Laurea Specialistica in Ingegneria dell'Automazione

Sistemi in Tempo Reale

Luigi Palopoli
email: palopoli@sssup.it   Tel. 050 883444

Finite-State Machines

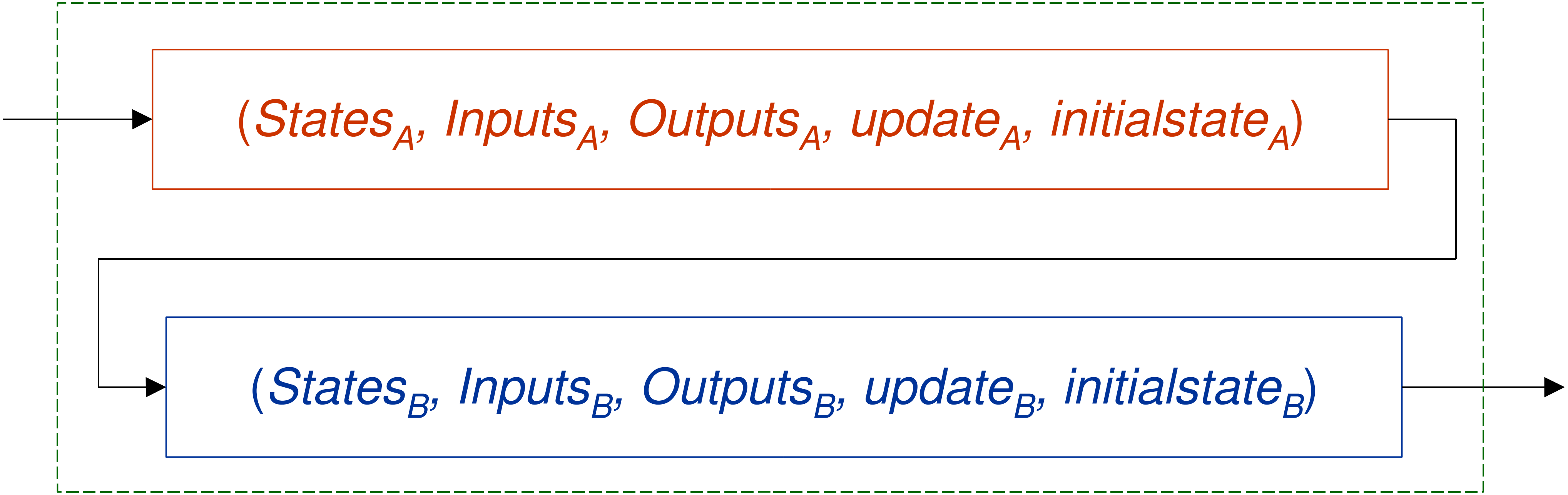
Outline

- Something more about simulation
- Connections of SM
- Feedback
- State-charts
- Implementation

Synchrony

- Each state machine in the composition reacts to external inputs *simultaneously* and *instantaneously*
- Our system react to external stimuli: for this reason they are called *reactive*
- Because our systems react synchronously to external inputs they are called synchronous/reactive (SR)

Cascade Composition



Here, the output of machine A is the input of machine B.

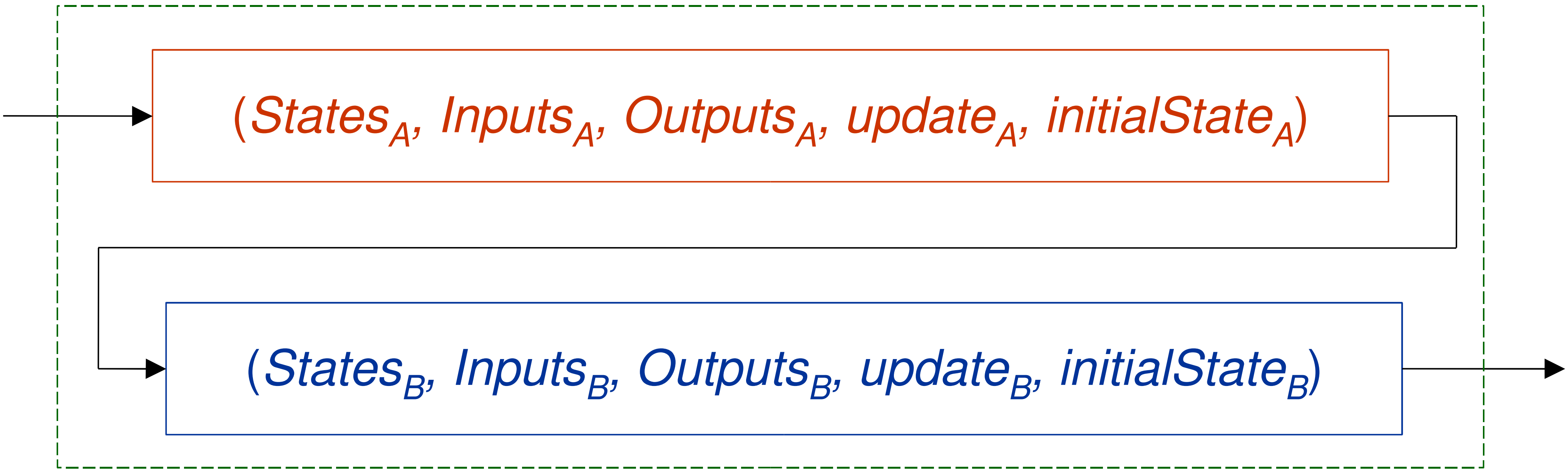
The two machines are running simultaneously (both on step n)

Each machine has its own input, current state, and output.

The effect of the input $x_A(n)$ propagates instantaneously through the cascade at each step: **synchrony**

We may view the cascade of two machines as a single machine.

Cascade Composition: Definition



Define the 5-tuple for the composite machine:

$States = States_A \times States_B$

$initialState = (initialState_A, initialState_B)$

$update((s_A(n), s_B(n)), x(n)) = ((s_A(n+1), s_B(n+1)), y(n))$

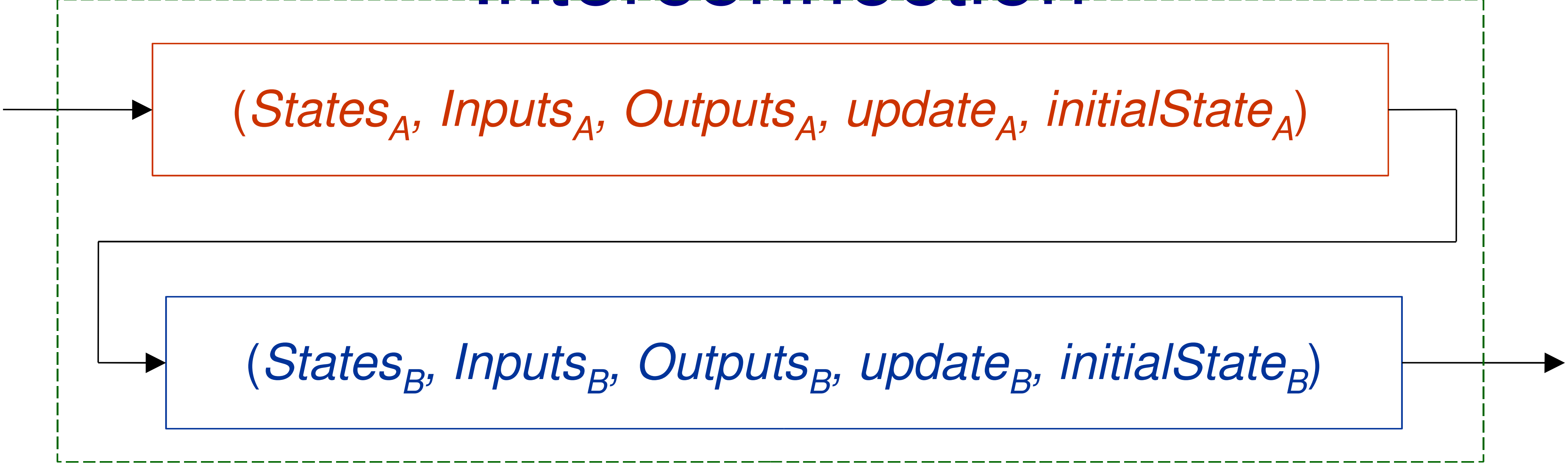
where $(s_A(n+1), y_A(n)) = update_A(s_A(n), x(n))$

and $(s_B(n+1), y(n)) = update_B(s_B(n), y_A(n))$

$Inputs = Inputs_A$

$Outputs = Outputs_B$

Cascade Composition: Interconnection



Note that the “internal” output $y_A(n)$ is used as the “internal” input $x_B(n)$ to machine B.

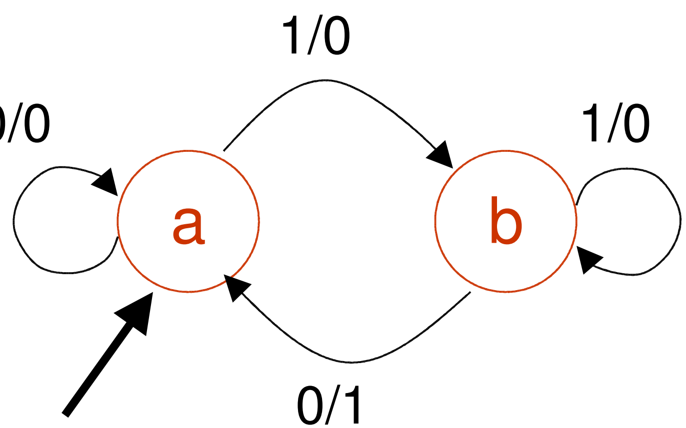
Thus, for the cascade connection to be valid, we must have

$$Outputs_A \subset Inputs_B$$

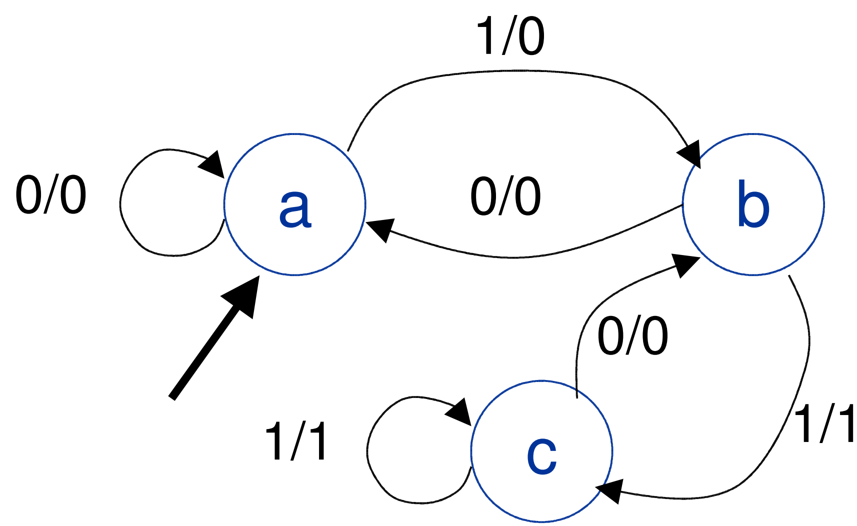
Cascade Composition: Example

Consider the cascade of machine A and machine B below, where the output of machine A is the input of machine B.

Find the composite state response and output for input x = 1 0 0 .



Machine A



Machine B

s = (a, a), (b, a), (a, b), (a, a)   y = 0 0 0

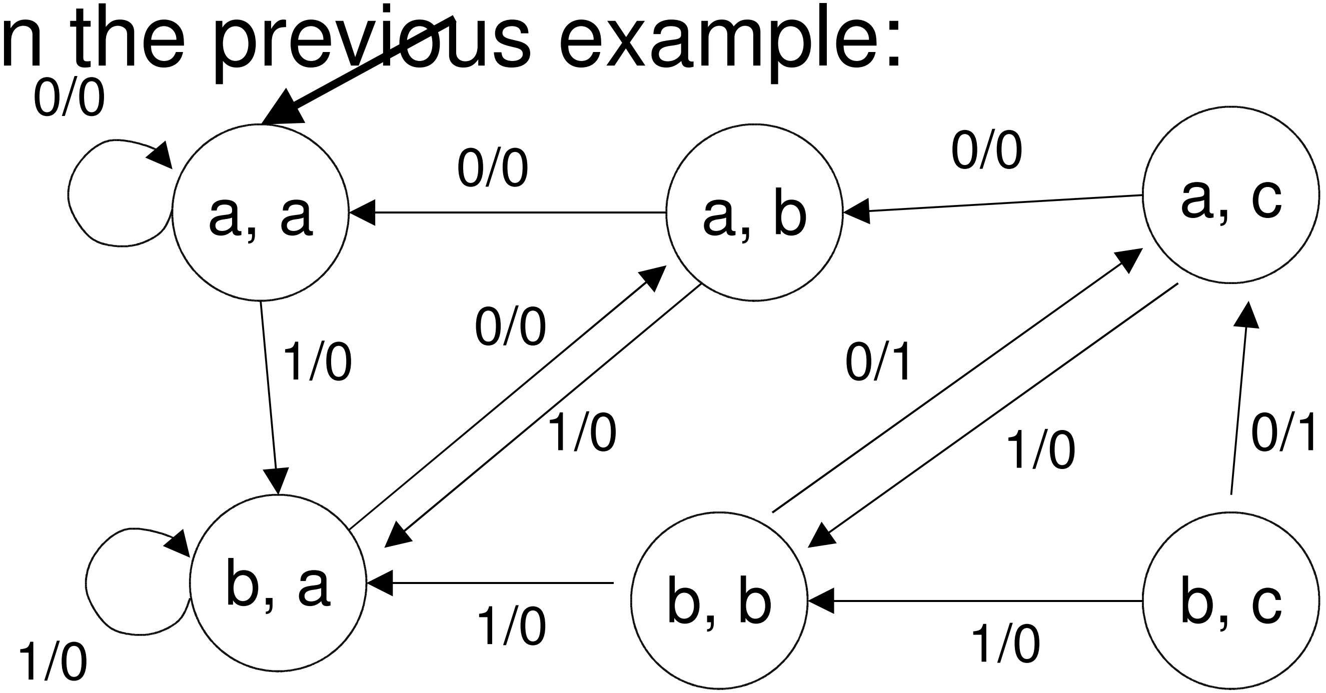
Cascade Composition: Diagram

Two cascaded machines can be drawn using one state diagram:

- Draw a circle for each state in $States_A$ x $States_B$.
- For each state, consider each possible input to machine A.
  a) Find the corresponding next state in machine A.
  b) Find the output of machine A, which forms the input of machine B.
  c) Find the corresponding next state in machine B.
  d) Find the output of machine B.
  e) Draw the transition arrow to $(s_A(n+1), s_B(n+1))$.
  f) Label the transition arrow with the input to machine A and the output from machine B.

Cascade Composition: Diagram

Try drawing a single state diagram for machines A and B in the previous example:

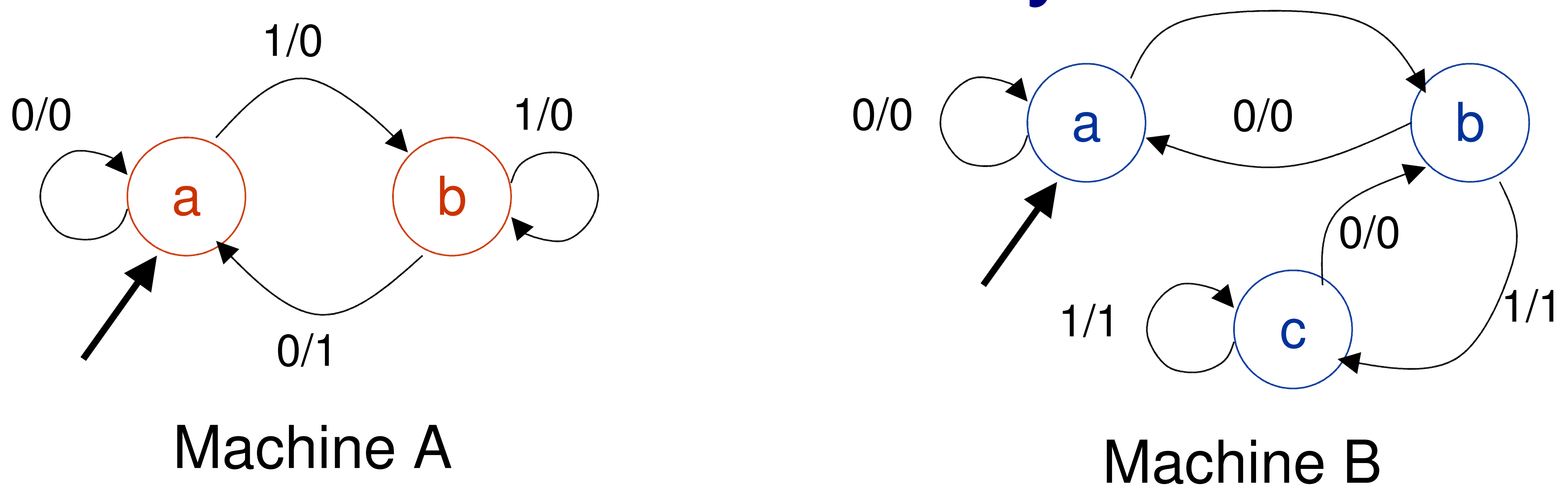


Can we ever get to state (b, c)?

Can we ever get to states (a, c) or (b, b)?

Can we ever get a nonzero output?

Reachability

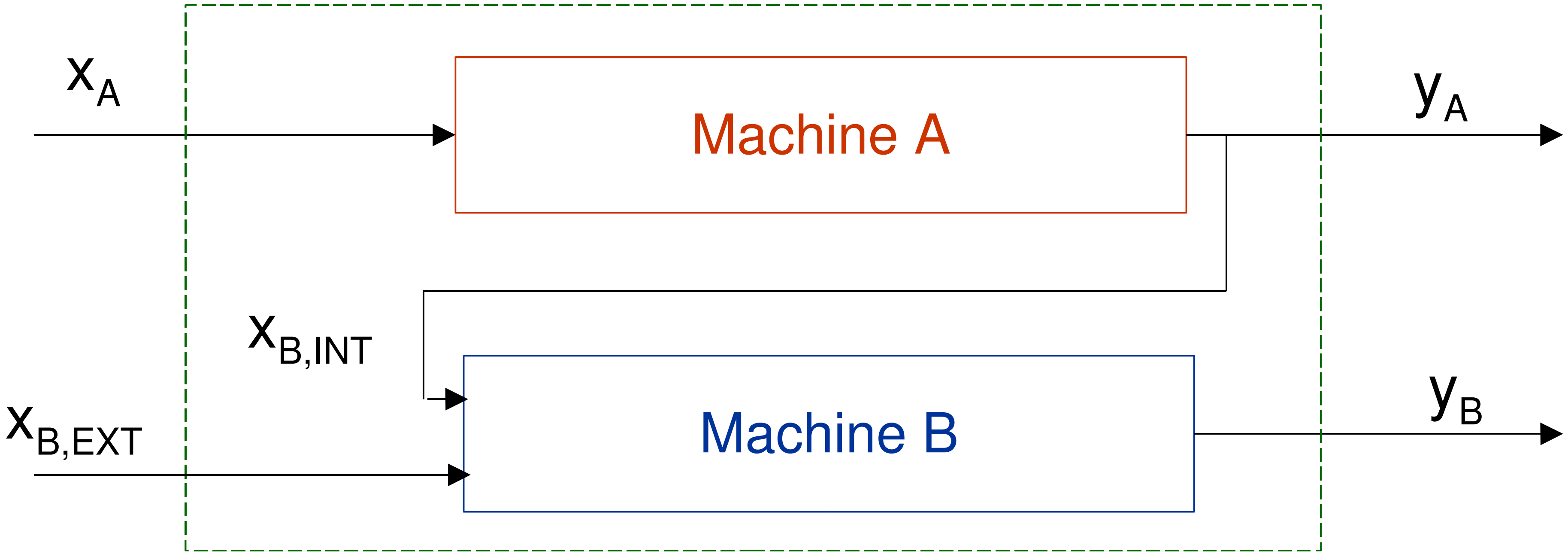


On its own, given its entire set of legal inputs, Machine B can reach state c and give an output of 1.

But, in cascade, the inputs of Machine B are limited to the possible outputs of Machine A.

Machine A cannot generate a sequence of outputs that would drive machine B into state c. This behavior is not in Behaviors_A.

More Complicated Connections

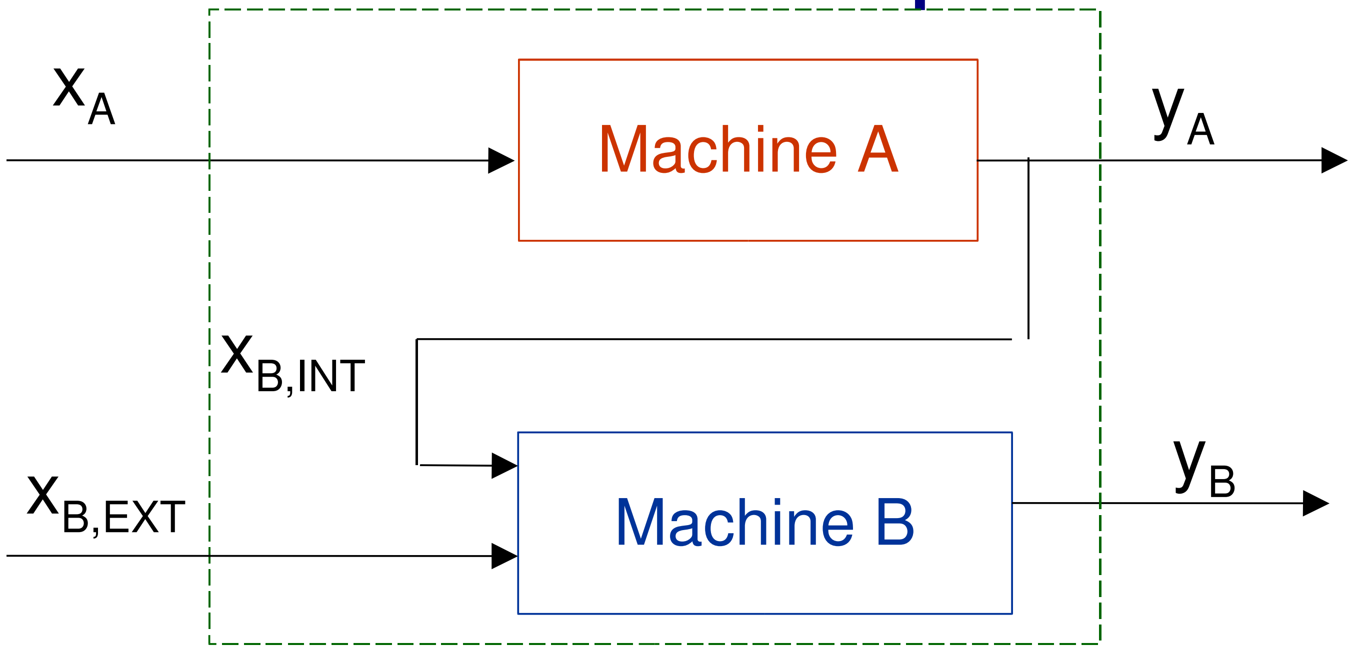


Here, we wish to have access to the individual output y_A, even when treating the cascade as one big machine.

Sending a signal to more than one destination is called **forking**.

Note also that there is an additional external input into machine B.

More Complicated Connections



Each arrow coming into or out of the composite machine is called a **port**.

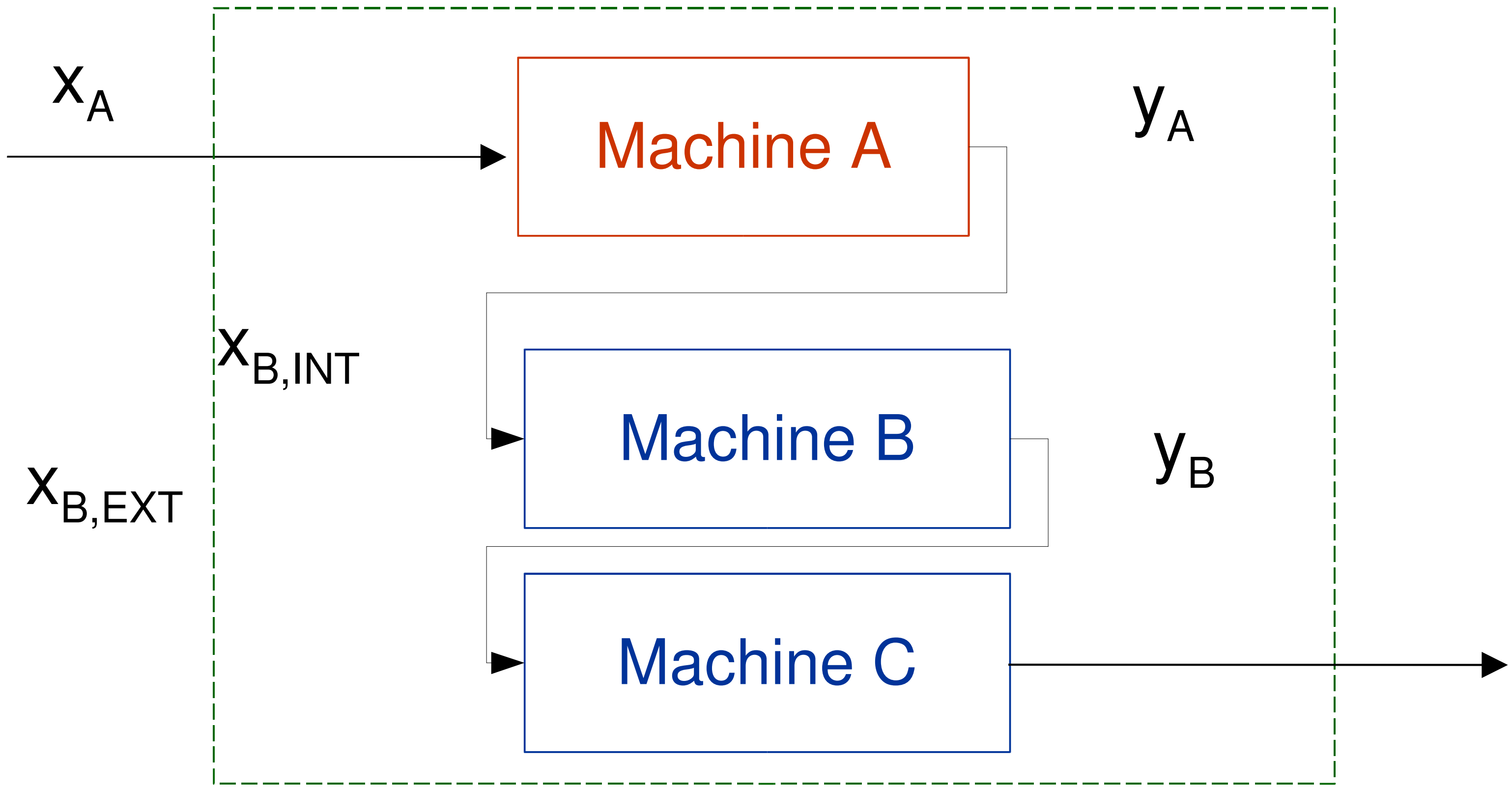
Here, there are two input ports and two output ports.

In the composite machine, we can express the input and output in the expected way using a set product:

$Inputs = Inputs_A \times Inputs_{B,EXT}$ $\quad$ $Outputs = Outputs_A \times Outputs_B$

The set of inputs or outputs for a particular port (Outputs_B, for example) is called a **port alphabet**.

Hierarchical composition

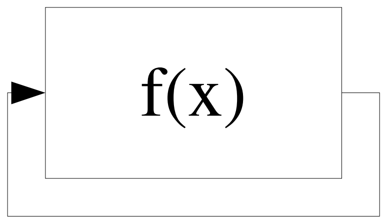


We can compose A and B and then C, or we can make it in any other order obtaining bisimilar machines.

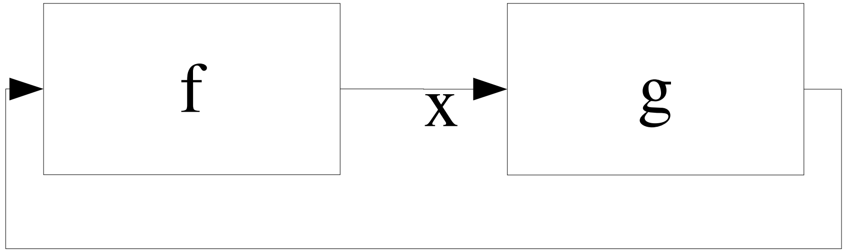
Outline

- Something more about simulation
- Connections of SM
- Feedback
- State-charts
- Implementation

Fixed point



Fixed point: x = f(x)



Fixed point: x = f(y), y = g(x)

Examples

$$f(x) = x^2$$

$$x = 0, x = 1$$

Two f'ps: ill-posed!

$$f(x) = x^2 + 1$$

$$x = \{\}$$

no f'ps: ill-posed!

$$f(x) = -x + 1$$

$$x = \frac{1}{2}$$

well-posed

Particular case: sm with no external inputs

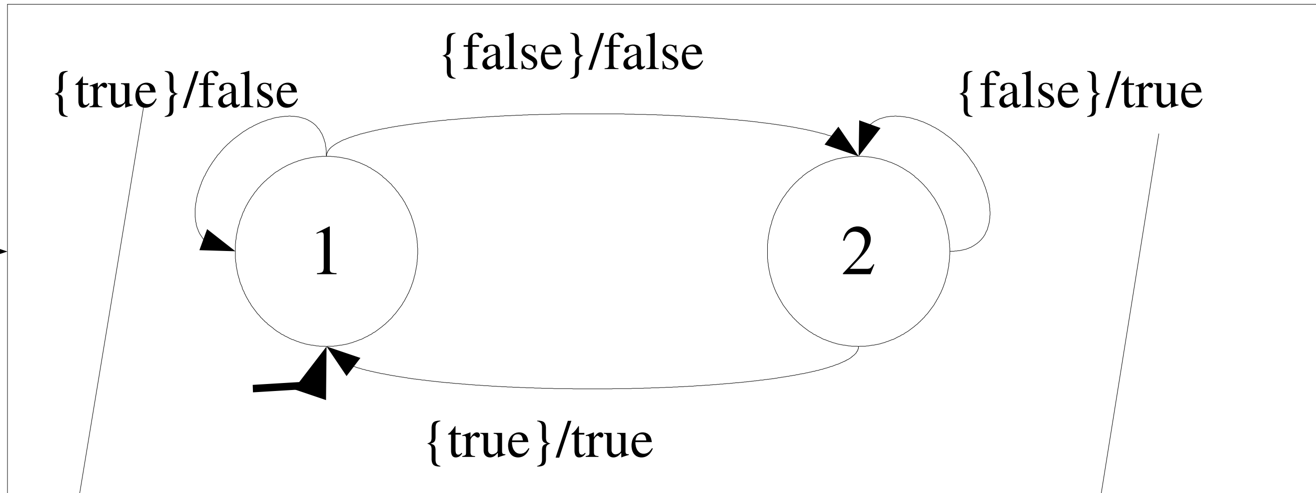
- Introduce two fictitious input symbols: *react* and *absent*

$$Outputs_A \subset Input_A$$
$$(s(n+1), y(n)) = update_A(s(n), y(n)) \rightarrow$$
$$y(n) = ouput_A(s(n), y(n))$$

Solving this FP problem is the key point

- Clearly the machine stuttering always works!
- We are interested in non-stuttering FP

Example 1

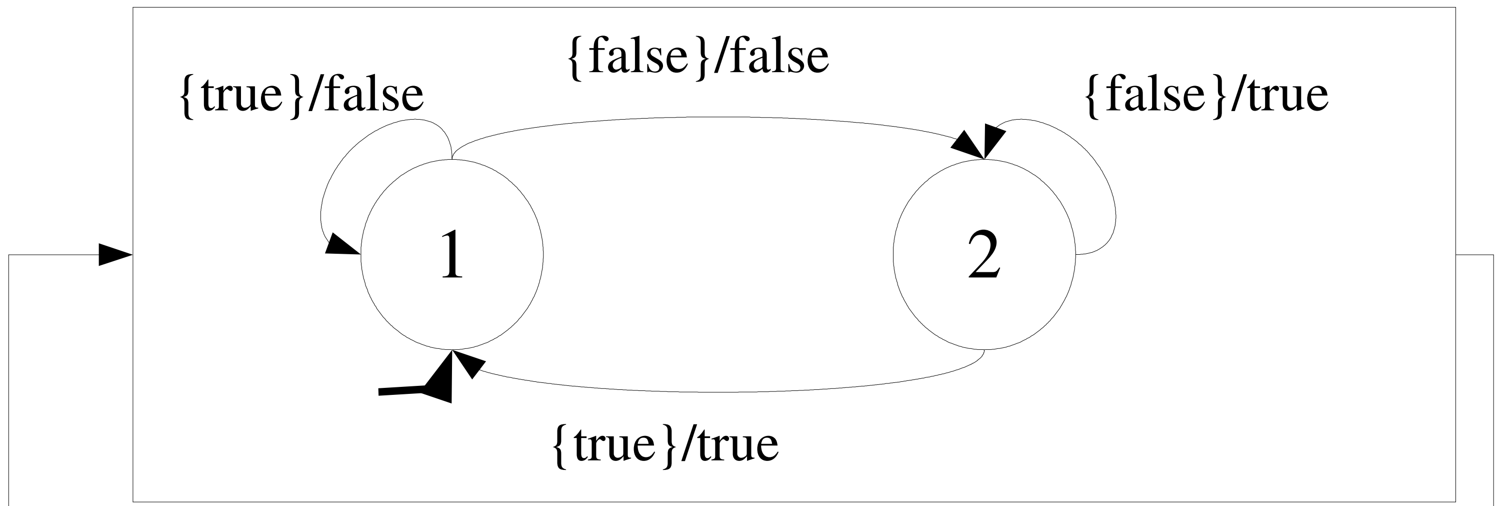


$$Outputs_A = Input_A = \{true, false, absent\}$$
$$y(n) = ouput_A(s(n), y(n))$$

All arcs outgoing from 1 produce false

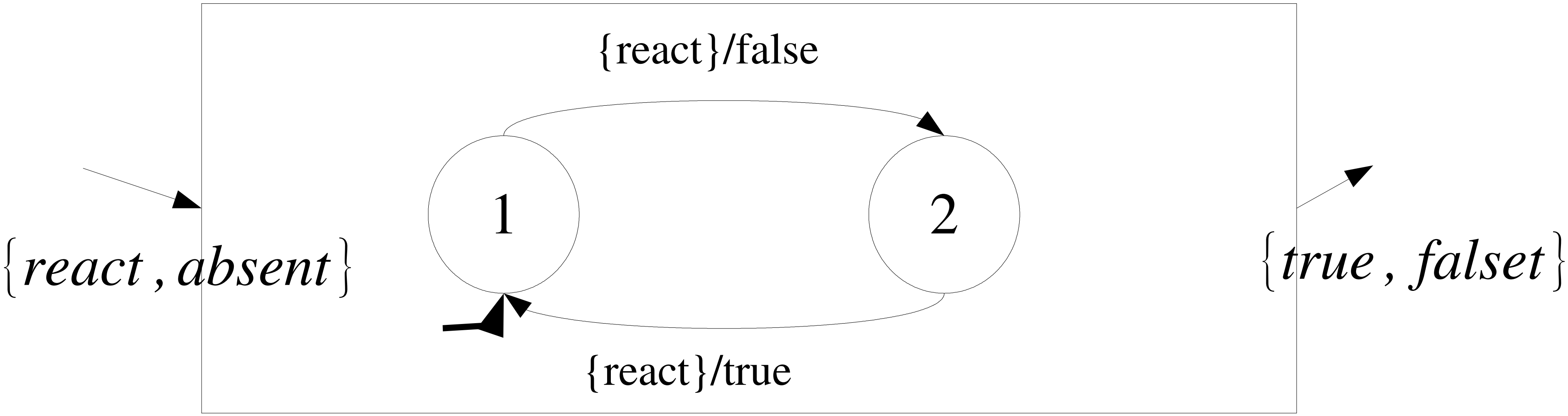
All arcs outgoing from 2 produce true

Example 1



$$false = ouput_A(1, false)$$
$$true = ouput_A(2, true)$$

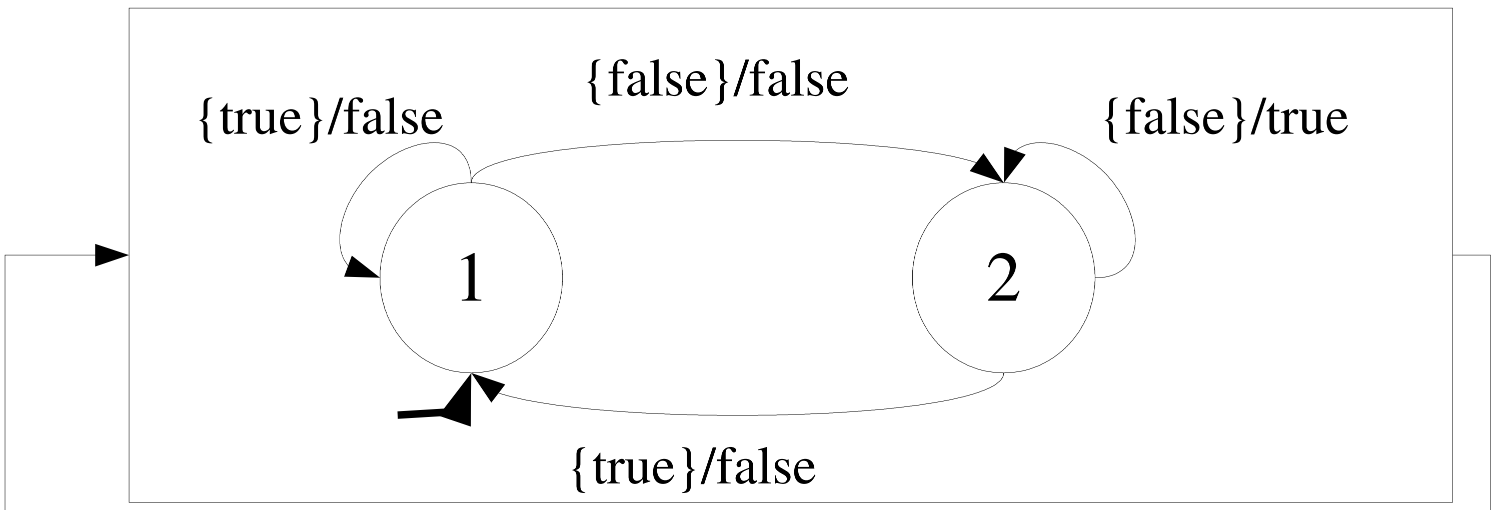
Example - Equivalent machine



$$false = ouput_A(1, false)$$
$$true = ouput_A(2, true)$$

$$\{react\ react\ react\ \dots\} \rightarrow \{false, true, false, true, \dots\}$$

Example 2

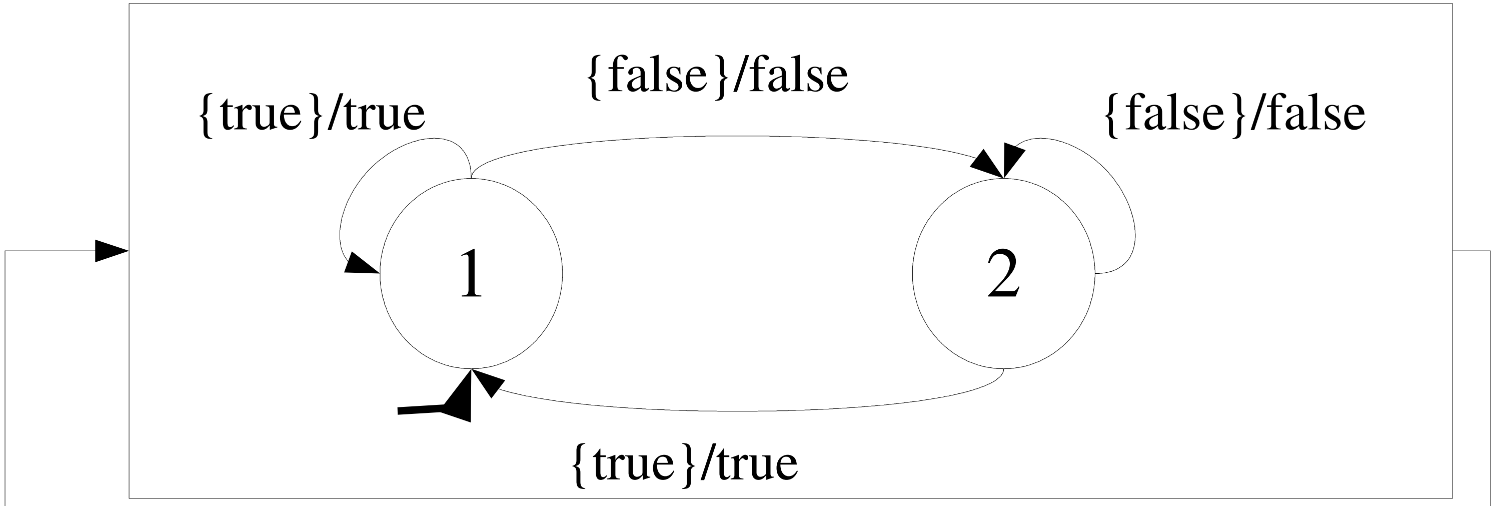


$$false = ouput_A(1, false)$$

No Fp for

$$y(n) = ouput_A(2, y(n))$$

Example 3



$$false = ouput_A(1, false)$$

$$true = ouput_A(1, true)$$

Two Fps!!!

State-determined outputs

- Feedback composition ia well formed if the output is state-determined (all arcs outgoing from a state have the same output):

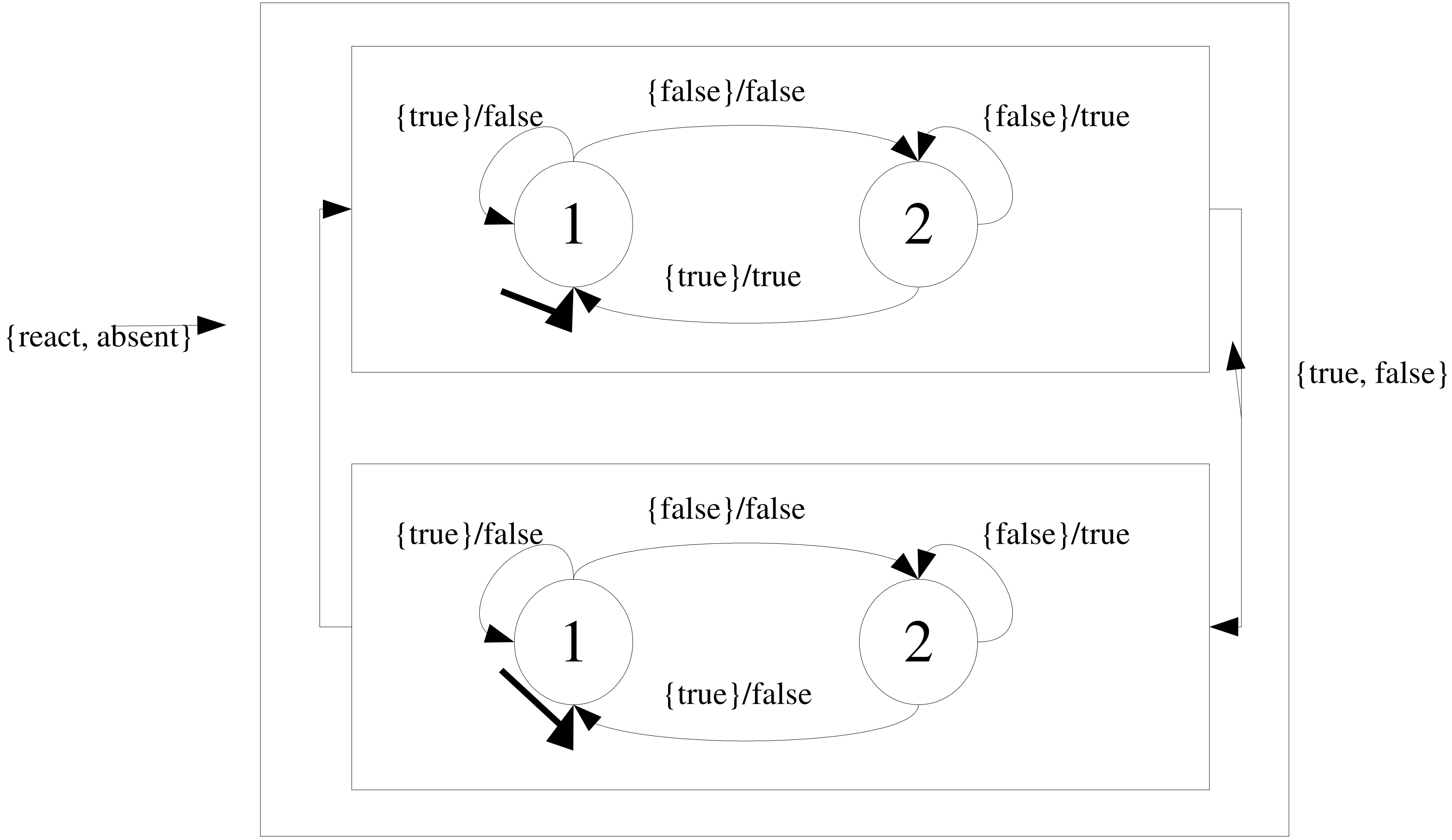
$$\forall\, reachable\; s(n)\, \forall\, x(n) \neq absent\,,\; update_A(s(n), x(n)) = b$$

- In this case the composition is defined as follows:

$$states = states_A$$
$$Inputs = \{react\,, absent\}$$
$$Outputs = Outputs_A$$
$$initialState = intialState_A$$
$$update(s(n), x(n)) = \begin{Bmatrix} update_A(s(n), b)\; where\; b = output_A(s(n), x(n)) & if\; x(n) = react \\ (s(n), x(n)) & if\; x(n) = absent \end{Bmatrix}$$

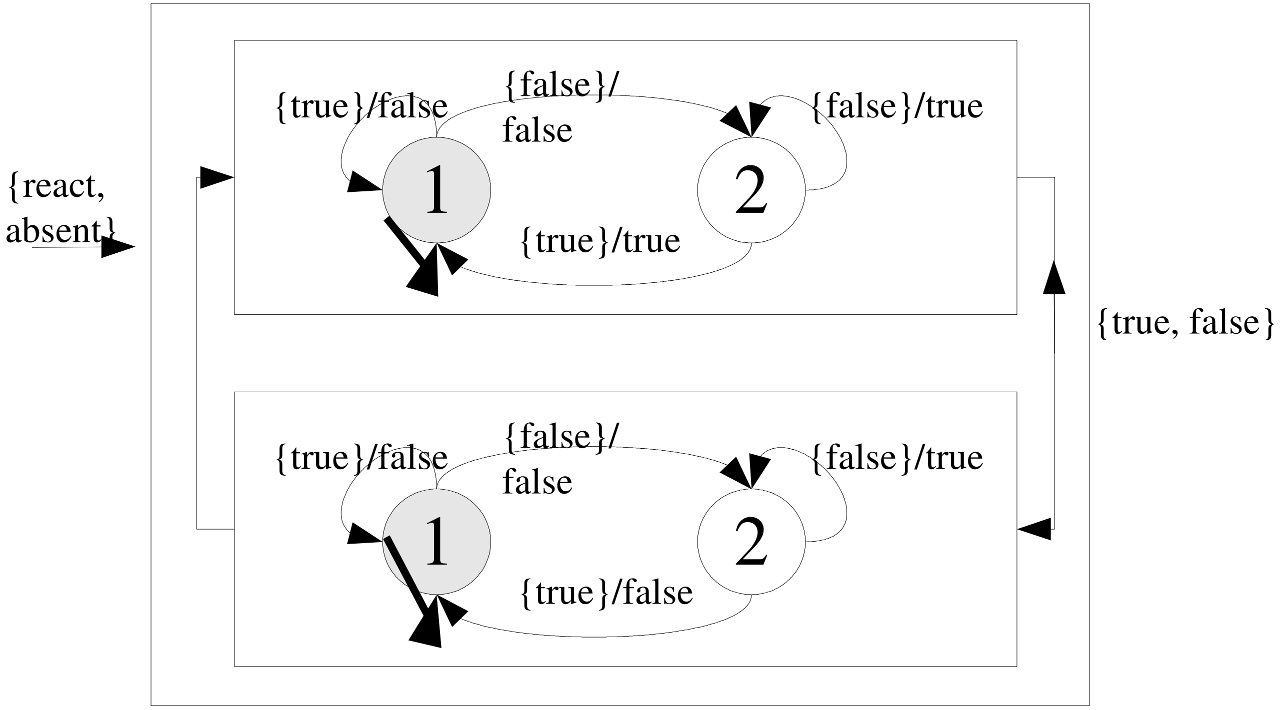
Example

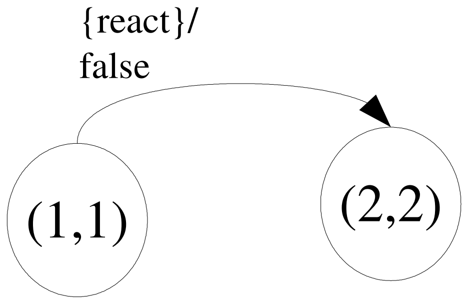
- State-determined outputs ensure well-formedness also in more complex situations



Example - (continued)

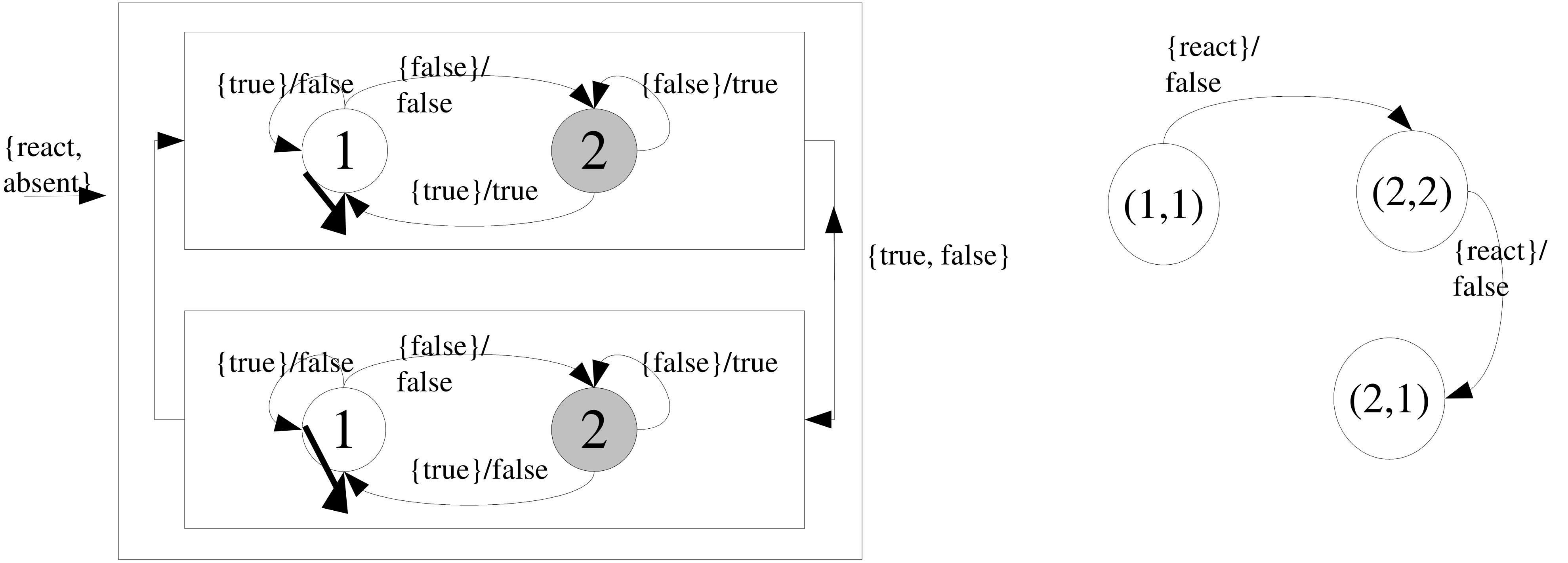
- Suppose both machines are in their initial states
  - The first emits false, which is propagated by the second:
    - Both go to 2





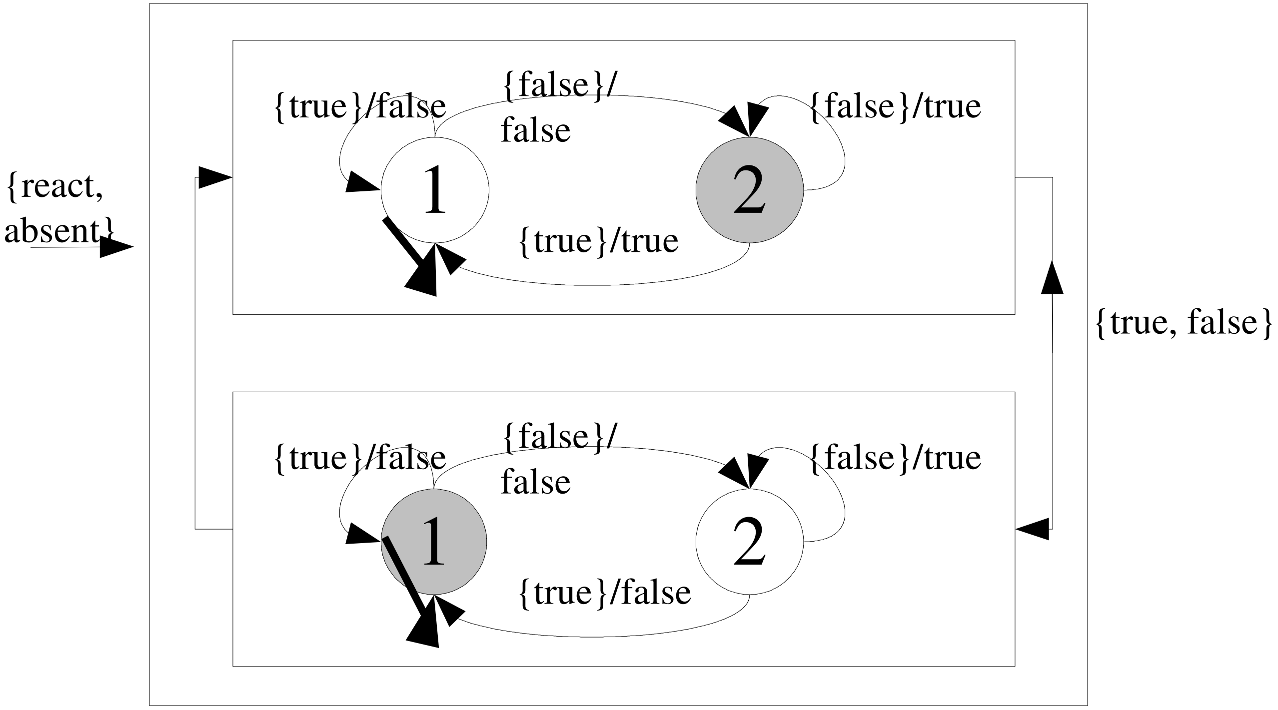
Example - (continued)

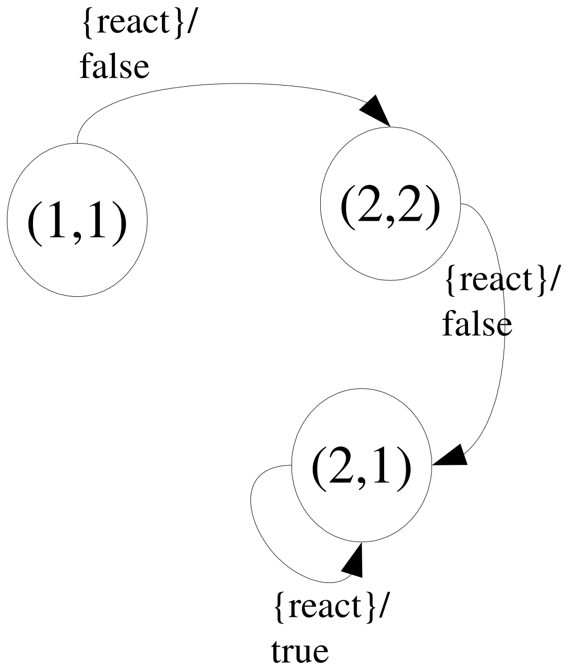
- Now, suppose both are in 2:
  - The top one emits true and the bottom one emits false
    - The top 1 remains in 2 and the bottom one goes to 1



Example - (continued)

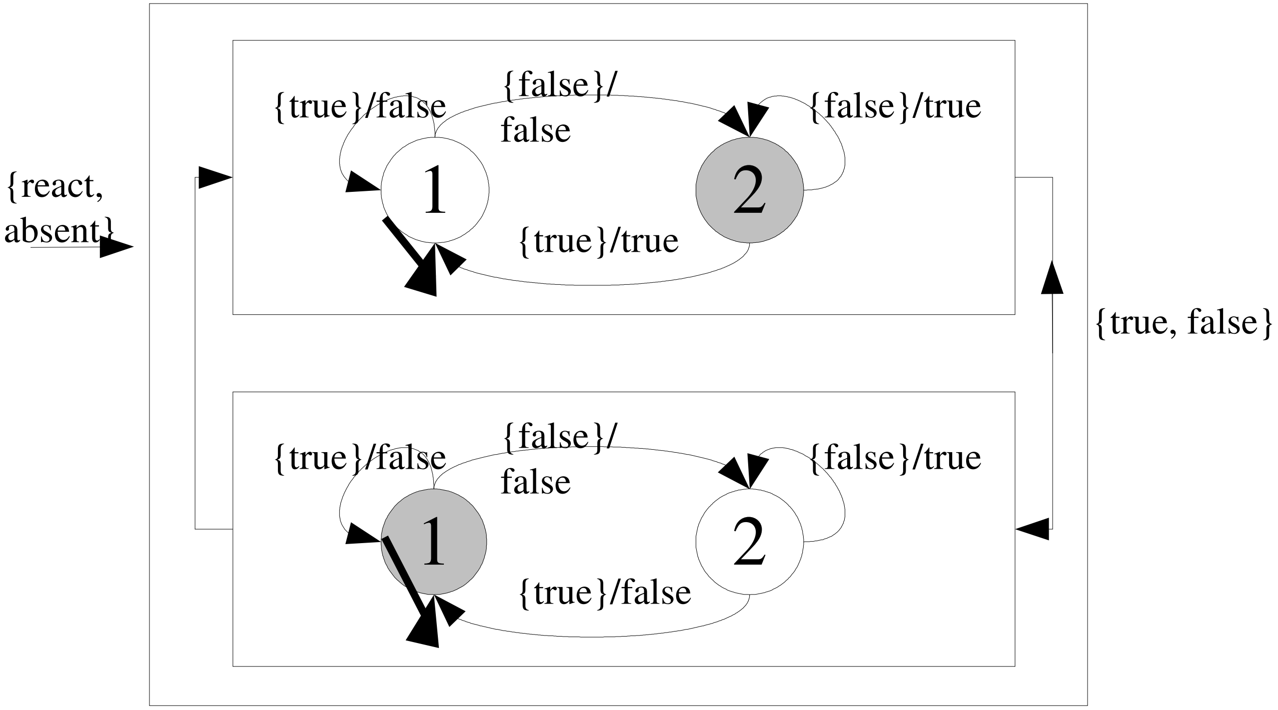
- Now, suppose top machine is in 2 and the bottom be in 1:
  - The top one emits true and the bottom one emits false
    - The top 1 remains in 2 and the bottom one remains in 1

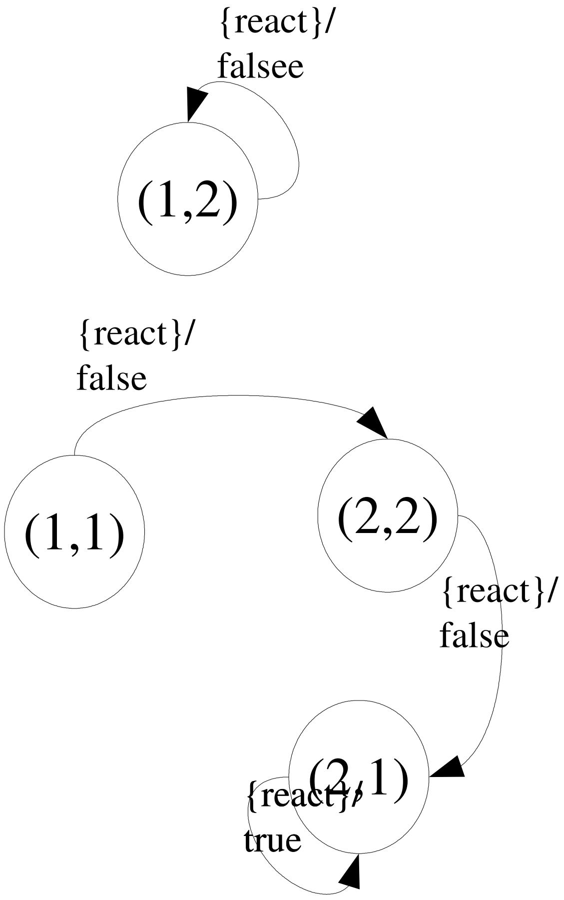




Example - (continued)

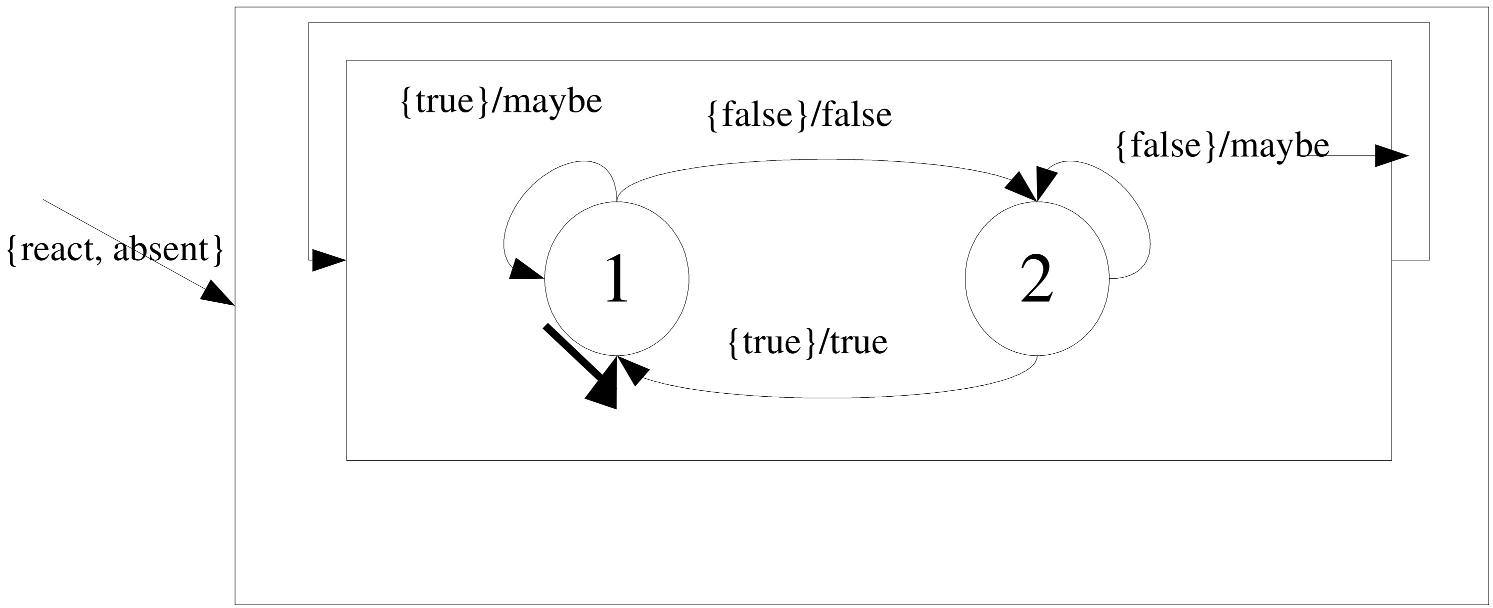
- The state (1,2) is unreachable



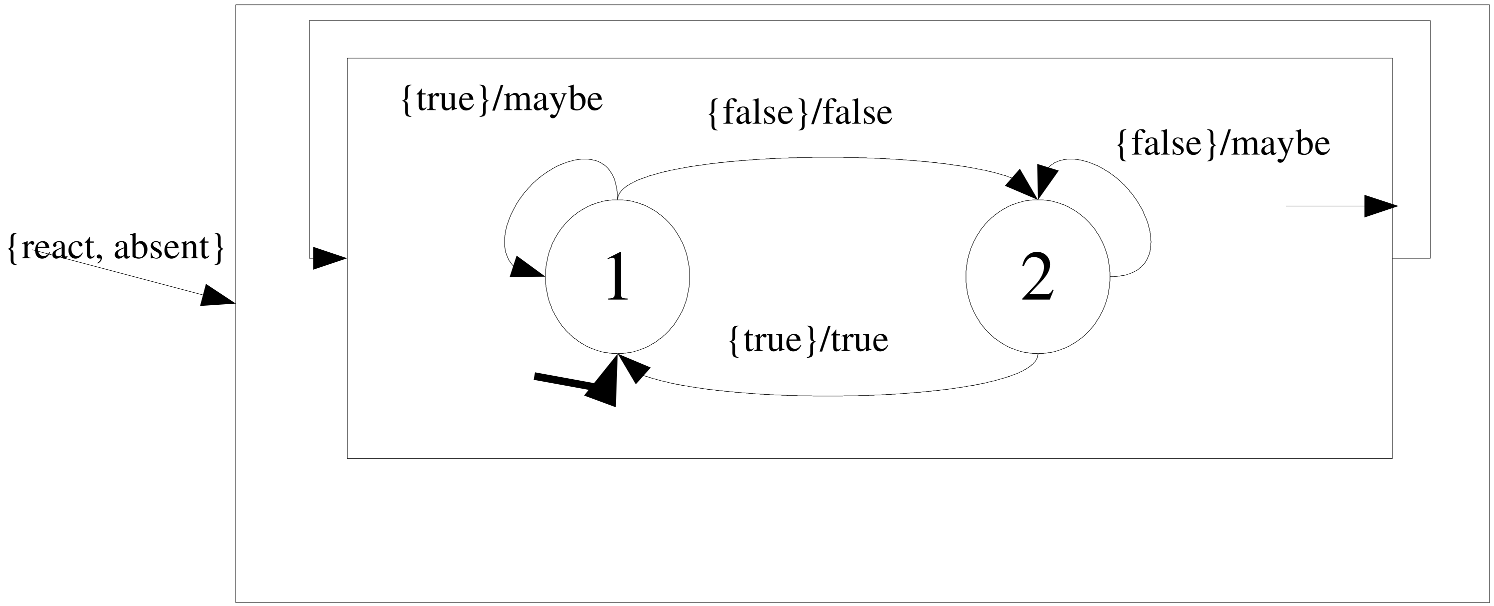


Consideration

- State-determined outputs is not a necessary condition for well-formedness
- The machine below is not state-determined ouput



Consideration

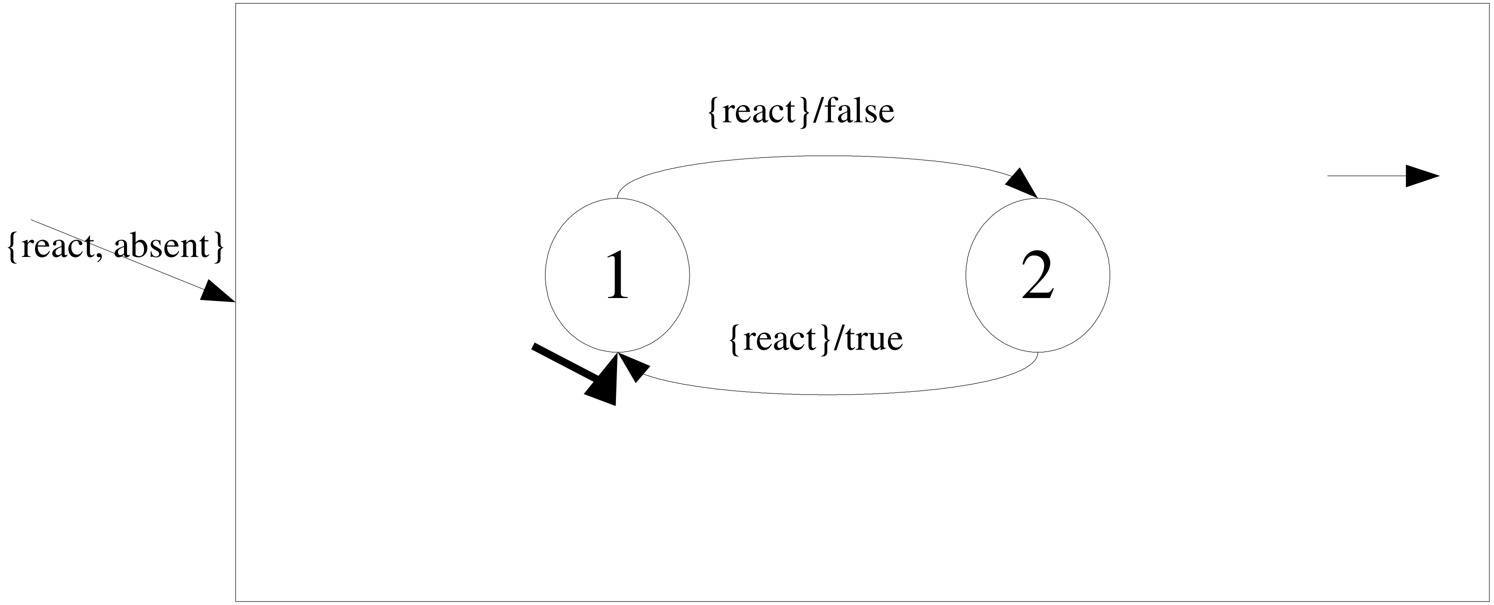


$$output_A(1, false) = false\,,\; ouput_A(2, true) = true$$

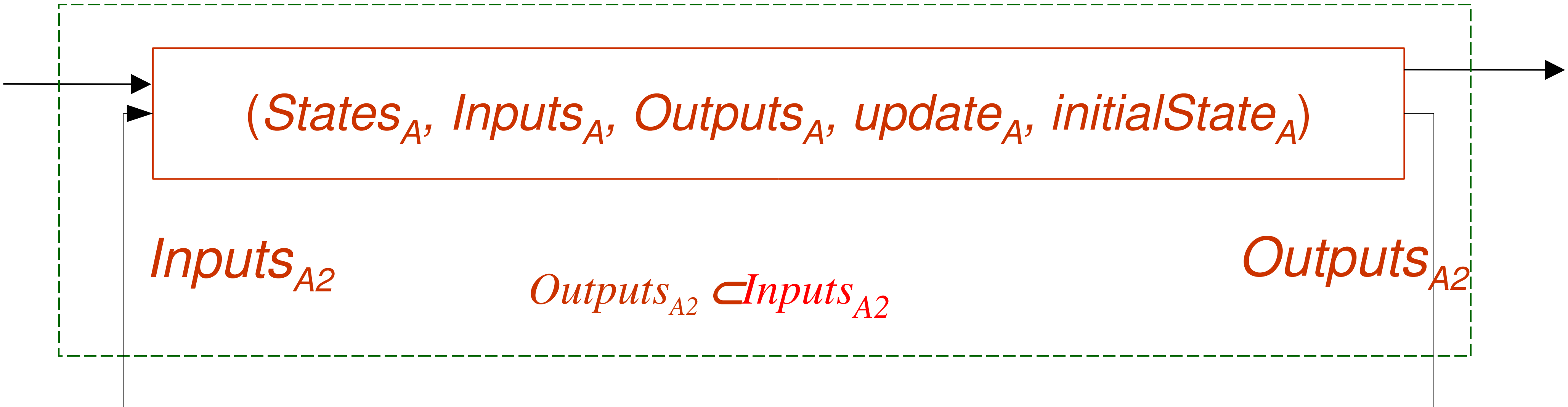
One fp for each state

Consideration - (continued)

Equivalent machine



Feedback with inputs



- Inputs and outputs can be expressed in product form

$$Inputs_A = Inputs_{A1} \times Inputs_{A2}$$
$$Ouptus_A = Outputs_{A1} \times Outputs_{A2}$$
$$Outputs_{A2} \subset Inputs_{A_1}$$
$$output_A = (output_{A1}, output_{A2})\ where$$
$$ouput_{A1} : states_A \times Inputs_A \rightarrow Ouputs_{A1}$$
$$ouput_{A2} : states_A \times Iinputs_A \rightarrow Ouputs_{A2}$$

The FP problem

- The fixed problem is two find the unknown y=(y1,y2) s.t.

$$output_A(s(n),(x_1(n),y_2(n)))=(y_1(n),y_2(n))$$

, Which is equivalent to

$$output_{A1}(s(n),(x_1(n),y_2(n)))=y_1(n)$$
$$output_{A2}(s(n),(x_1(n),y_2(n)))=y_2(n)$$

Composition well formed if there is a unique solution to this!

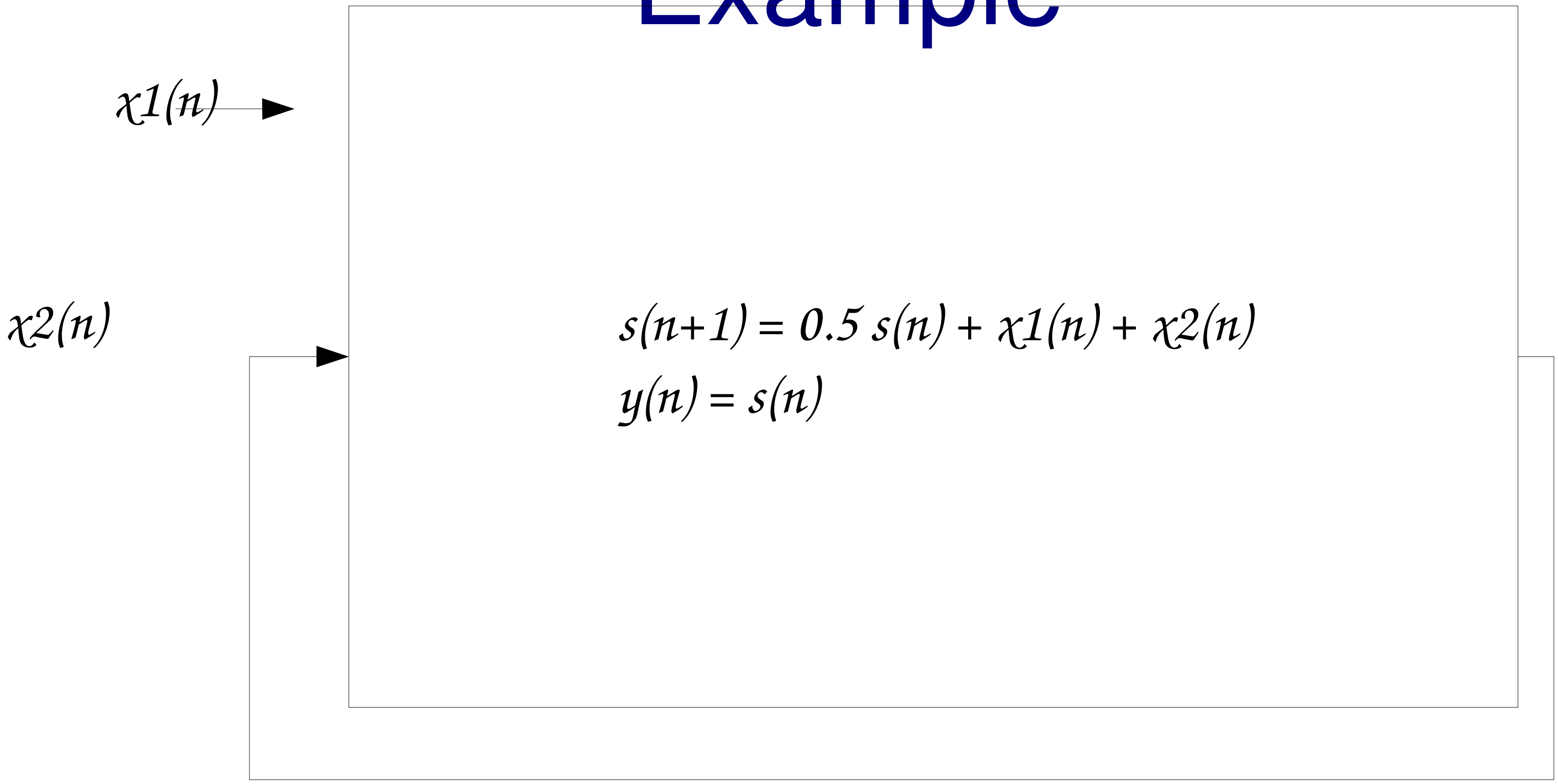
x1 and s(n) are known: this is the crucial equation!

The FP problem - (continued)

- If the composition is well formed it is described by

$$States = States_A$$
$$Inputs = Inputs_{A1}$$
$$Outputs = Outputs_{A1}$$
$$initialState = initialState_A$$
$$update(s(n), x(n)) = (nextState(s(n), x(n)), output(s(n), x(n)))$$
$$nextState(s(n), x(n)) = nextState_A(s(n), (x(n), y_2(n)))$$
$$output(s(n), x(n)) = output_A(s(n), (x(n), y_2(n))) \, where$$
$$y_2(n) = output_{A2}(s(n), (x(n), y_2(n)))$$

Example



$$Inputs_A = Real \times Real$$
$$Outputs_A = Reals$$
$$States_A = Reals$$

Example - (continued)

$x(n)$

$$s(n+1) = 1.5\, s(n) + x(n)$$
$$y(n) = s(n)$$

$$output_A(s(n), (x_1(n), x_2(n))) = x_2(n)$$
$$y(n) = outputs_A(s(n), (x_1(n), x_2(n))) = s(n)\ which\ yields$$
$$x_2(n) = s(n)\ thus$$
$$update(s(n), x(n)) = (0.5\, s(n) + x(n) + s(n), s(n)) = (1.5\, s(n) + x(n), s(n))$$

Outline

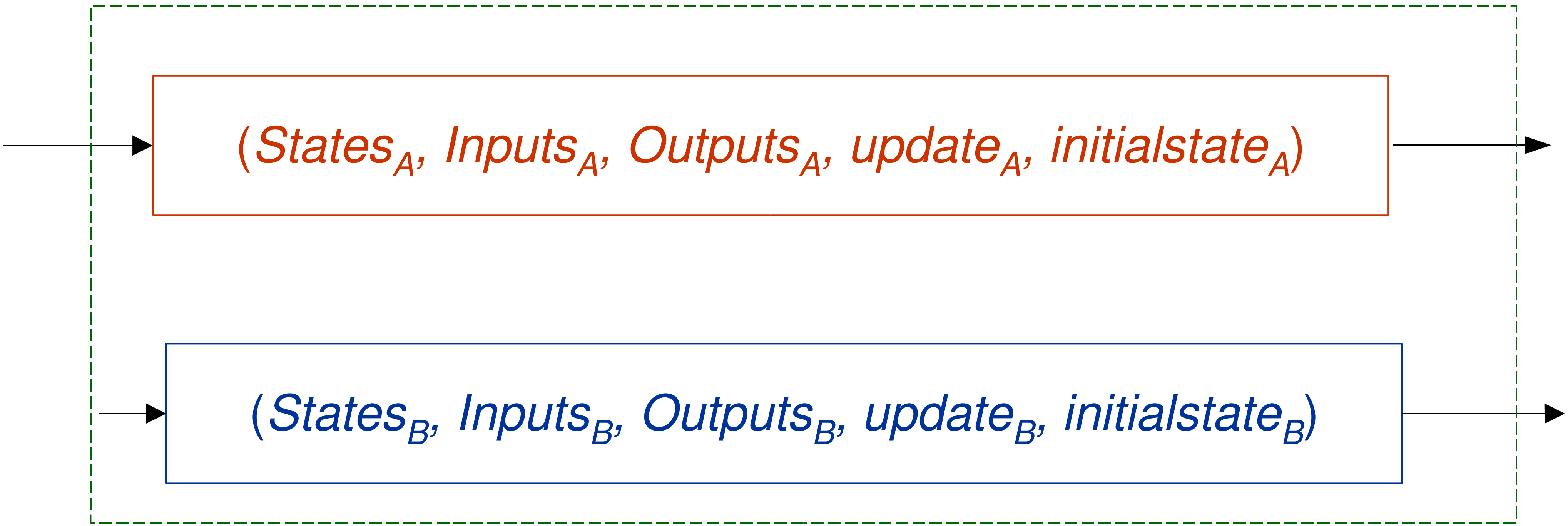
- Something more about simulation
- Connections of SM
- Feedback
- State-charts
- Implementation

Limitations of “bare” FSM

- By bare FSM we mean a single state-machine
- Big problems with “bare” FSMs are:
  - Expressing concurrent behaviours
  - Mastering complexity: hiearhchical organization
- Both issues are of utmost importance in embedded systems design

Concurrency

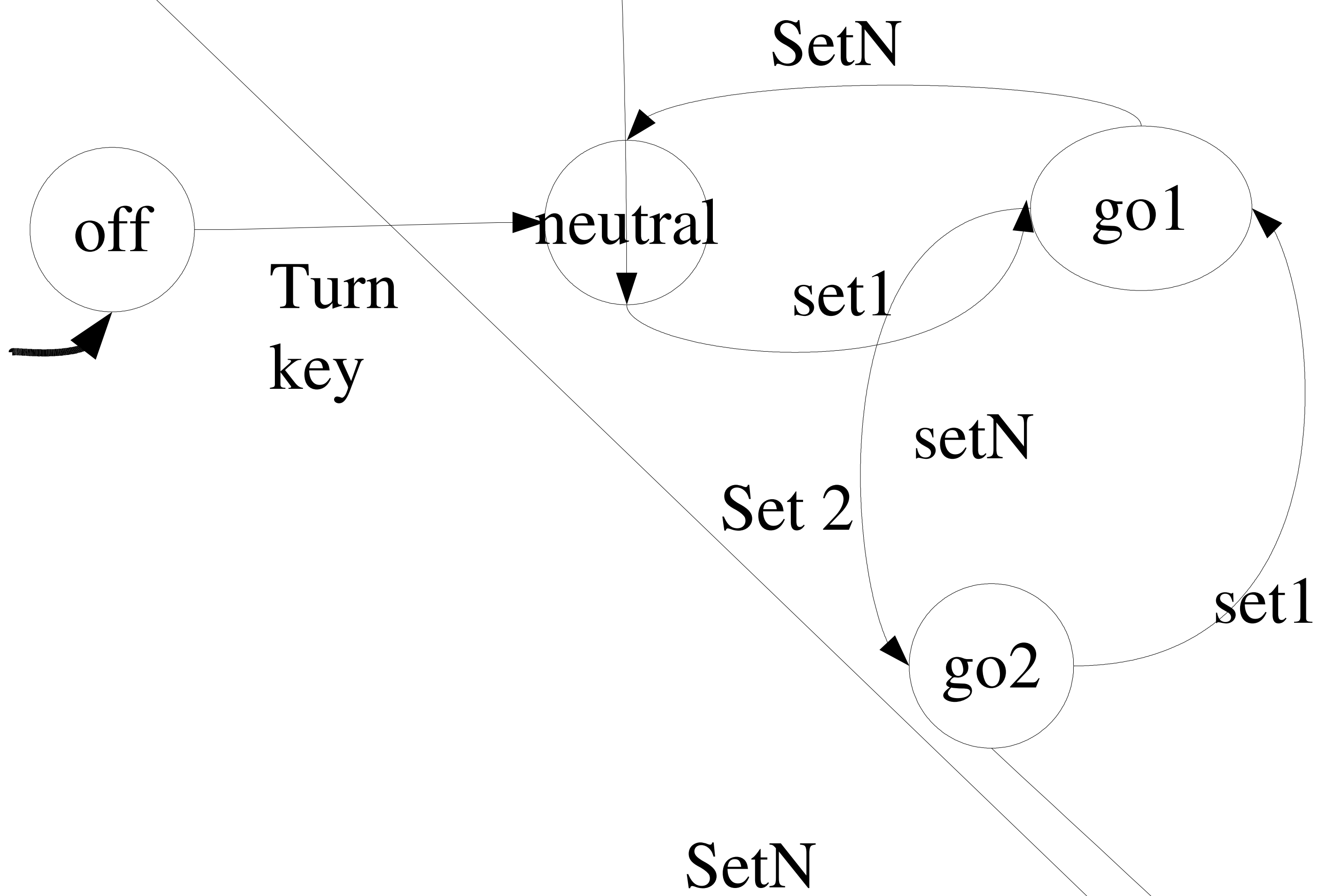
- Product machines, under the synchrony hypothesis, are inherently concurrent
- Example side by side composition
- We have seen concurrent specification sequential (one machine) implementation
- The notion of port enables also a specification for intermachine communications



Hierarchy

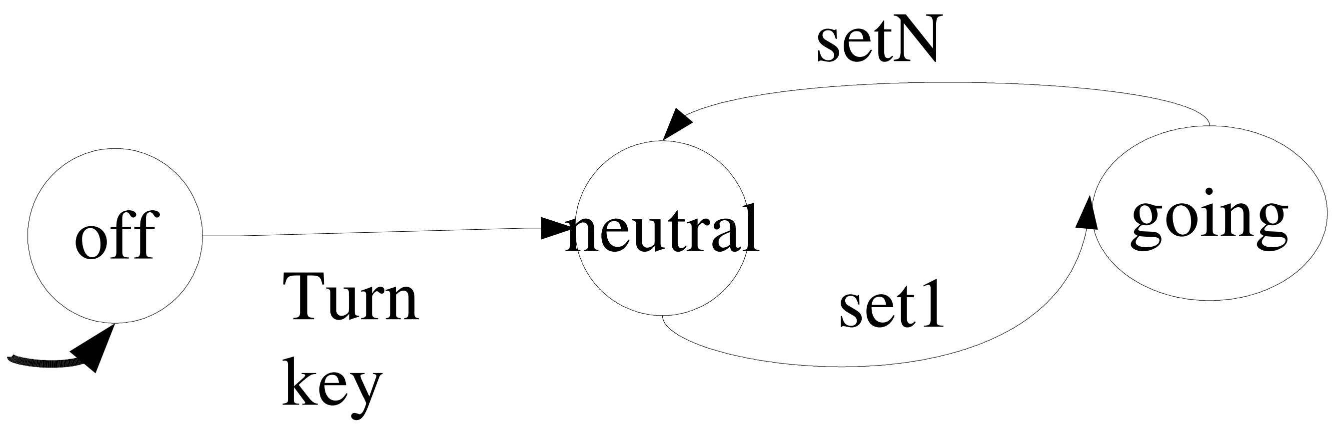
- For complex systems state-machine often become unreadable
- We need the ability of estabilishing a hiearchy on the state diagrams
- This is a purely syntactical operation that leads to interesting facts

Example



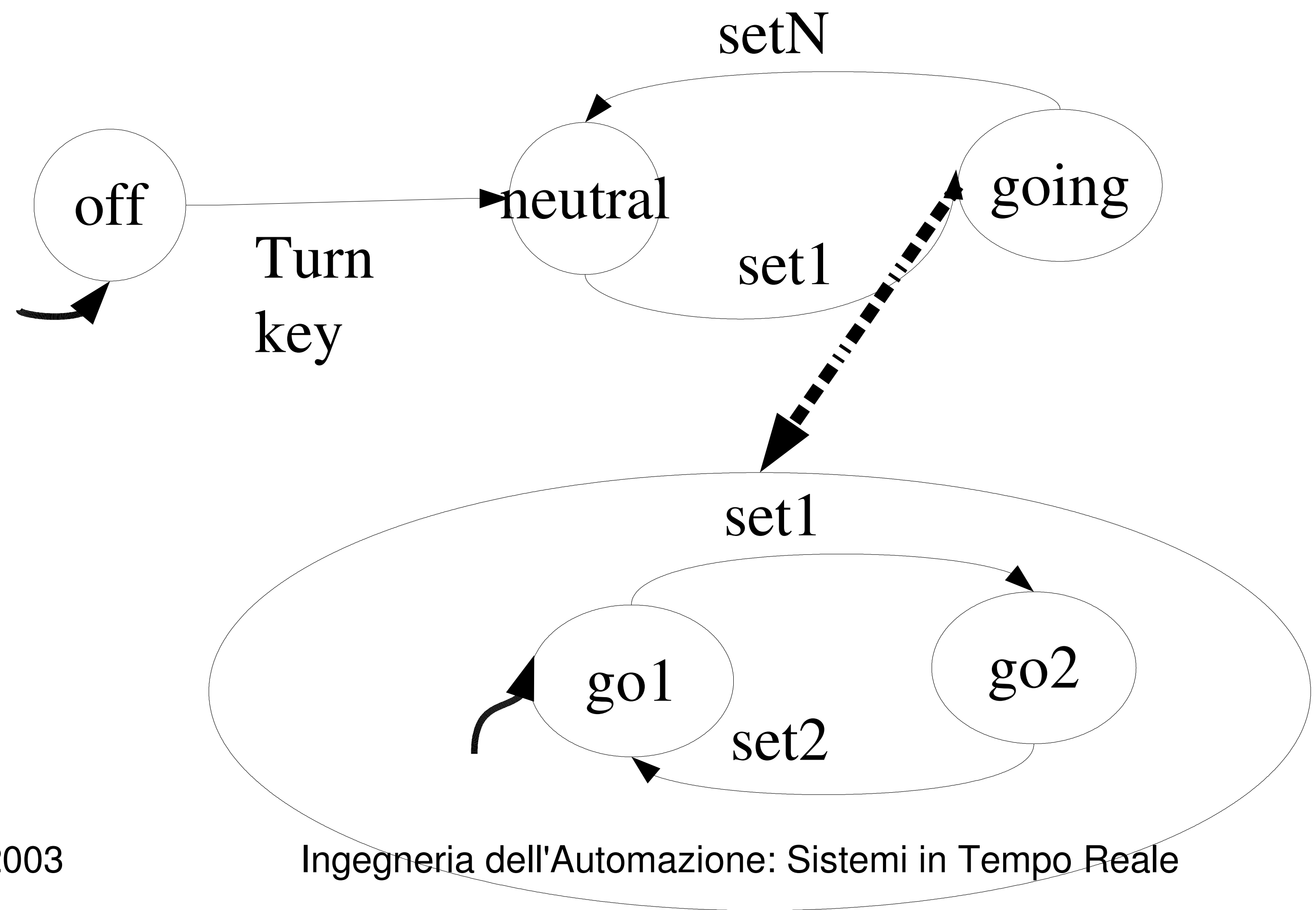
Example

- It is possible to rewrite the machine as follows:



Example

- Exploding going....

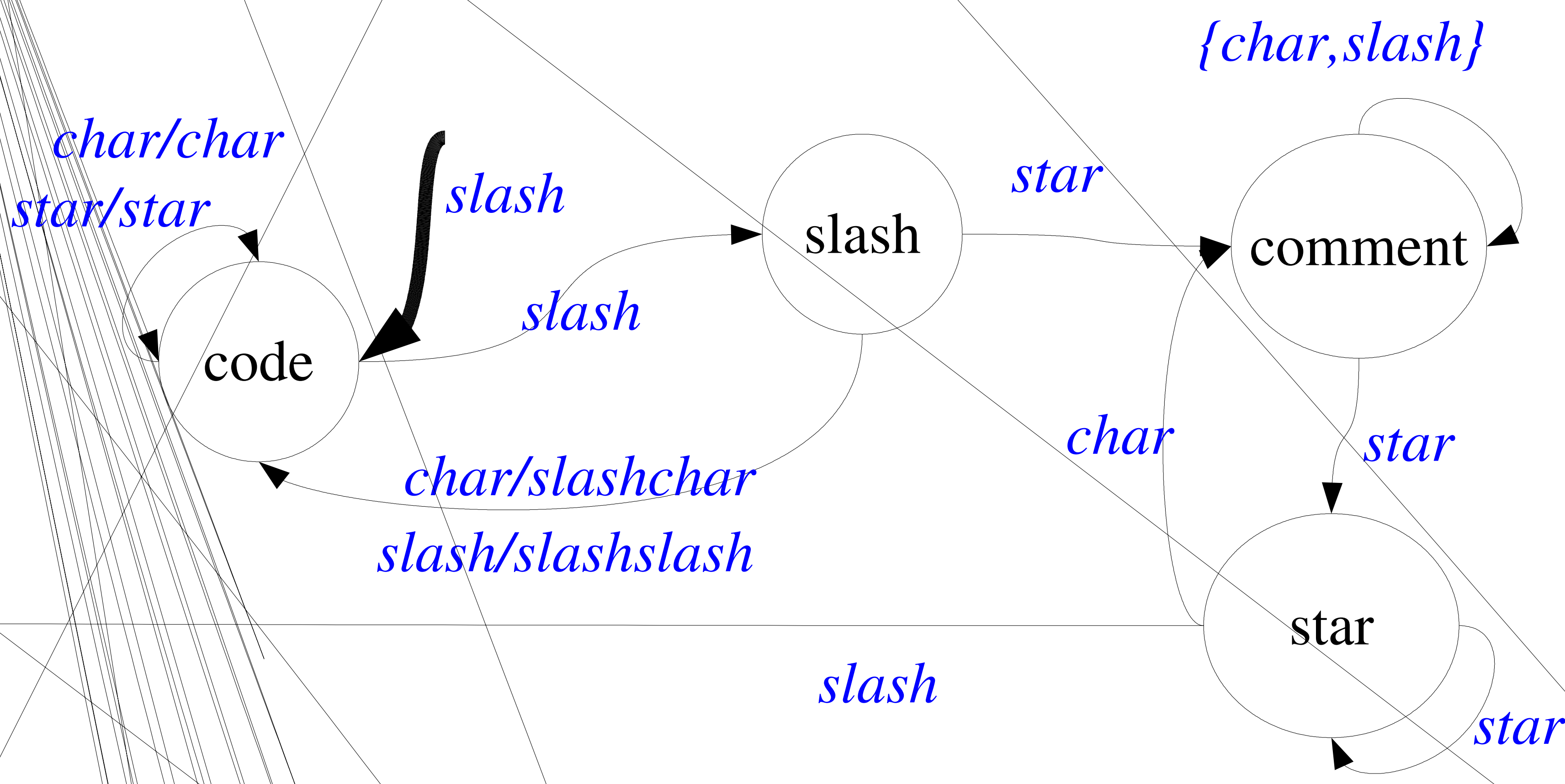


Outline

- Something more about simulation
- Connections of SM
- Feedback
- State-charts
- Implementation

Example

- C comment eraser



States and inputs

```
typedef  enum event {CHAR_SIG, STAR_SIG, SLASH_SIG, MAX_SIG} Event;

typedef enum state {CODE, SLASH, COMMENT, STAR, MAX_STATE} State;

State currState = CODE;
char currChar;
```

Modeling Transitions

```
typedef void (*Action)();
typedef struct tran {
        Action action;
        State nextState;
} Tran;
Tran Table [MAX_STATE][MAX_SIG]={
        {{&outChar, CODE},
        {&outChar,CODE},
        {&doNothing,SLASH}},
        {{&outSlashChar,CODE},
        {&doNothing,COMMENT},
        {&outSlashSlash,CODE}},
        {{&doNothing,COMMENT},
        {&doNothing,STAR},
        {&doNothing,COMMENT}},
        {{&doNothing,COMMENT},
        {&doNothing,STAR},
        {&doNothing,CODE}}
}
```

Modeling Transitions (continued)

```
void doNothing() {};
void outChar() {
        printf("%c",currChar);
};

void outSlashChar() {
        printf("/%c",currChar);
};

void outSlashSlash() {
        printf("//");
};
```

Engine

```
void process(Event e) {
	Tran * t = &(Table[currState][e]);
	t->action();
	currState = t->nextState;
};

void main() {
	Event sig;
	while(!feof(stdin)){
		scanf(“%c”, &currChar);
		if (currChar=='/')
			sig = SLASH_SIG;
		else
			if (currChar=='*')
				sig=STAR_SIG;
		else
			currChar = CHAR_SIG;
		process(sig);
}
```