

Laurea Specialistica in Ingegneria dell'Automazione

Sistemi in Tempo Reale

Luigi Palopoli

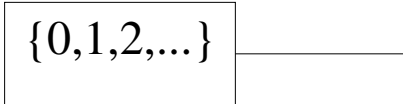
email: palopoli@sssup.it Tel. 050 883444

Selected topics on discrete-time and sampled-
data systems

Outline

- Finite State Machines (FSMs)

The concept of SM

- SM are systems operating on sequences:
 - $F: \text{InputSignals} \rightarrow \text{OutputSignals}$
where both input signals and output signals have the form:
 - EventStream: $\text{Naturals}_0 \rightarrow \text{Symbols}$

 - This definition captures the concept of *ordering* between events
 - Reference to time (discrete or continuous) is neither explicit nor implied
- Crucial is the definition of a *state* that summarises the past story of the system

Example



$$x \in \text{InputSignals} = [Nats_0 \rightarrow \{0, 1\}]$$

$$y \in \text{OutputSignals} = [Nats_0 \rightarrow \{True, False\}]$$

$$\text{Recognizer}(x)(n) = \begin{cases} True & \text{if } x(n)=0 \wedge x(n-1)=1 \wedge x(n-2)=0 \\ False & \text{otherwise} \end{cases}$$

Example:

$$x = \{0, 0, 0, 1, 0, 0, \dots\} \rightarrow y = \{-, -, False, False, True, False, \dots\}$$

A formal definition

A state machine has several components:

States Set of possible states (called **state space**)

Inputs Set of possible input elements (called **input alphabet**)

Outputs Set of possible output elements (called **output alphabet**)

update: States $\times$ Inputs $\rightarrow$ States $\times$ Outputs the **update function**

The update function defines the new state and output given the current state and input.

initialState The **initial state**

Thus, a state machine can be described as a 5-tuple:

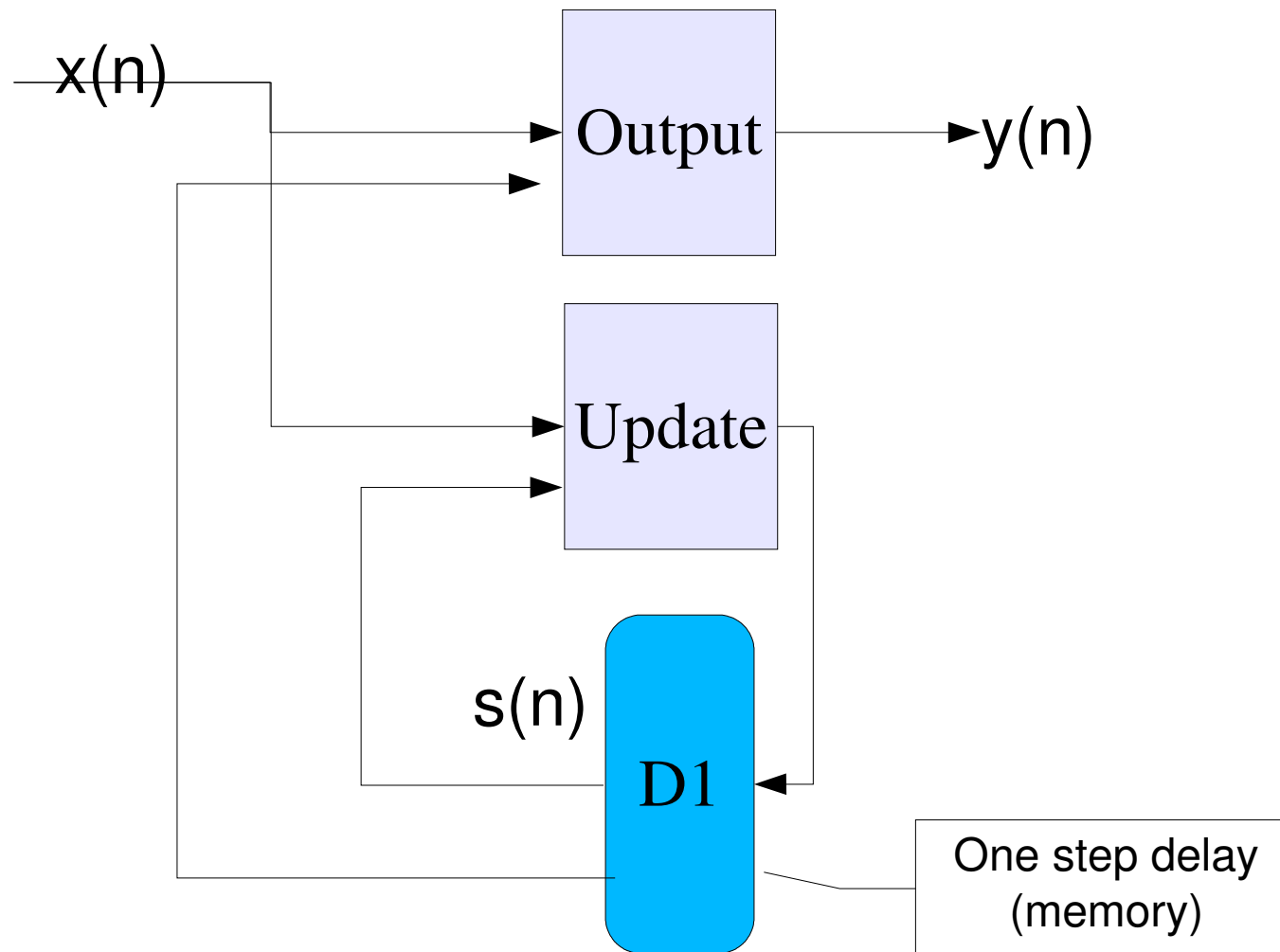
(States, Inputs, Outputs, update, initialState)

Update

- SM are causal: output depends only on the current state (that summarises the past history of the system) and on the current inputs
- The update dunction is often split into two functions:

$$\begin{aligned} \textit{nextState} : \textit{States} \times \textit{Inputs} &\rightarrow \textit{States}, & s(n+1) &= \textit{nextState}(s(n), x(n)) \\ \textit{output} : \textit{States} \times \textit{inputs} &\rightarrow \textit{outputs} & y(n) &= \textit{output}(s(n), x(n)) \end{aligned}$$

Conceptual scheme



Stuttering

We define a special “do nothing” input called *absent*.

When $x(n) = \textit{absent}$,

- the state does not change, so $s(n+1) = s(n)$;
- the output is also “nothing”, i. e. $y(n) = \textit{absent}$.
- We can always add absent symbols to input sequences without changing non absent outputs

This symbol, *absent*, is called the **stuttering** symbol.

Note that this symbol is always a possible input and output, so

$$\textit{absent} \in \textit{Inputs} , \textit{absent} \in \textit{Outputs}$$

Example

The Parity System

$Parity : [\mathcal{N}ats_0 \rightarrow \mathcal{B}ools] \rightarrow [\mathcal{N}ats_0 \rightarrow \mathcal{B}ools]$

such that $\forall \chi \in [\mathcal{N}ats_0 \rightarrow \mathcal{B}ools], \forall y \in \mathcal{N}ats_0,$

$$(Parity(\chi))(y) = \begin{cases} true & \text{if } | trueValues(\chi, y) | \text{ is even} \\ false & \text{if } | trueValues(\chi, y) | \text{ is odd} \end{cases}$$

where $trueValues(\chi, y) = \{ z \in \mathcal{N}ats_0 \mid z < y \wedge \chi(z) = true \}$

Example - (continued)

State-Machine Implementation of the Parity System

State after i-th input :

true, if first i inputs contain even number of true's

false, if first i inputs contain odd number of true's

Two states

Example - (continued)

Inputs = Bools, Outputs = Bools

States = Bools

initialState = true

nextState : $States \times Inputs \rightarrow States$

such that $\forall q \in States, \forall \chi \in Inputs,$

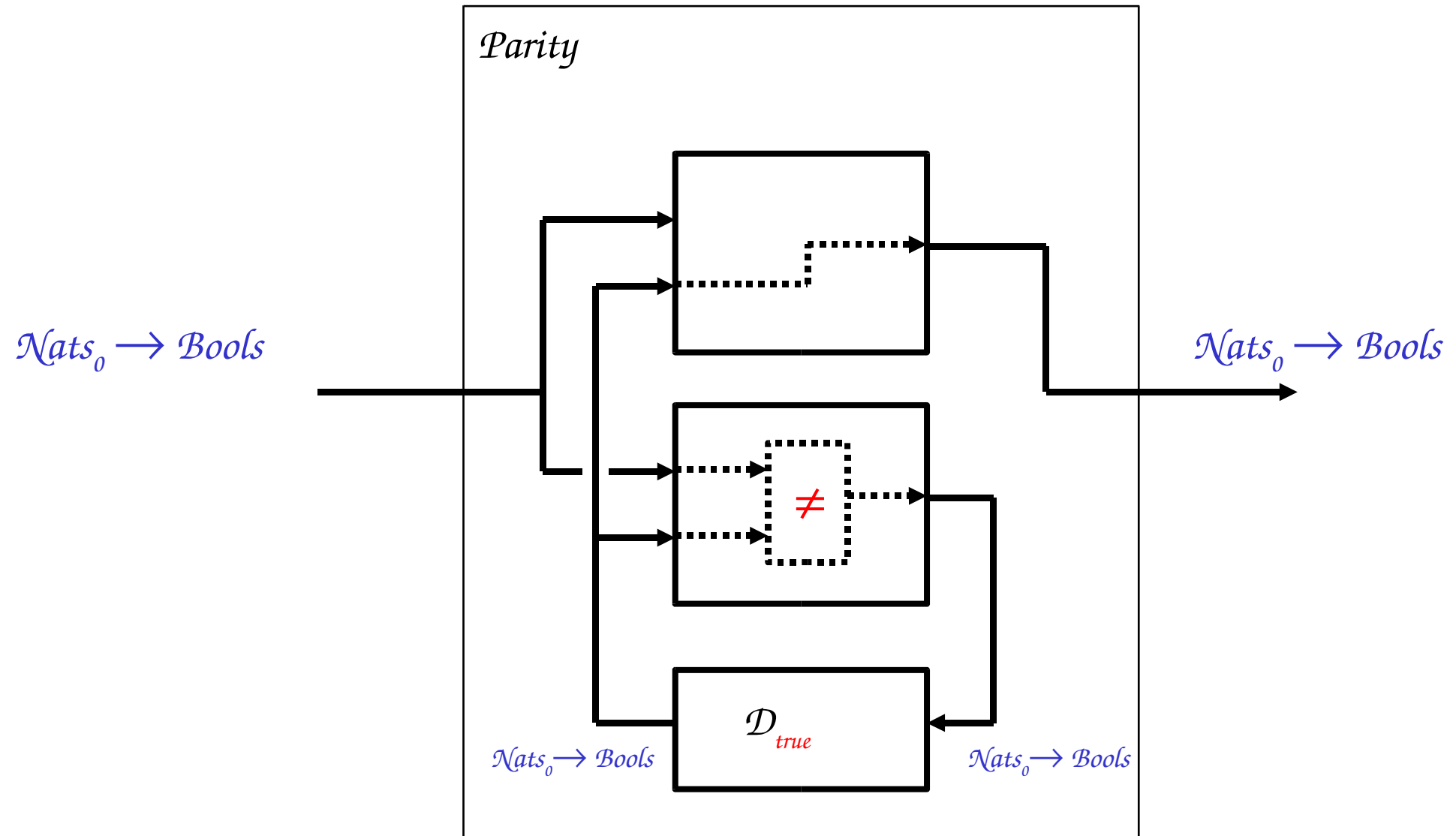
$$nextState(q, \chi) = (q \neq \chi)$$

output : $States \times Inputs \rightarrow Outputs$

such that $\forall q \in States, \forall \chi \in Inputs,$

$$output(q, \chi) = q$$

Example - (continued)



FSM

- Both examples shown above have a common features: **A finite number of states**
- For this reason they are called Finite State Machines
- We will also restrict to the case of finite input and output alphabet

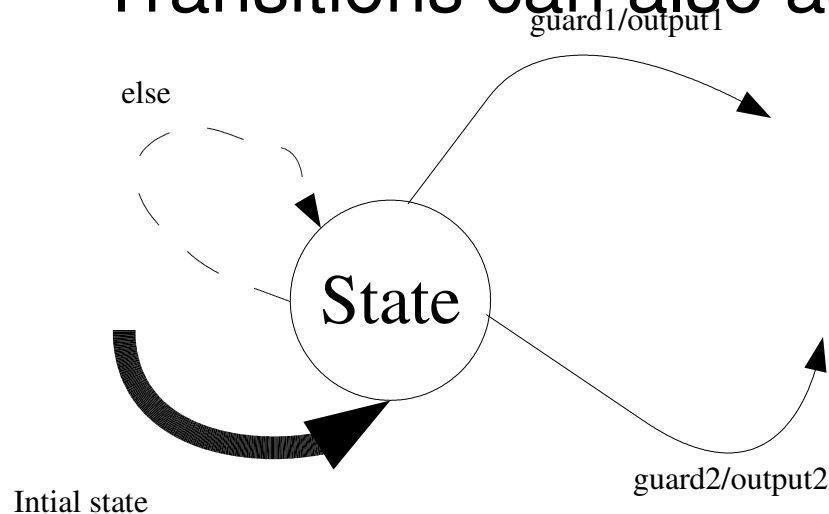
Update tables

- Update functions can be specified using tables
- Parity example:

$(s(n+1), y(n)) = \textit{update}(s(n), x(n))$		
	$x(n) = \textit{True}$	$x(n) = \textit{False}$
$s(n) = \textit{True}$	$(\textit{False}, \textit{True})$	$(\textit{True}, \textit{True})$
$s(n) = \textit{False}$	$(\textit{True}, \textit{False})$	$(\textit{False}, \textit{False})$

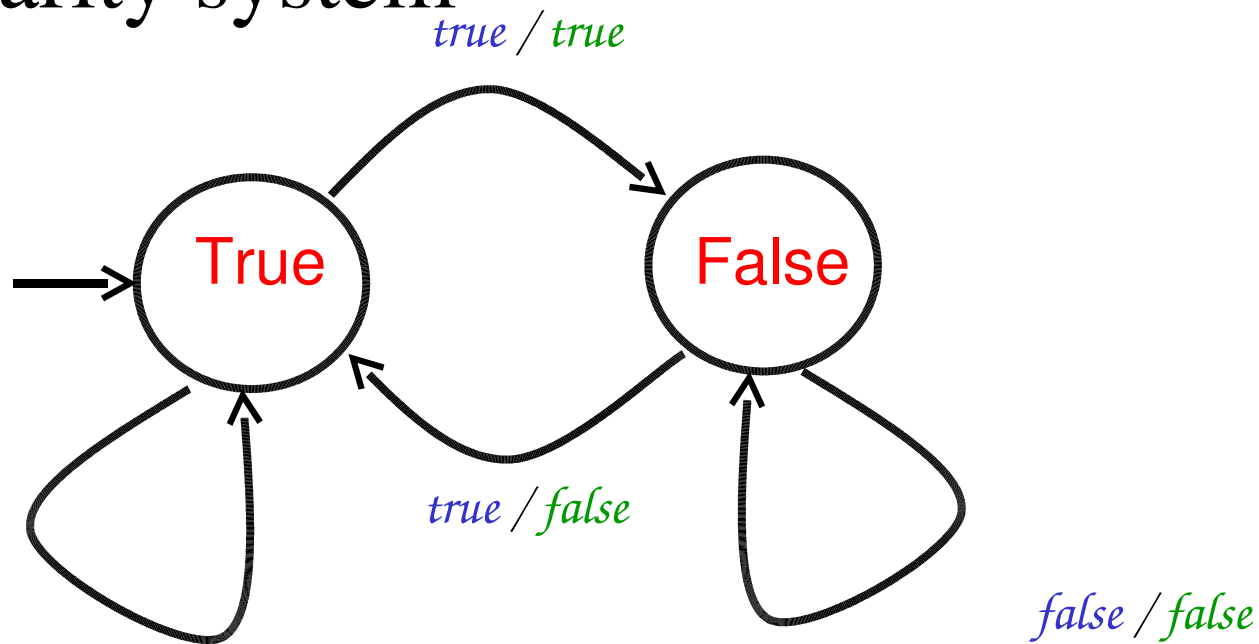
State transition diagrams

- State transition diagrams are a popular way for representing state machines
 - Bubbles represent states
 - Directed Arcs represent transitions
 - Transition are enabled by a guard
 - Guards are collections of input symbols
 - Transitions can also associated to outputs



Example

The parity system



States = *Bools*

Inputs = *Bools*

Outputs = *Bools*

Shorthand

- If no guard is specified the transition is always taken (except for the stuttering symbol)
- If no output symbol is specified then the output is the stuttering symbol
- If no else transition is specified then we assume a self loop
- If inputs are $\{a,b,c,d,\dots,z\}$ we can use $\{\text{not } a\}$ in a guard instead of $\{b,c,\dots,z\}$

Excercise

Draw the transition diagram of a state machine with

Inputs = Bools

Outputs = Bools

At all times t , the output is true iff the inputs at times $t-2$, $t-1$, and t are all true .

Exercise - (continued)

State after i -th input :

0, if i -th input is false (or $i = 0$)

1, if i -th input is true and $(i-1)$ -st input is false

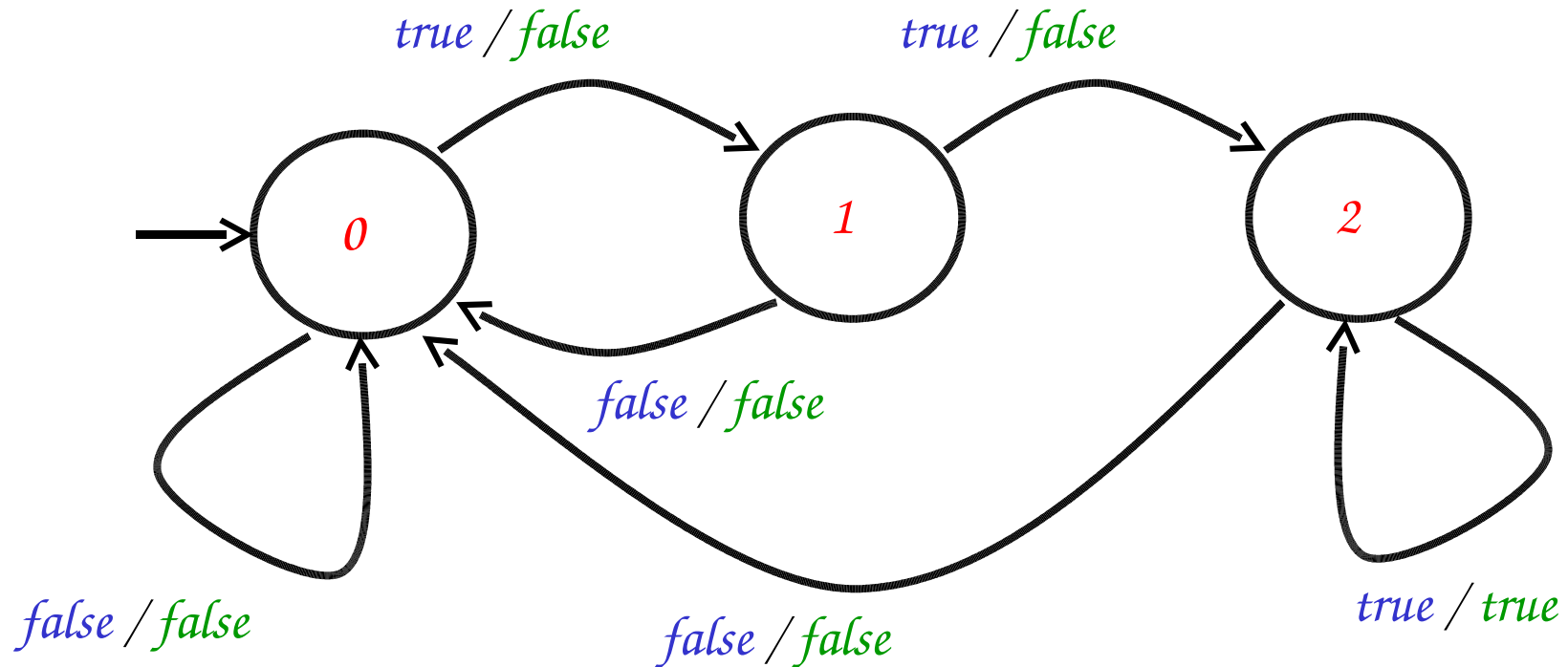
(or $i = 1$)

2, if both i -th and $(i-1)$ -st inputs are true

Three states

Exercise - (Continued)

Transition Diagram



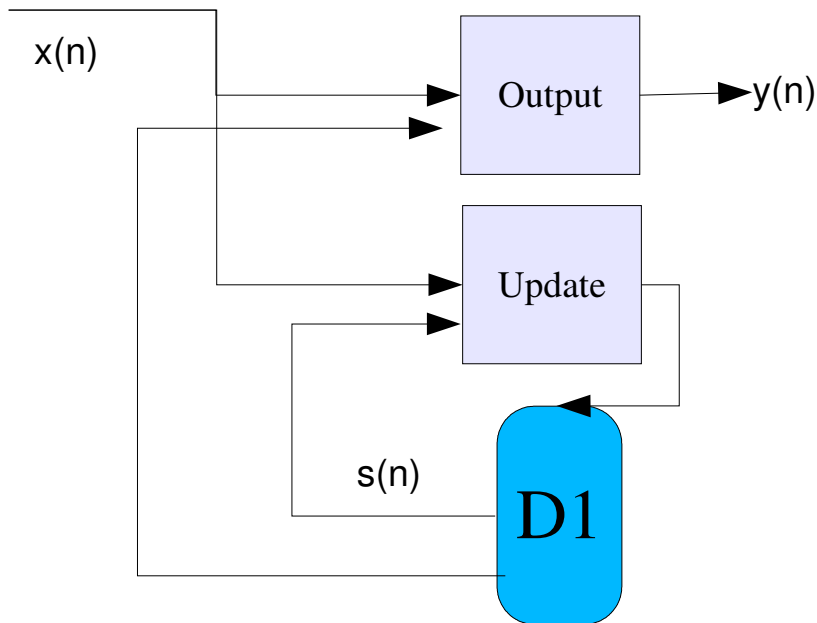
States = { 0, 1, 2 } *Inputs* =
Bools *Outputs* = *Bools*

Some terminology

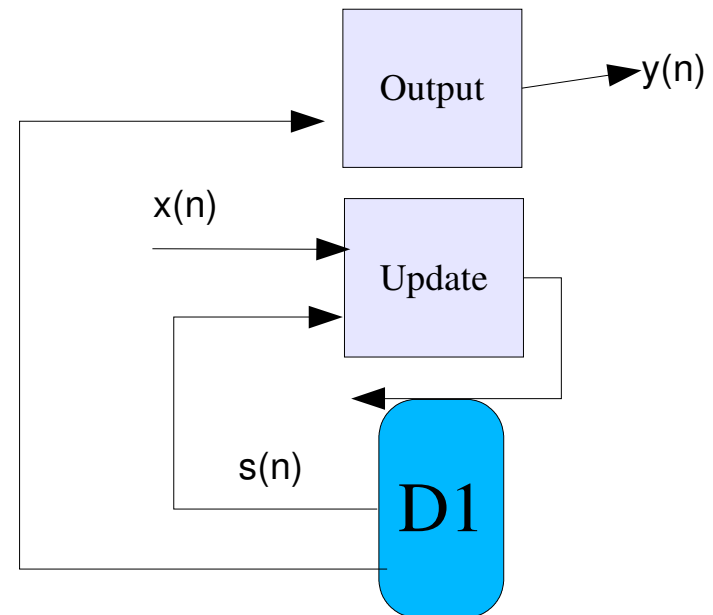
- **Receptiveness:** our SM always react to a symbol (there is an outgoing arc specified, or an else arc either specified or implied)
- **Determinism:** a SM is said deterministic if every input symbols is contained in one and only one outgoing arc
- State transitions triggered by a sequence of inputs are called **State response**
- A finite machine without output is called *finite automata*

Some definitions - (continued)

- The state machines addressed here are called **Mealy machines**. Mealy machines generate outputs during state transitions.
- **Moore machines** generate output while the system is in a particular state (output depends on state only).



Mealy



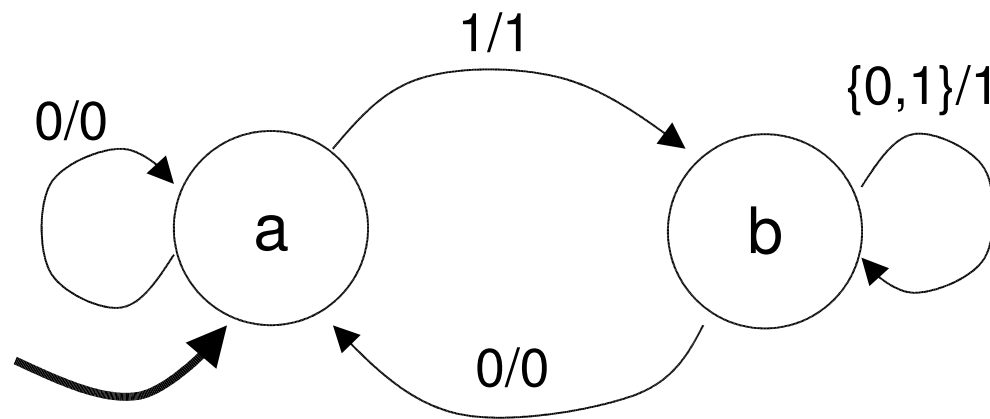
Moore

Some facts

- **A SM is state-determined:** Each transition and output depends only on the current state and current input.
 - Previous input elements only affect the transitions and output insofar as they determine the current state.
- A representation of a SM may not be unique
- Representations of a SM
 - **Update tables** (useful in computer implementations)
 - **State transition diagrams** (human friendly)
 - **Sets and Functions** (useful for mathematical purposes)

Nondeterministic State Machines

Nondeterministic state machines are like the deterministic state machines, except there may be **more than one possible transition** for a given current state and input.



When in state $s(n)=b$,
if $x(n) = 0$,
the next state $s(n+1)$
can be either a or b.

The output $y(n)$
can be either 0 or 1.

In a **nondeterministic** state machine, there is more than one possible state response and output sequence.

Nondeterministic State Machines: Parts

Nondeterministic state machines are described with the 5-tuple:

*(States, Inputs, Outputs, **possibleUpdates**, initialState)*

The update function is now called **possibleUpdates**.

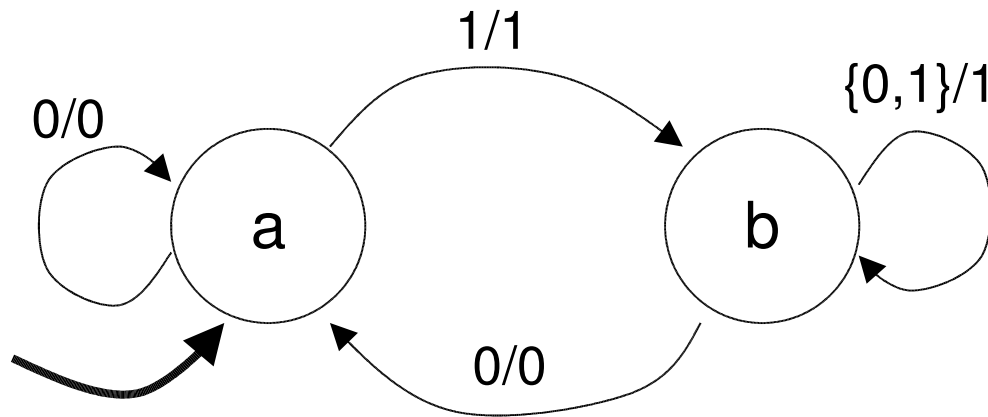
For a given input $x(n)$ and current state $s(n)$, *possibleUpdates* provides the **set** of possible next states $s(n+1)$ and outputs $y(n)$.

possibleUpdates: States $\times$ Inputs $\rightarrow P(\text{States} \times \text{Outputs})$

Here, the notation $P()$ denotes the power set. For some set S , $P(S)$ is the set of all subsets of S .

Nondeterministic State Machines: Example

Provide the 5-tuple definition for the following state machine:



States = {a, b}

Inputs = {0, 1, *absent*}

Outputs = {0, 1, *absent*}

initialState = a

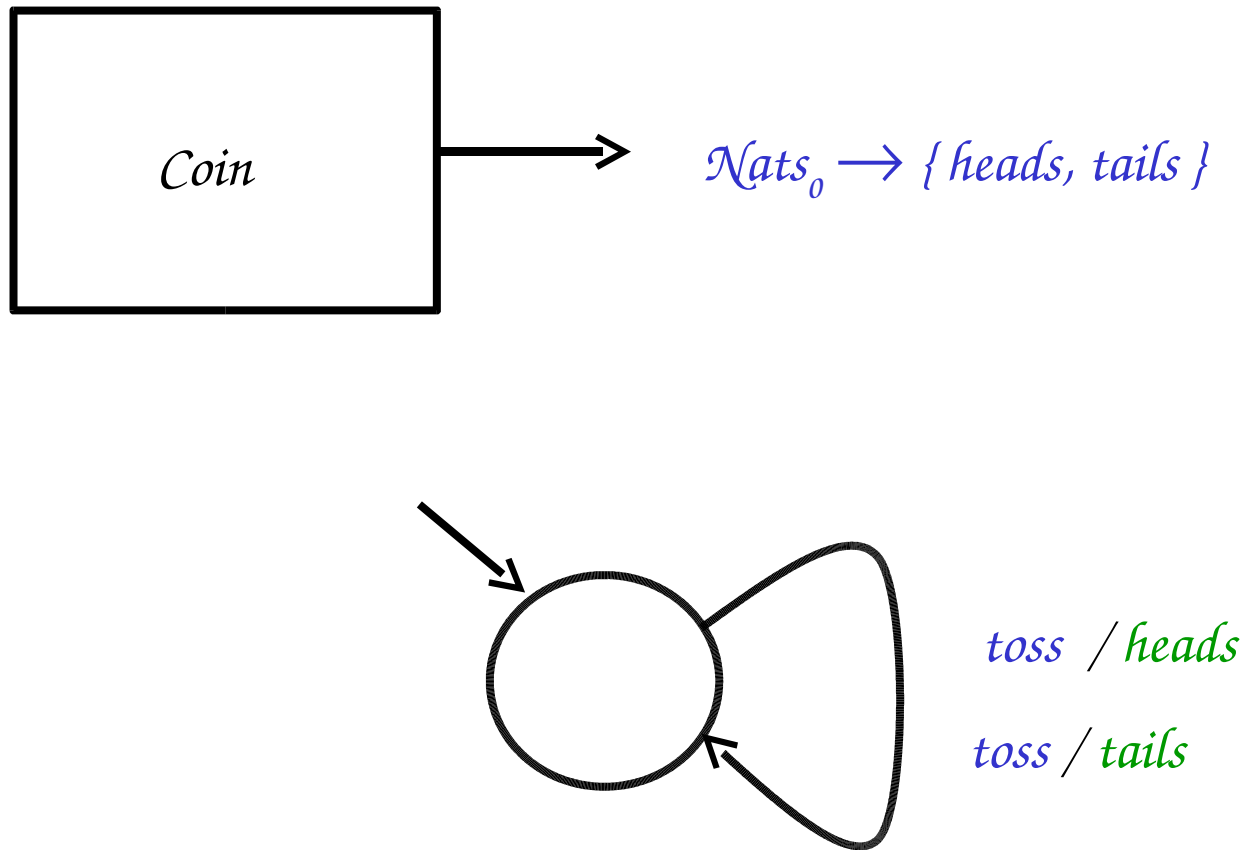
$(s(n+1), y(n)) = possibleUpdates(s(n), x(n))$		
	$x(n) = 0$	$x(n) = 1$
$s(n) = a$	(a, 0)	(b, 1)
$s(n) = b$	{(b, 1), (a, 0)}	(b, 1)

Why nondeterminism?

- modeling randomness
- modeling abstraction
- ...

Modeling randomness

Coin Tossing



Modeling randomness

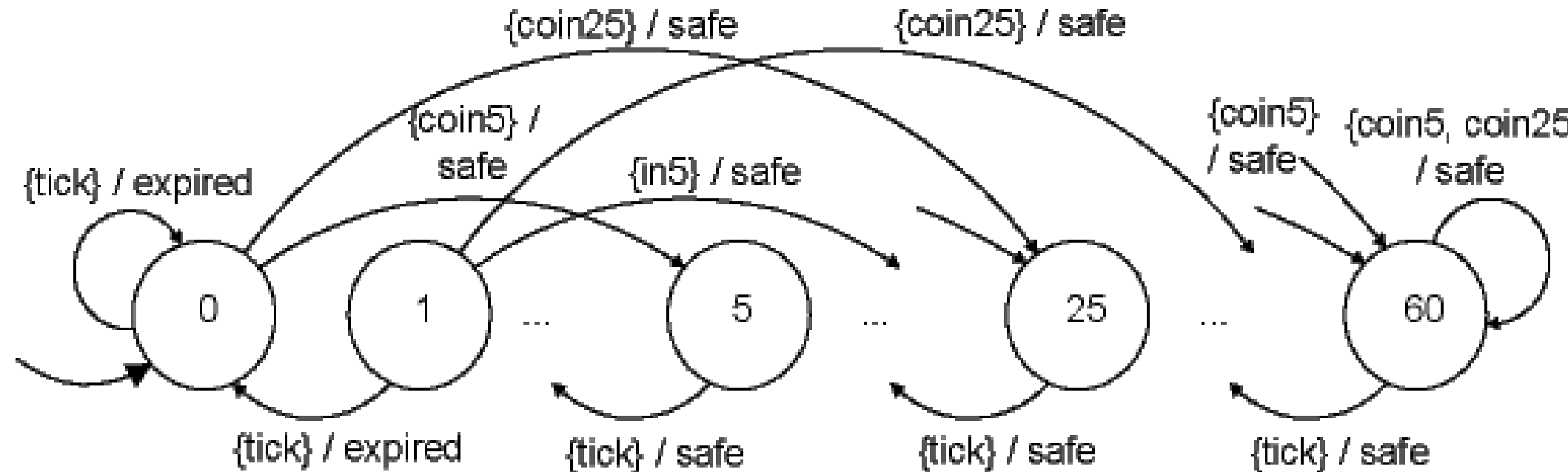
One possible behavior

<i>Time</i>	0	1	2	3	4
<i>Input</i>					
<i>Output</i>	<i>heads</i>	<i>heads</i>	<i>tails</i>	<i>heads</i>	<i>tails</i>

Another possible behavior

<i>Time</i>	0	1	2	3	4
<i>Input</i>					
<i>Output</i>	<i>tails</i>	<i>heads</i>	<i>tails</i>	<i>heads</i>	<i>heads</i>

Nondeterministic State Machines: Abstraction



This example is a deterministic model of a parking meter.

The inputs are **coin5** (5 minutes) **coin25** (25 minutes) and **tick**.

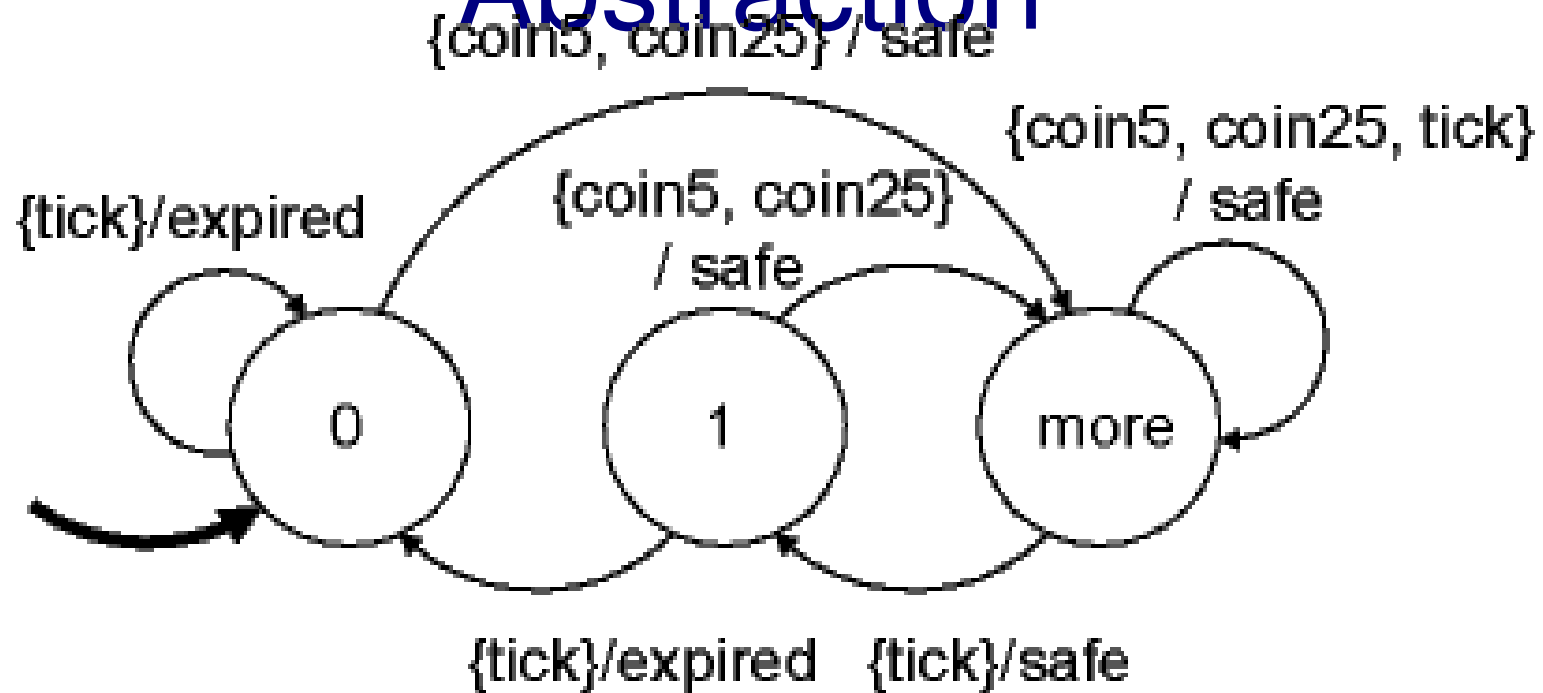
The outputs are **safe** and **expired**.

The states represent the number of minutes left until expiration.

This model of meter operation is accurate, but complicated!

Nondeterministic State Machines:

Abstraction



Here is a nondeterministic model of the parking meter.

It contains less detail – it is an **abstraction** of the previous model.

But, some outputs are possible in this model that are impossible in the previous model:

$x = \text{coin5, tick, tick}$ can result in $y = \text{safe, safe, expired}$

The meter expires 2 minutes after inserting a 5 minute coin!

State Machines: Behaviors

A **behavior** of a state machine is a pair (x, y) where x is an input sequence and y is a corresponding output sequence:

$Behaviors = \{(x, y) \in [Naturals_0 \rightarrow Inputs] \times [Naturals_0 \rightarrow Outputs] \mid y \text{ is a possible output sequence for the input } x\}$

Deterministic behaviours

Deterministic SM:

*for every input signal, there is **exactly one** output signal.*

Behaviours is the graph of a Function:

$$\mathcal{DetSys} : [\mathcal{Nats} \rightarrow Inputs] \rightarrow [\mathcal{Nats} \rightarrow Outputs]$$

Nondeterministic behaviours

Nondeterministic Reactive System:

*for every input signal, there is **one or more** output signals.*

Behaviours are a Binary relation:

$$\mathcal{NondetSys} \subseteq [\mathcal{Nats} \rightarrow Inputs] \times [\mathcal{Nats} \rightarrow Outputs]$$

*such that $\forall x \in [\mathcal{Nats} \rightarrow Inputs],$
 $x \in [\mathcal{Nats} \rightarrow Outputs], (x,y) \in \mathcal{NondetSys}$*

$\exists y$

Every pair $(x,y) \in \mathcal{NondetSys}$ is called a behavior.

Refinement

S1 is a more detailed description of S2;
S2 is an abstraction or property of S1.

System S1 **refines** system S2
iff

1. Time [S1] = Time [S2] ,
2. Inputs [S1] = Inputs [S2] ,
3. Outputs [S1] = Outputs [S2] ,
4. Behaviors [S1] $\subseteq$ Behaviors [S2] .

Equivalence

Systems $S1$ and $S2$ are **equivalent**
iff

1. $\text{Time } [S1] = \text{Time } [S2]$,
2. $\text{Inputs } [S1] = \text{Inputs } [S2]$,
3. $\text{Outputs } [S1] = \text{Outputs } [S2]$,
4. **$\text{Behaviors } [S1] = \text{Behaviors } [S2]$.**

Simulation

A binary relation $S \subseteq \text{States } [\mathcal{M1}] \times \text{States } [\mathcal{M2}]$ is a *simulation* of $\mathcal{M1}$ by $\mathcal{M2}$ iff

1. $\forall p \in \text{possibleInitialStates } [\mathcal{M1}] ,$

$\exists q \in \text{possibleInitialStates } [\mathcal{M2}] , (p, q) \in S$ and

2. $\forall p \in \text{States } [\mathcal{M1}] , \forall q \in \text{States } [\mathcal{M2}] ,$

if $(p, q) \in S ,$

then $\forall \chi \in \text{Inputs} , \forall y \in \text{Outputs} , \forall p' \in \text{States } [\mathcal{M1}] ,$

if $(p', y) \in \text{possibleUpdates } [\mathcal{M1}] (p, \chi)$

then $\exists q' \in \text{States } [\mathcal{M2}] ,$

$(q', y) \in \text{possibleUpdates } [\mathcal{M2}] (q, \chi)$ and

$(p', q') \in S .$

Conditions for refinement

condition on behaviors

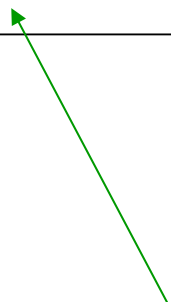


Theorem :

The nondeterministic state machines M1 **refines** the
nondeterministic state machine M2

if

there exists a **simulation** of M1 by M2 .



relation between states

Beware

Theorem :

The nondeterministic state machines $M1$ **refines** the
nondeterministic state machine $M2$

if
there exists a **simulation** of $M1$ by $M2$.

Not “if-and-only-if” !

Not symmetric !

*(We say that “ $M2$ **simulates** $M1$ ”.)*

Intuition

- The theorem simply states that M2 can match every move of m1 producing the same output
- Moreover, if M2 cannot produce an output sequence, neither can M1, as stated below:

Corollary:

Let M2 simulate M1. If $(x,y) \notin \text{Behaviours}_{M2}$ then

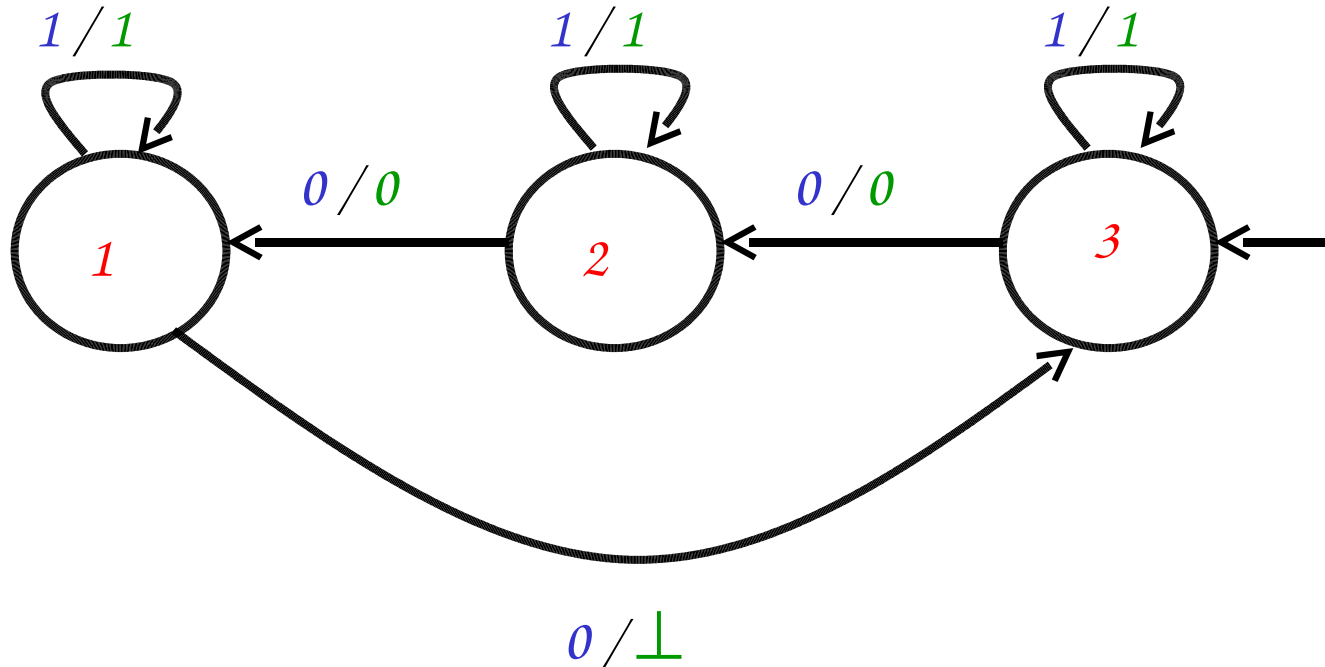
$(x,y) \notin \text{Behaviours}_{M1}$.

Example

Channel that drops every third zero



Example - (continued)

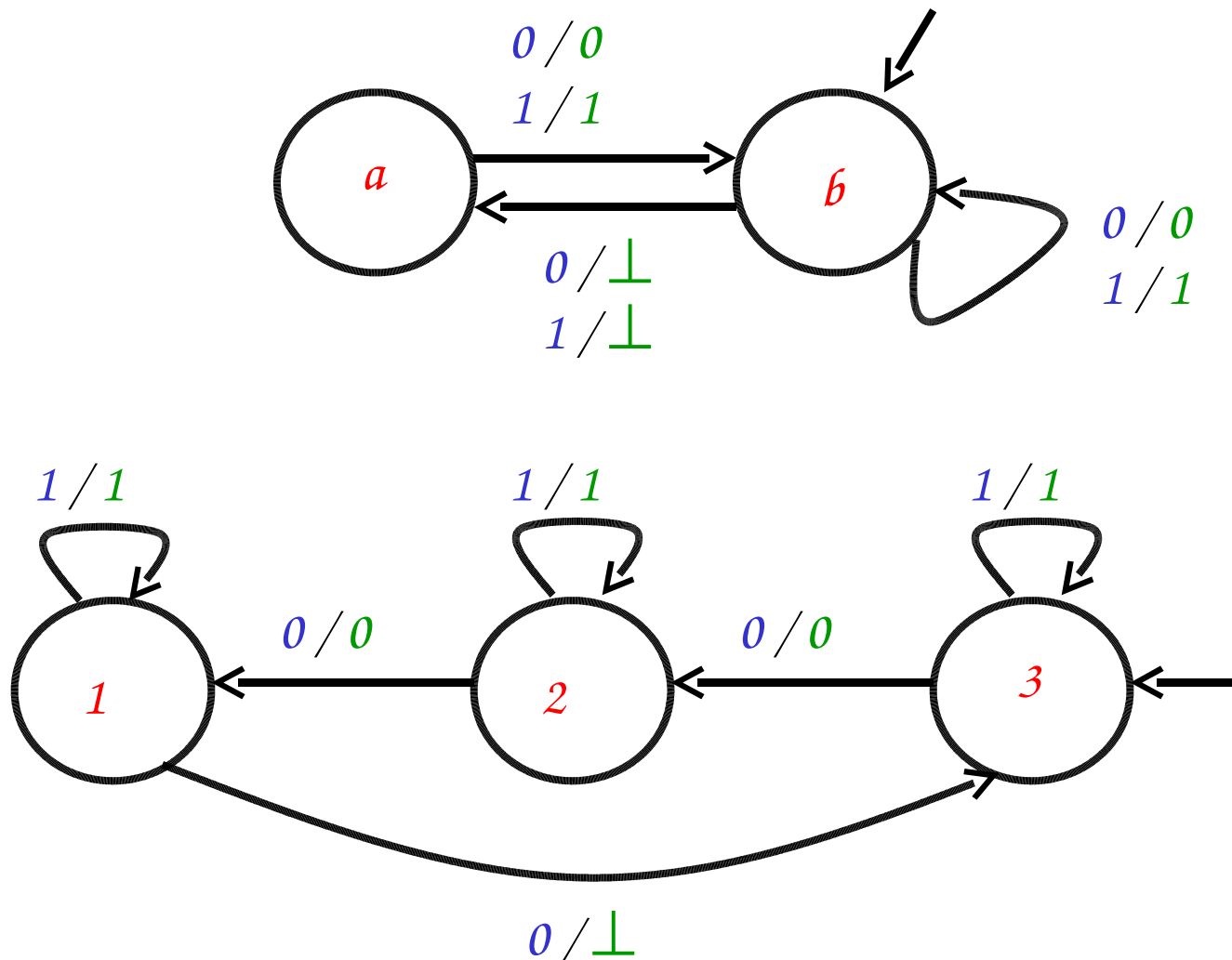


State between time $t-1$ and time t :

- 3 third zero from now will be dropped*
- 2 second zero from now will be dropped*
- 1 next zero will be dropped*

Example - (continued)

ThirdZero refines NotTwice

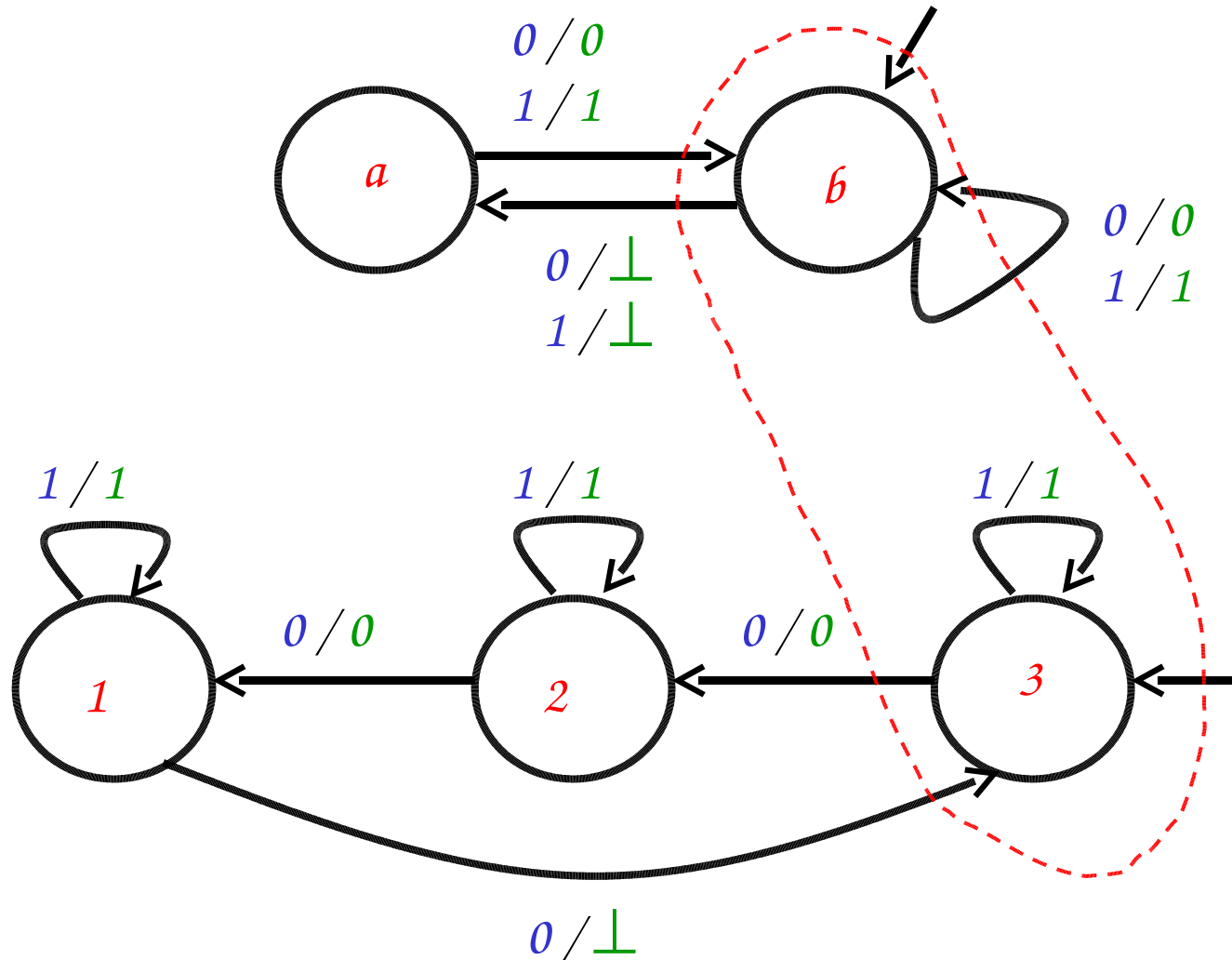


Example - (continued)

ThirdZero is simulated by *NotTwice*

Simulation

$S = \{(3, b),$

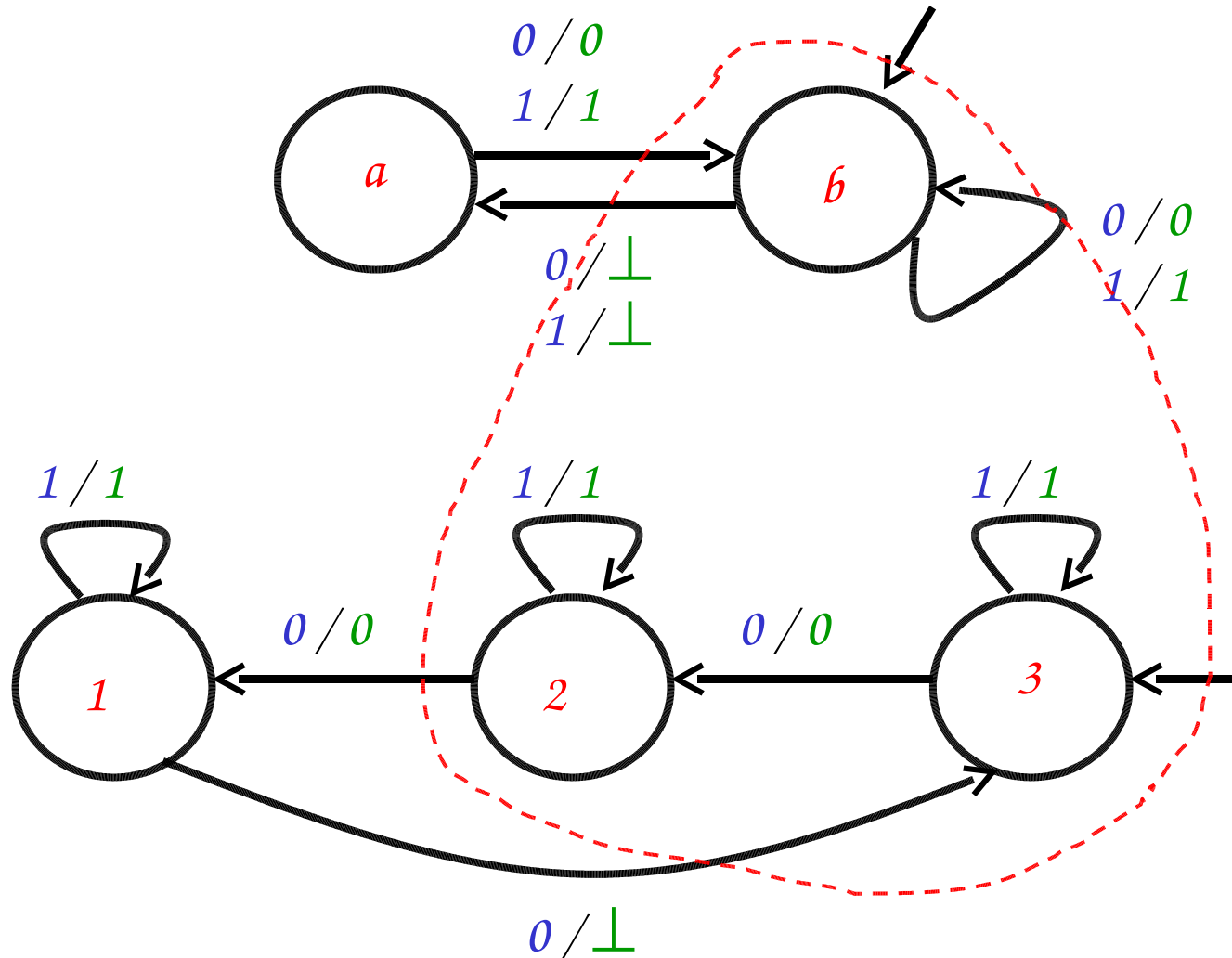


Example - (continued)

ThirdZero is simulated by *NotTwice*

Simulation

$\mathcal{S} = \{(3,b), (2,b),$

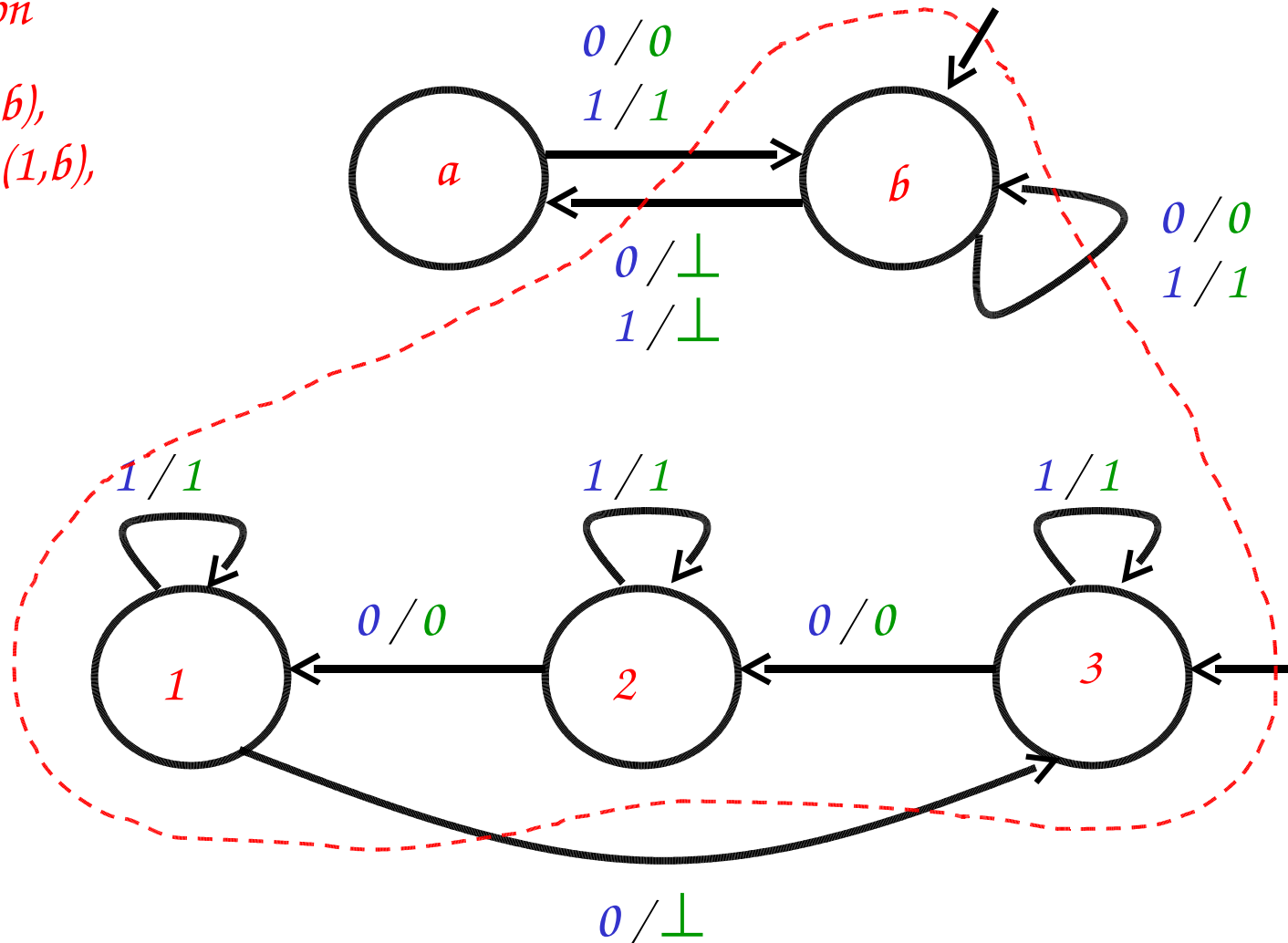


Example - (continued)

ThirdZero is simulated by *NotTwice*

Simulation

$\mathcal{S} = \{ (3,b), (2,b), (1,b) \}$

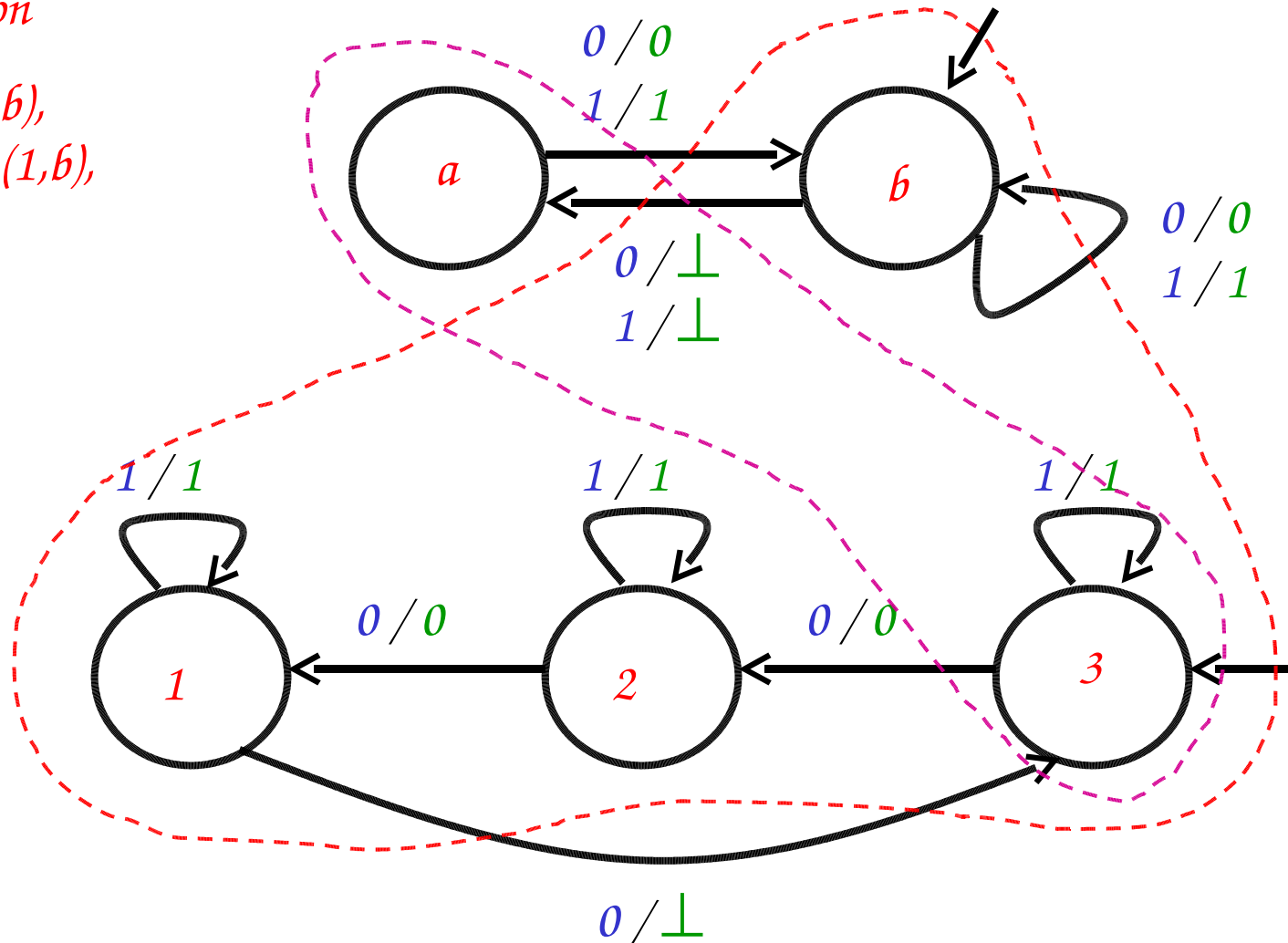


Example - (continued)

ThirdZero is simulated by *NotTwice*

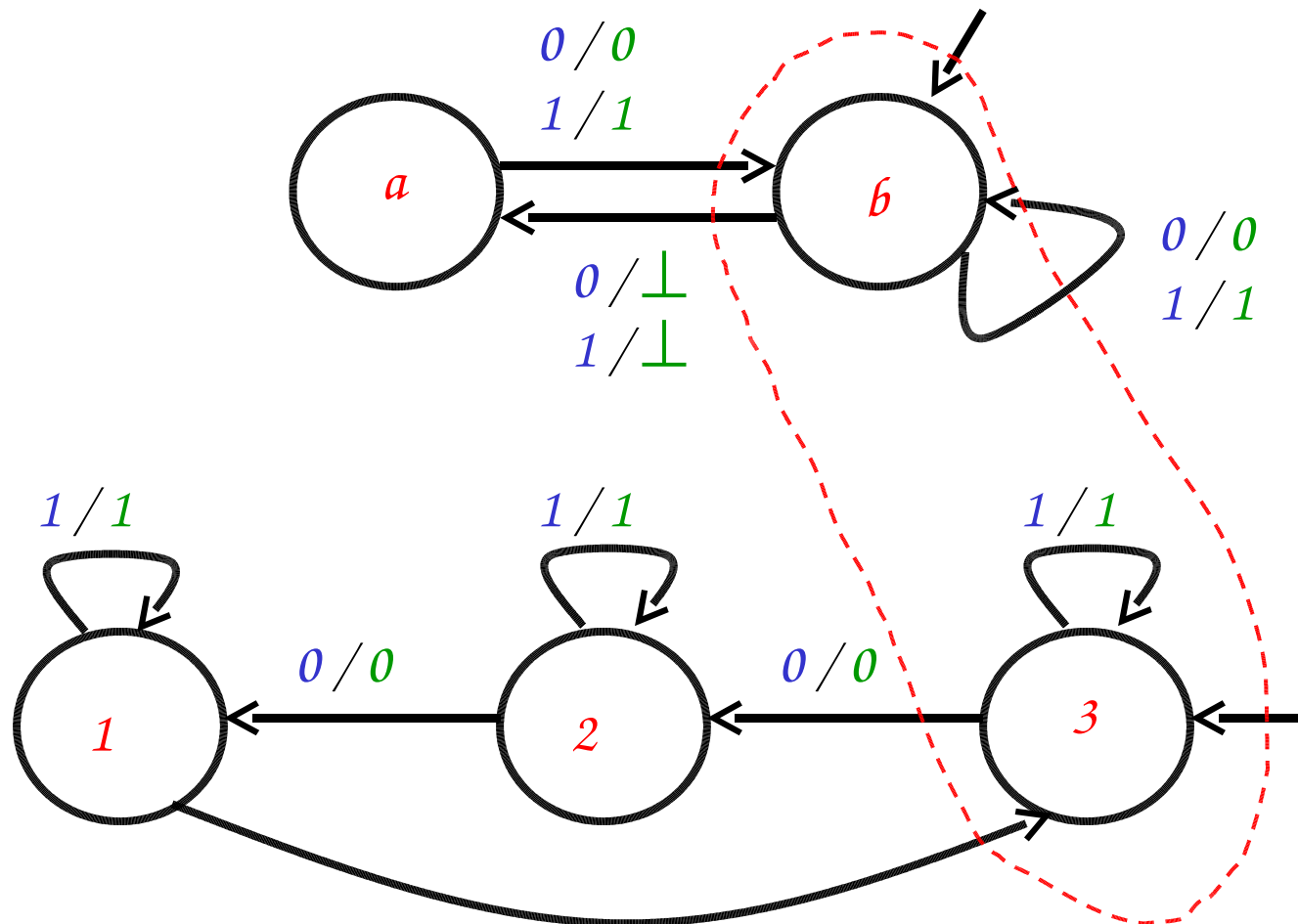
Simulation

$\mathcal{S} = \{ (3,b), (2,b), (1,b), (3,a) \}$



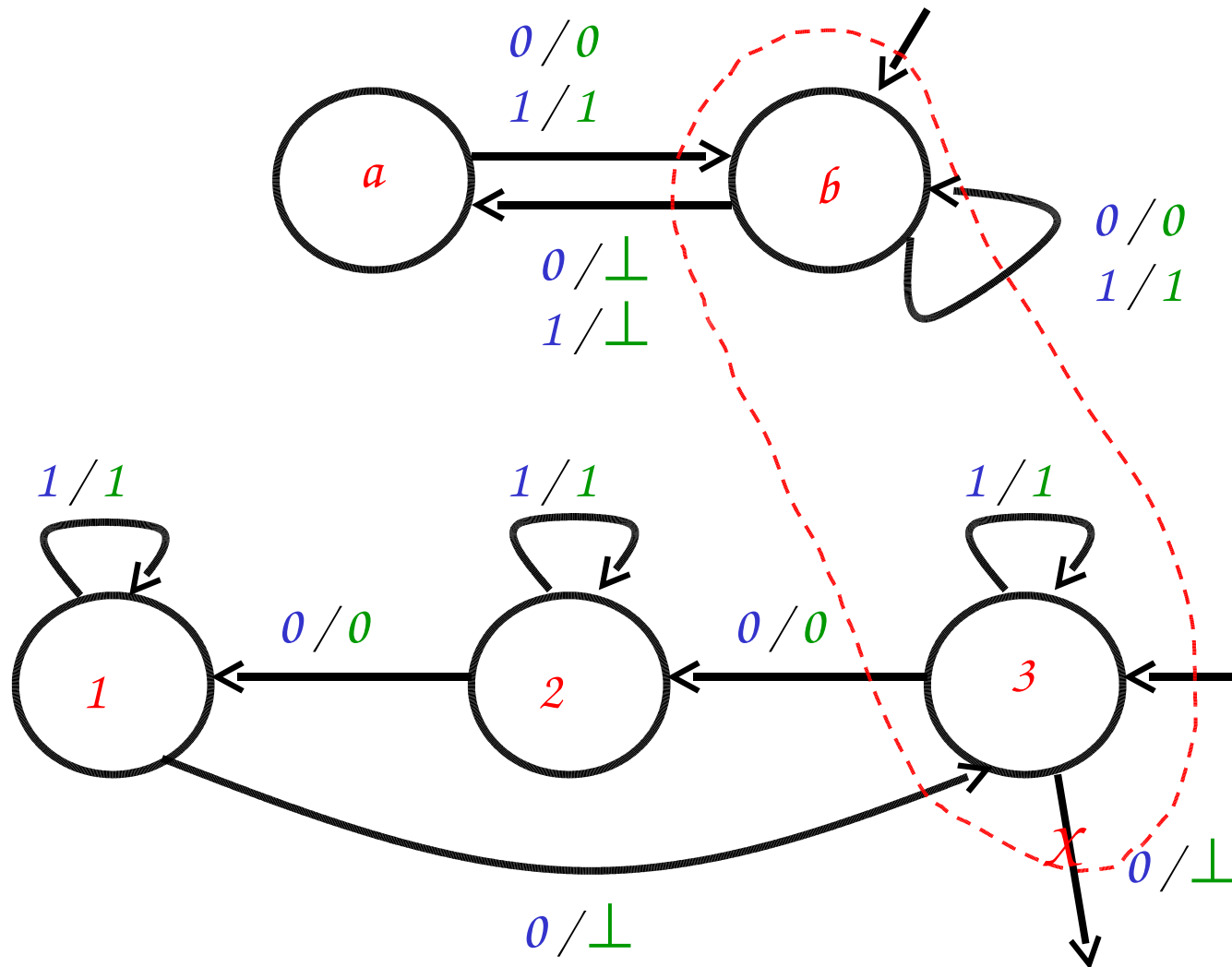
Example - (continued)

NotTwice is **not** simulated by *ThirdZero*



Example - (continued)

NotTwice is **not** simulated by *ThirdZero*



Remark

- If M2 satisfies a property, so does M1
- For instance, insofar as the property not-twice, is of interest, we can prove it on *NotTwice* and it also applies to *ThirdZero*

Remark 1

If $M2$ is a **deterministic** state machine, then

$M1$ is simulated by $M2$

iff

$M1$ is equivalent to $M2$.

Remark 2

If $M2$ is a **nondeterministic** state machine, then

$M1$ is simulated by $M2$

implies

$M1$ refines $M2$,

but $M2$ may not refine $M1$.

(Recall $M1 = ThirdZero$ and $M2 = NotTwice$.)