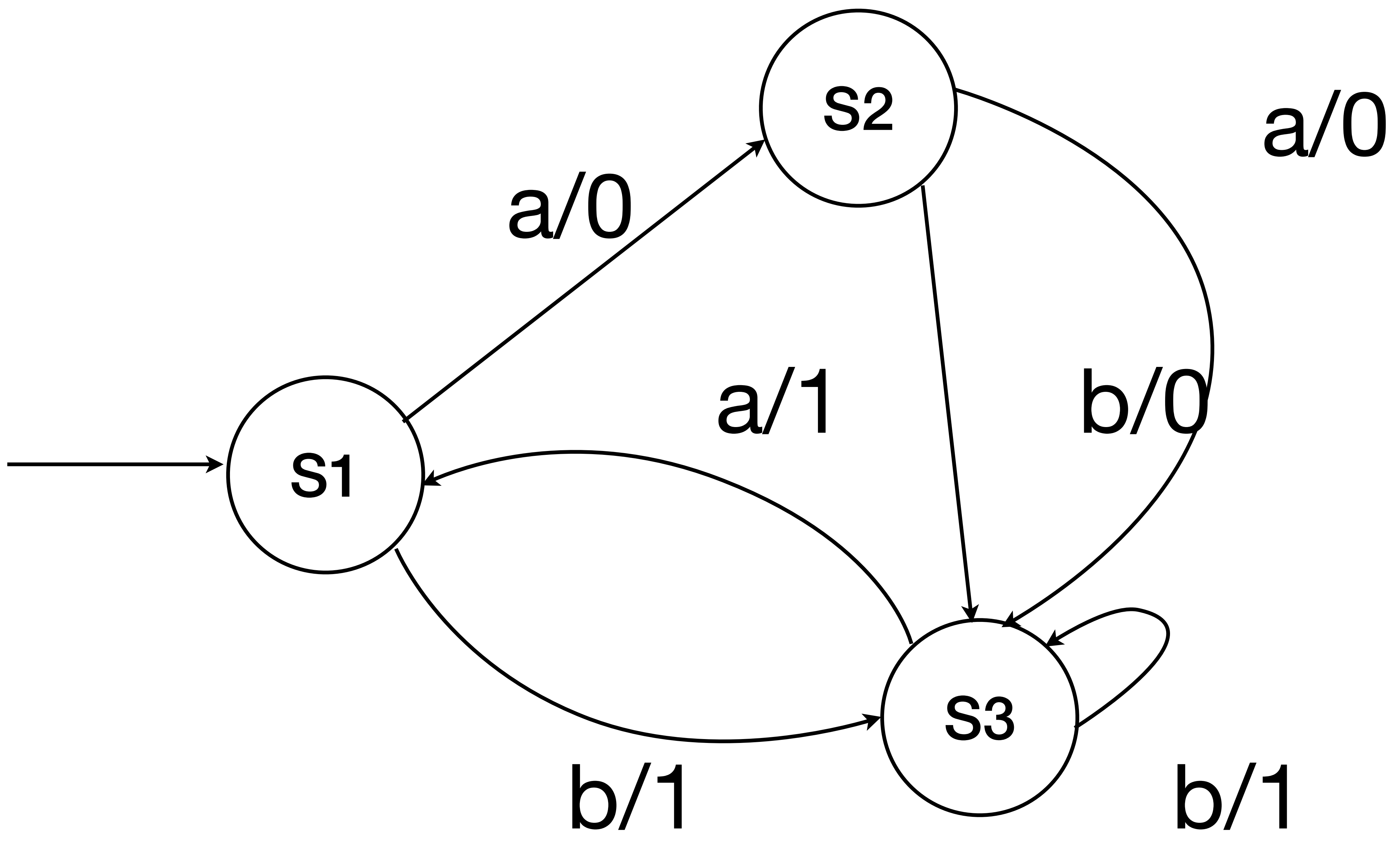
Distinguishing Sequence Example

- We can simply check different input sequences, producing a column in a table for each.
- Normally we start with short sequences and extend these.



Distinguishing Sequence Example

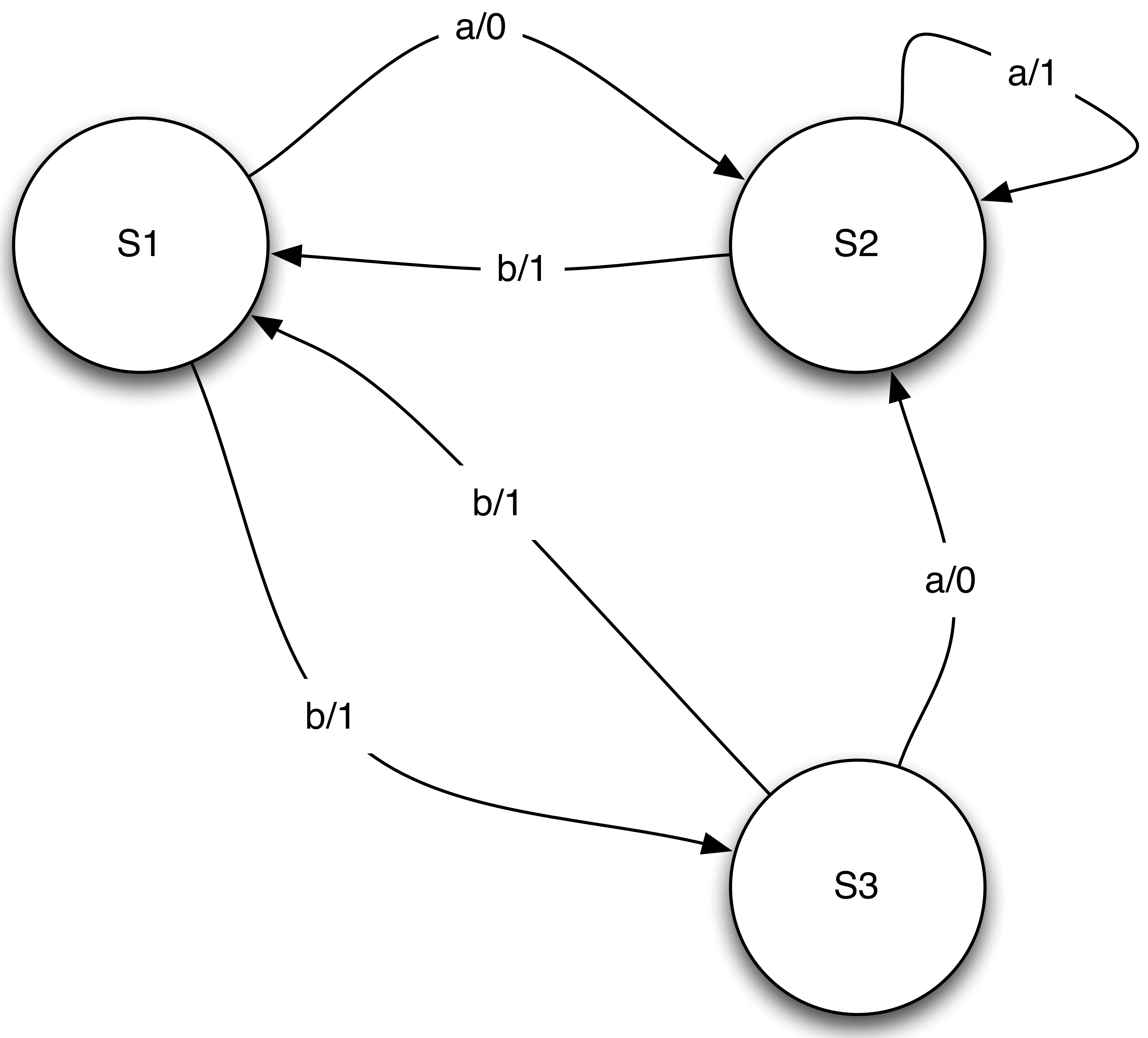
	a	b	ab	aa
S1	0	1	00	00
S2	0	0	01	01
S3	1	1	11	10



It is possible to have multiple distinguishing sequences.

Is it enough?

- Consider the machine on the right. Can it have a distinguishing sequence? If so, what is it? If not, why?



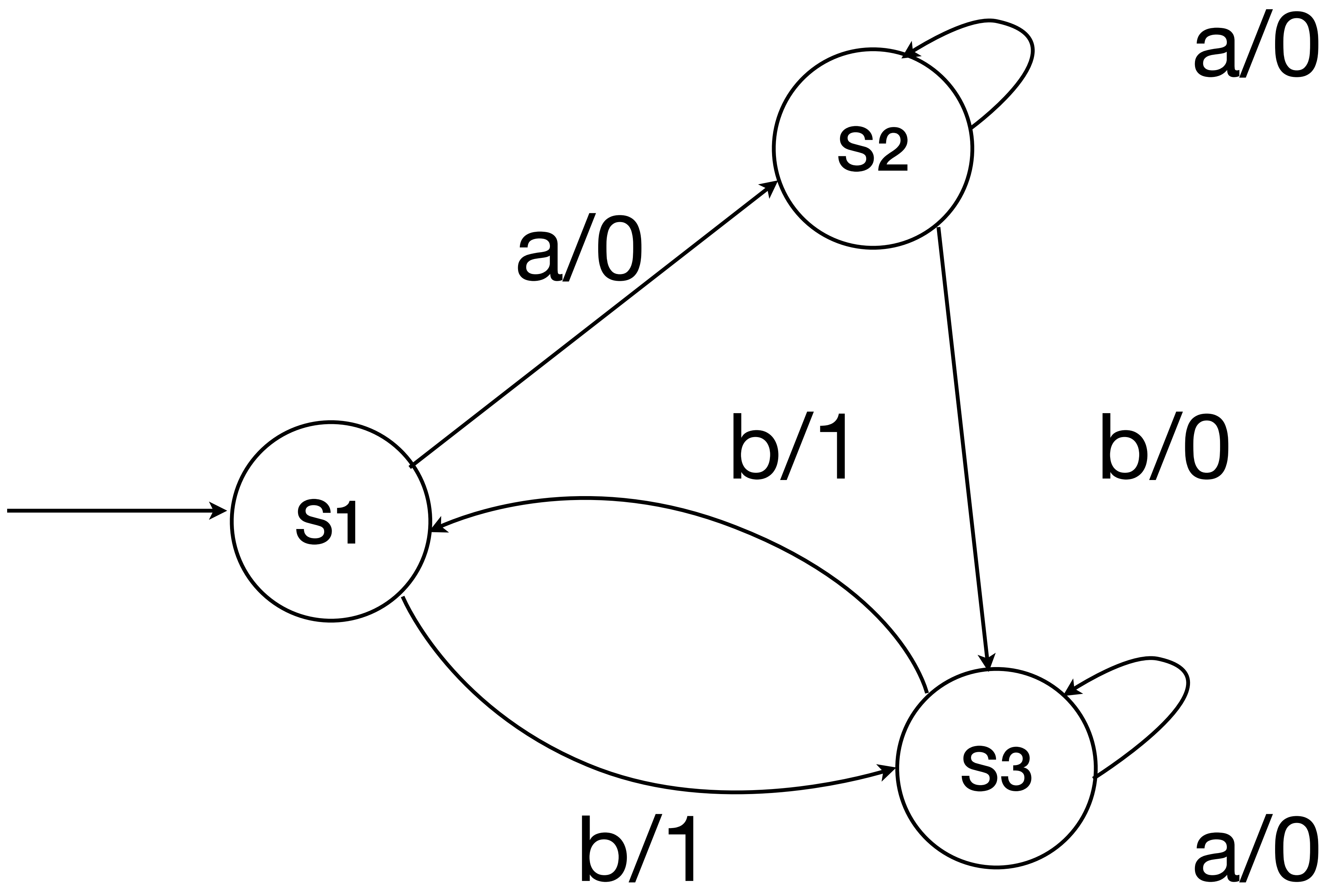
Unique Input/Output Sequences

- A sequence x/y is a unique input/output sequence (UIO) for state s if:
- $y = \lambda^*(s, x)$ and for every state s' of M such that $s \neq s'$ we have that $\lambda^*(s, x) \neq \lambda^*(s', x)$.
- This means that input x identifies the state s since: if y is produced in response to x we must have been in state s, otherwise we must have been in a different state.
- Thus, x is capable of verifying s in M but not necessarily any other state of M.

UIOs Example

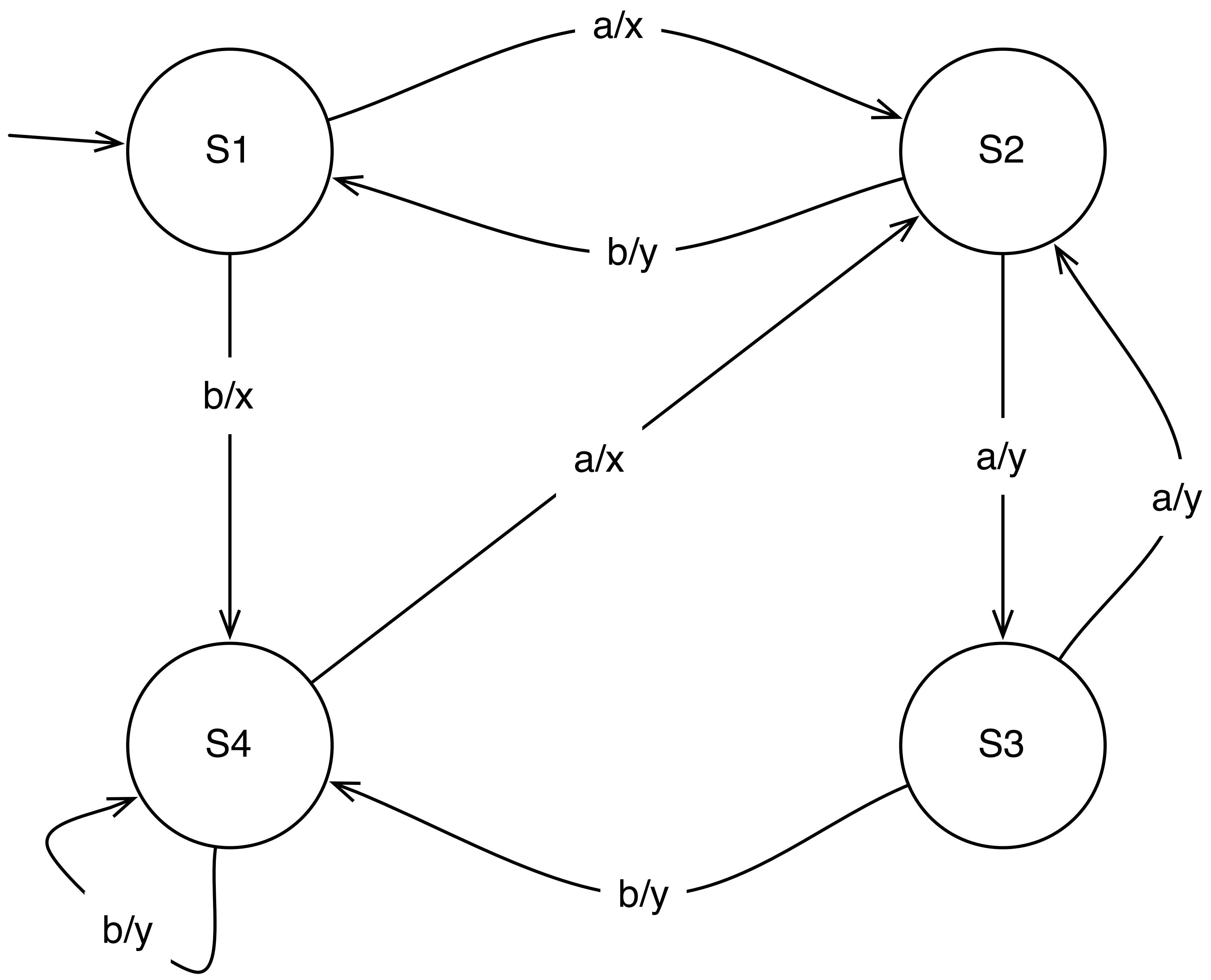
Find UIO for S3 ?

	a	b	ab	ba
S1	0	1	00	10
S2	0	0	00	00
S3	0	1	01	10



Is UIOs enough?

- Consider the machine on the right: does S4 have an UIO sequence? If so, what is it? If not, why?

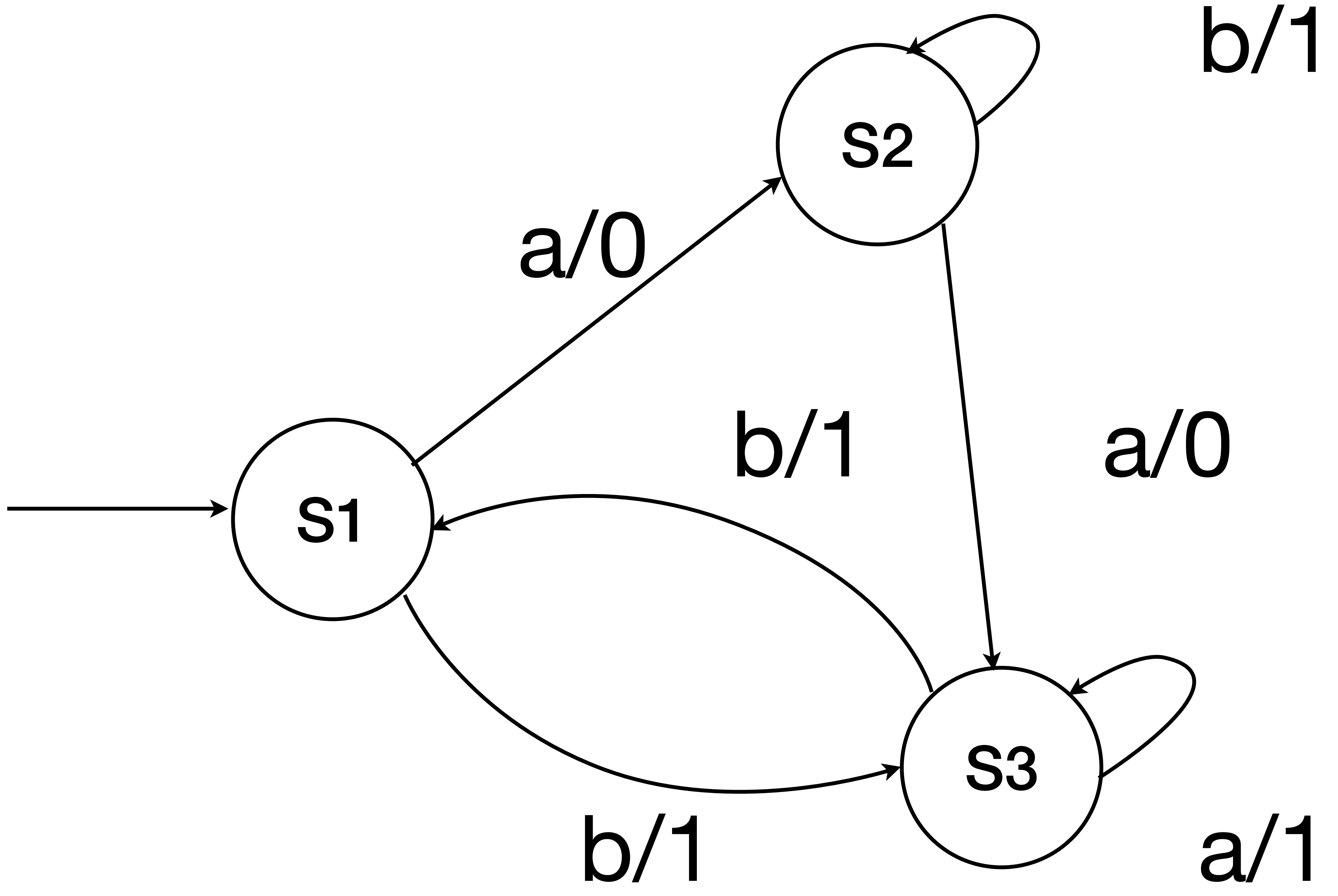


Characterising Set

- A set W of input sequences is a *characterising set* for M if:
  - for every pair of states s, s' of M such that $s \neq s'$ we have some $w \in W$ such that $\lambda^*(s, w) \neq \lambda^*(s', w)$.
- This means that, for each pair of states, there exists **at least one** input sequence from W that distinguishes them.
- Note: there is always a characterising set for a minimal FSM.

Characterising Set

	a	b	ab	ba
S1	0	1	01	11
S2	0	1	01	10
S3	1	1	11	10



Testing a Transition

- First, we check whether the source transition is reachable. That is, if we apply certain sequence, we arrive at the source state.
- Second, we check whether executing the transition from the source state takes us to the correct target state.
- How do we do this systematically?

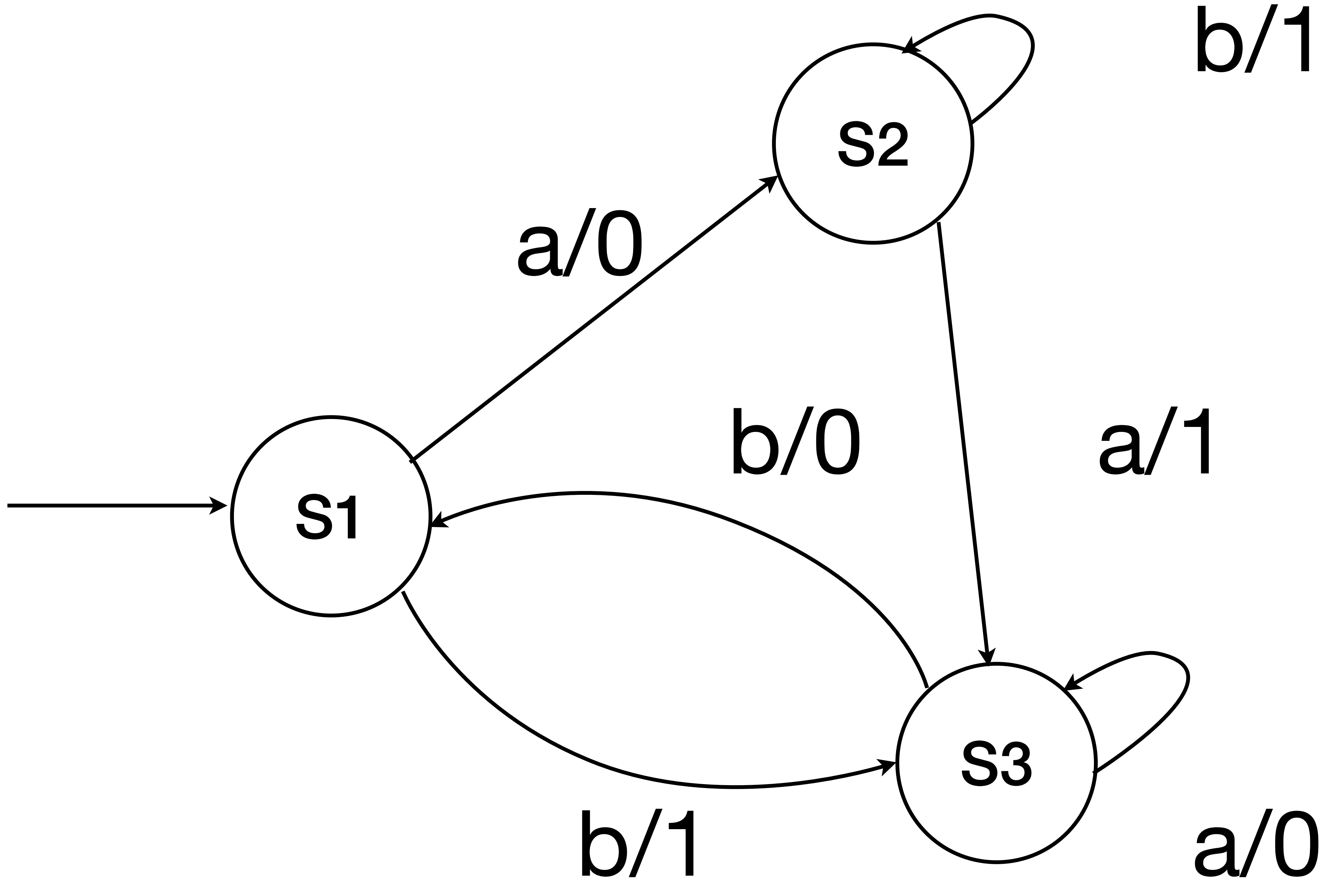
Chow’s Method

- This is based on using
  - a input set: X
  - a characterising set: W
  - a state cover set: V
  - a reliable reset, and a concatenation operator $\cdot$
- Resulting Test set : $V \cdot W \cup V \cdot X \cdot W$

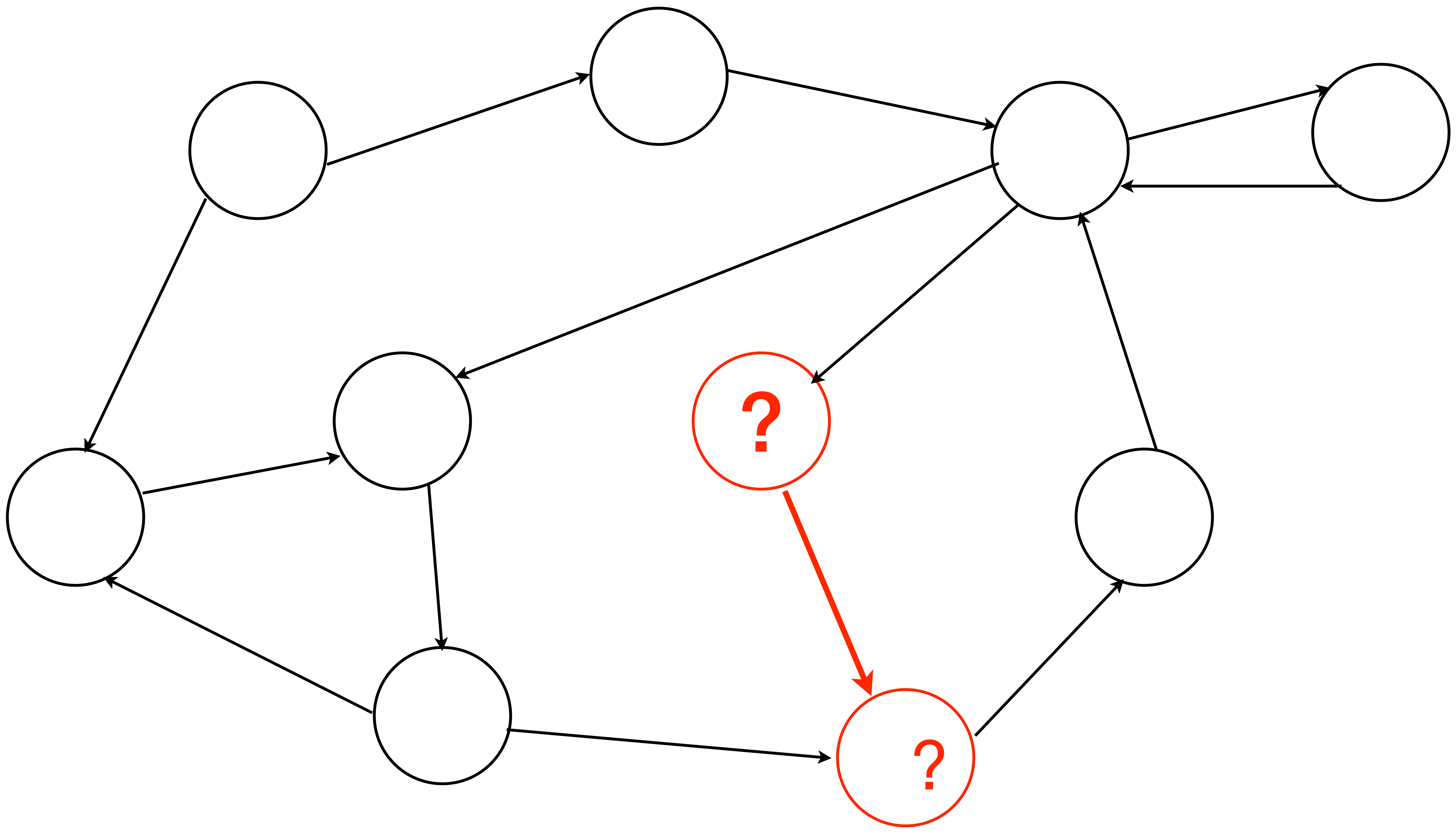
State Cover Set

- The State Cover Set V is a set of sequences such that each state of M is reached by a sequence from V.

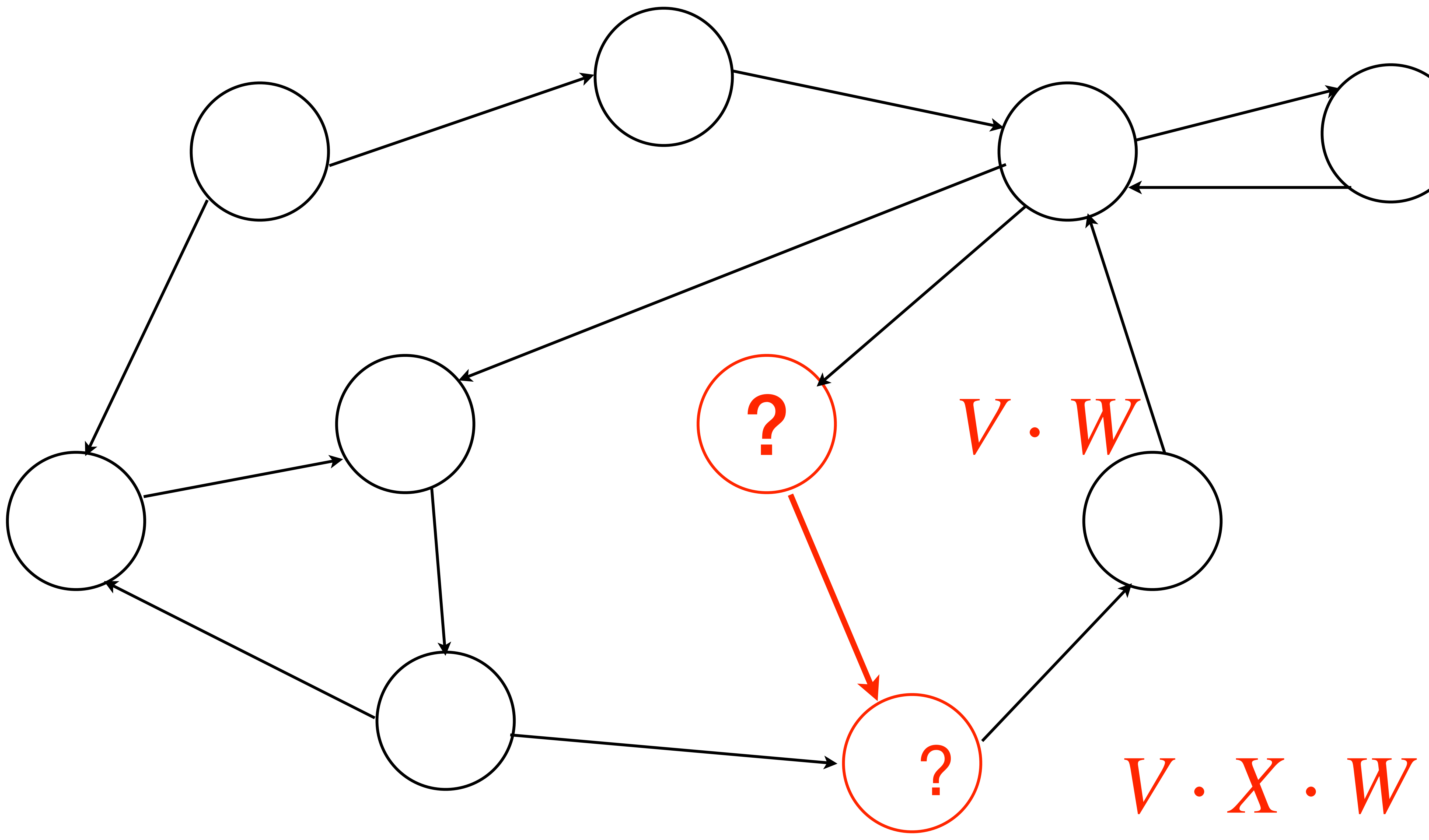
- For the machine on the right, $V = \{\epsilon, a, b\}$



Finding State Transfer Faults



Finding State Transfer Faults



Chow's Method Example

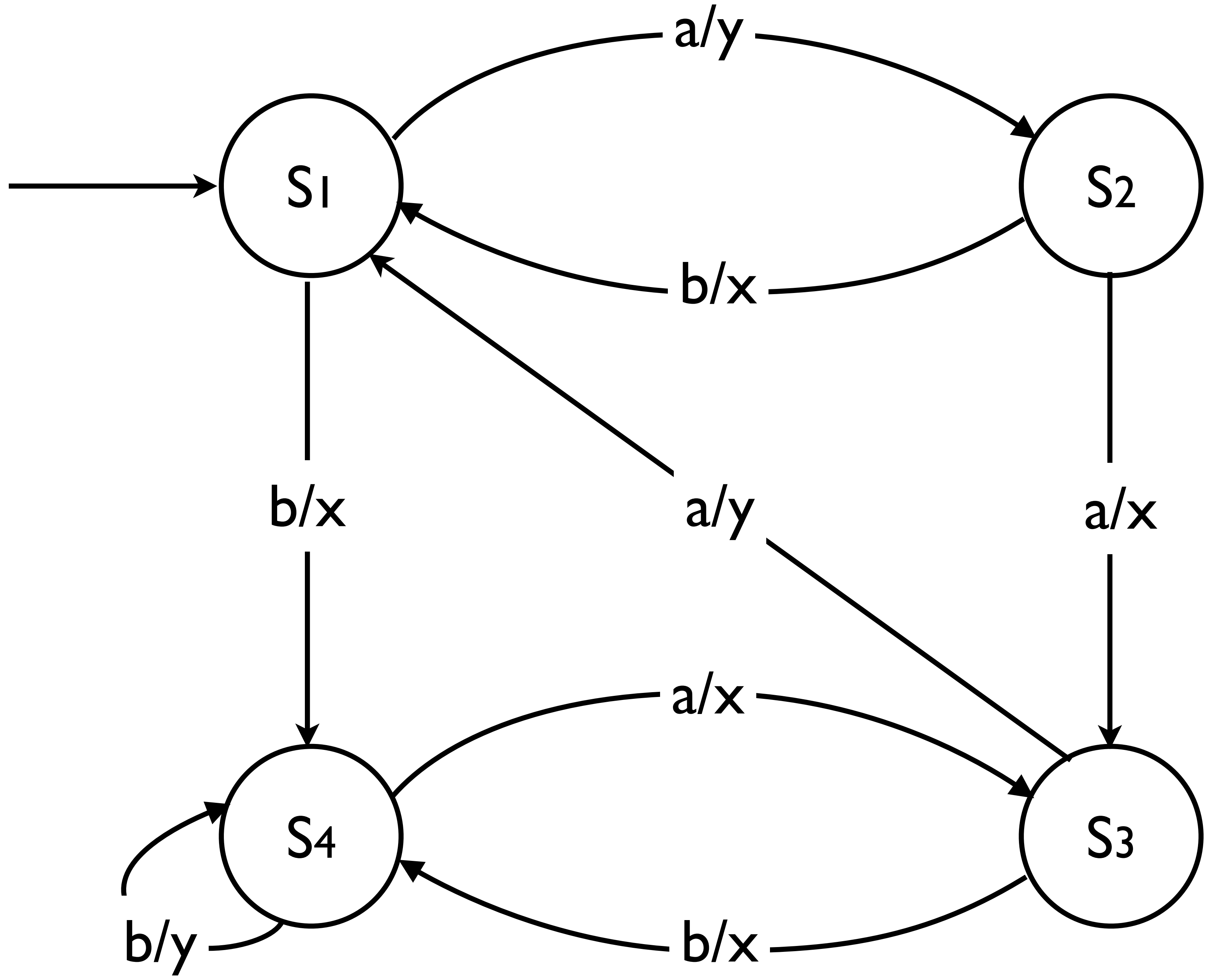
- Suppose we have $X = \{a, b\}, V = \{\epsilon, a, b\}$ and $W = \{a, b\}$.
  - We need $V \cdot W \cup V \cdot X \cdot W$: $\{\epsilon, a, b\} \cdot \{a, b\} \cup \{\epsilon \,.\, a \,.\, b\}\{a, b\}\{a, b\}$
  - Therefore, we get $\{a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb\}$
- we can remove some tests: those that are prefixes of others.

Summary

- Testing output faults:
  - Transition Tour
- Testing transition faults
  - Distinguishing Sequence
  - Unique Input/Output Sequences
  - Characterising Set
- Further readings:
  - D. Lee and M. Yannanakis. Principles and Methods of Testing: Finite State Machines - A Survey. Proceedings of the IEEE, 84(8):1090--1123, 1996.
  - http://people.brunel.ac.uk/~csstrmh/research/fsm_testing.html

FSM Exercise

- Generate an input sequence that achieves the transition tour.
- Generate UIO for each state.



FSM Exercise

- Can you generate UIOs for all states in the following FSM? If so, generate one for all. If not, explain why.