System Specification, Verification and Synthesis (SSVS) – CS 4830/7485, Fall 2019

3: Formal System Modeling: State Machines

Stavros Tripakis

Northeastern University
Khoury College of Computer Sciences

Outline

- Finite State Machines (FSMs)
- Moore and Mealy machines
- Modeling digital circuits as FSMs
- State machines in nuXmv

Reminder: what is a system? (so far)

- **System**: state + dynamics (+ inputs/outputs)
- **Dynamics**: rules defining how state evolves in time

STATE MACHINES

Finite State Machines of type Moore and Mealy

A finite state machine (FSM) is a tuple

$$(I, O, S, s_0, \delta, \lambda)$$

- I: finite set of inputs
- O: finite set of outputs
- S: finite set of states
- $s_0 \in S$: initial state
- $\delta : S \times I \to S$: transition function
- λ: output function
  - If the FSM is of type **Moore**:
  - If the FSM is of type **Mealy**:

Finite State Machines of type Moore and Mealy

A finite state machine (FSM) is a tuple

$$(I, O, S, s_0, \delta, \lambda)$$

- I: finite set of inputs
- O: finite set of outputs
- S: finite set of states
- $s_0 \in S$: initial state
- $\delta : S \times I \to S$: transition function
- λ: output function
  - If the FSM is of type **Moore**:

$$\lambda : S \to O$$

  - If the FSM is of type **Mealy**:

Finite State Machines of type Moore and Mealy

A finite state machine (FSM) is a tuple

$$(I, O, S, s_0, \delta, \lambda)$$

- I: finite set of inputs
- O: finite set of outputs
- S: finite set of states
- $s_0 \in S$: initial state
- $\delta : S \times I \to S$: transition function
- λ: output function
  - If the FSM is of type **Moore**:

$$\lambda : S \to O$$

  - If the FSM is of type **Mealy**:

$$\lambda : S \times I \to O$$

Parenthesis: Basic Logic and Math Notations

- Sets: e.g., $A = \{1, 2, 3\}$, $B = \{0, 2, 4, ...\}$, $C = \{n \mid n \text{ is a non-negative even number}\}$
- Set membership: $x \in A$: x is an element of set A; $x \notin A$: x is not an element of A
- $x \notin A \equiv \neg(x \in A)$
- Boolean logic: $\phi_1 \wedge \phi_2$ (conjunction, "ϕ_1 and ϕ_2"), $\phi_1 \vee \phi_2$ (disjunction, "ϕ_1 or ϕ_2"), $\neg\phi$ (negation, "not ϕ"), $\phi_1 \Rightarrow \phi_2$ (implication, "if ϕ_1 then ϕ_2", also written $\phi_1 \rightarrow \phi_2$), $\equiv$ (equivalence, "ϕ_1 iff ϕ_2", also written $\phi_1 \Leftrightarrow \phi_2$)
- Subset: $A \subseteq B$: every element of A is also an element of B, i.e., $\forall x : x \in A \Rightarrow x \in B$
- First-order logic: $\forall x : \phi$ (universal quantification, "ϕ holds for any x"), $\exists x : \phi$ (existential quantification, "ϕ holds for some x")
- Set equality: $A = B$ iff $A \subseteq B$ and $B \subseteq A$
- Strict subset: $A \subset B$ iff $A \subseteq B$ and $A \neq B$
- Set operations: $A \cup B$ (union), $A \cap B$ (intersection), $\overline{A}$ (complement), $A - B$ or $A \setminus B$ (set difference), $A \times B$ (cartesian product)
- $A \cup B = \{x \mid x \in A \vee x \in B\}$
- $A \cap B = \{x \mid x \in A \wedge x \in B\}$
- $\overline{A} = \{x \mid x \notin A\} = U - A$ (assuming some universe U)
- $A - B = \{x \mid x \in A \wedge x \in B\} = A \cap \overline{B}$
- $A \times B = \{(x, y) \mid x \in A \wedge y \in B\}$
- (Binary) Relations: $R \subseteq A \times B$
- Functions: $f : A \rightarrow B$, a special kind of relation $f \subseteq A \times B$ such that for every $x \in A$ there is at most one $y \in B$ such that $(x, y) \in f$; this y is written $f(x)$

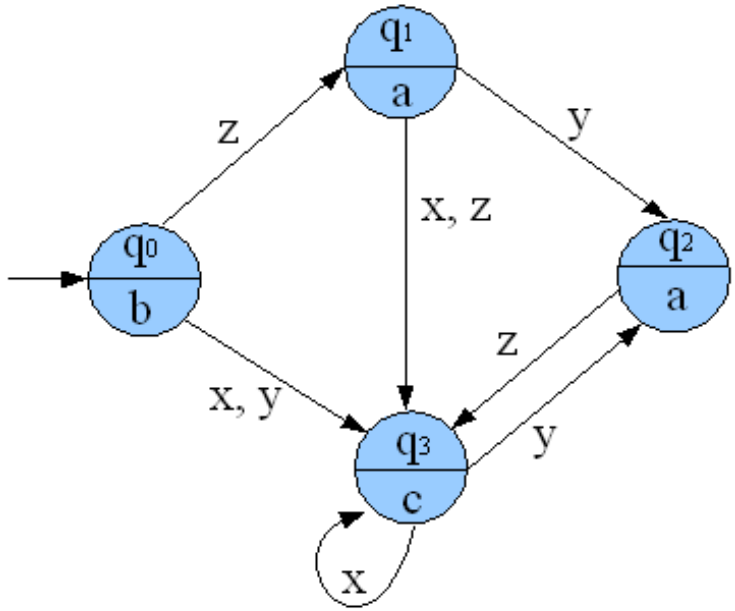
Example: a Moore Machine

States: {q0, q1, q2, q3}

Initial state: q0

Input symbols: {x,y,z}

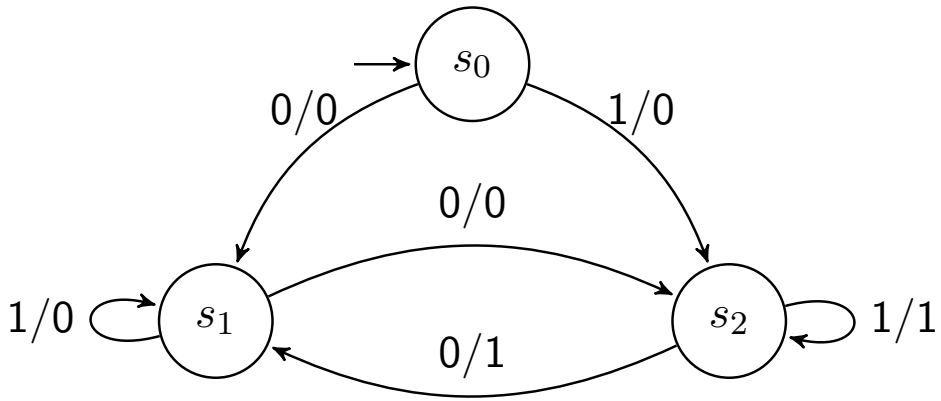
Output symbols: {a,b,c}



How are the output and transition functions defined?

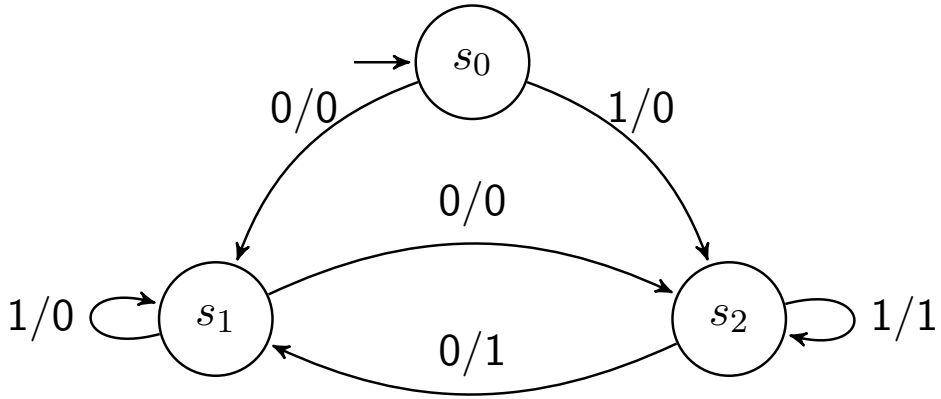
Example: a Mealy Machine

States $= \{s_0, s_1, s_2\}$, initial state $= s_0$
Inputs $=$ Outputs $= \{0, 1\}$



Example: a Mealy Machine

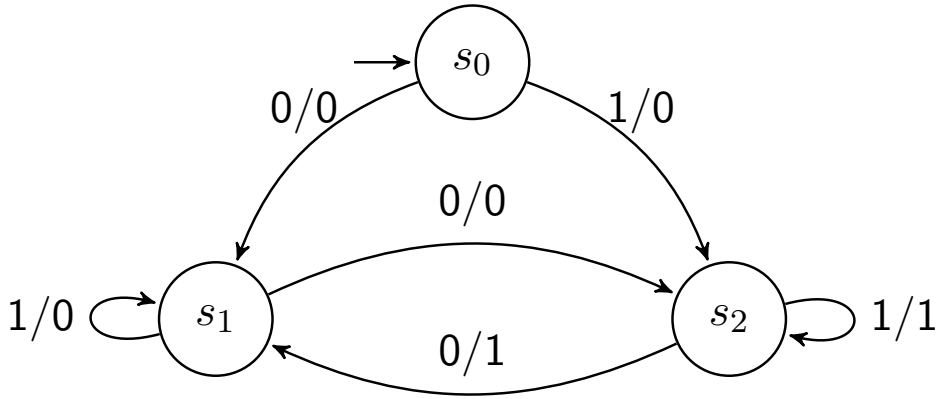
States $= \{s_0, s_1, s_2\}$, initial state $= s_0$
Inputs $=$ Outputs $= \{0, 1\}$



How are the transition and output functions defined?

Example: a Mealy Machine

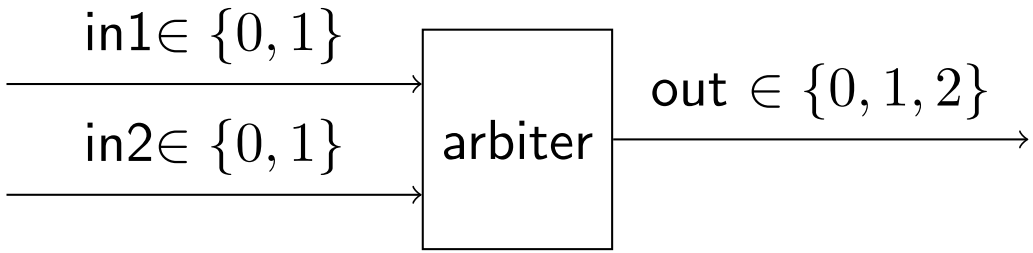
States $= \{s_0, s_1, s_2\}$, initial state $= s_0$
Inputs $=$ Outputs $= \{0, 1\}$



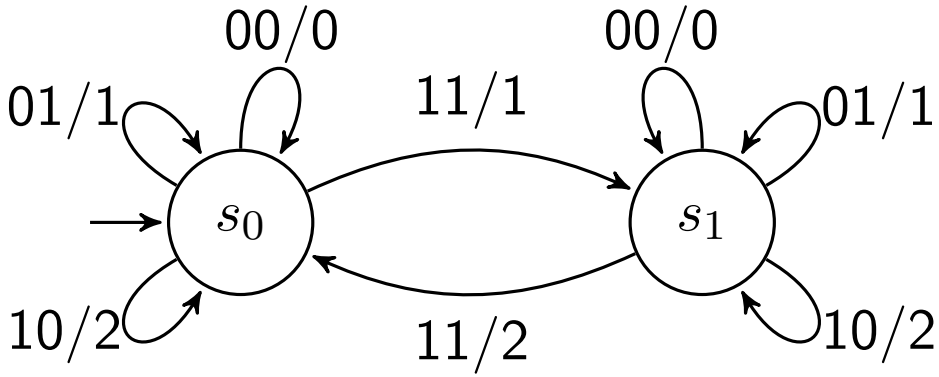
How are the transition and output functions defined?
Would it be OK to drop a few arrows in the diagram?

Example: another Mealy Machine

structure:

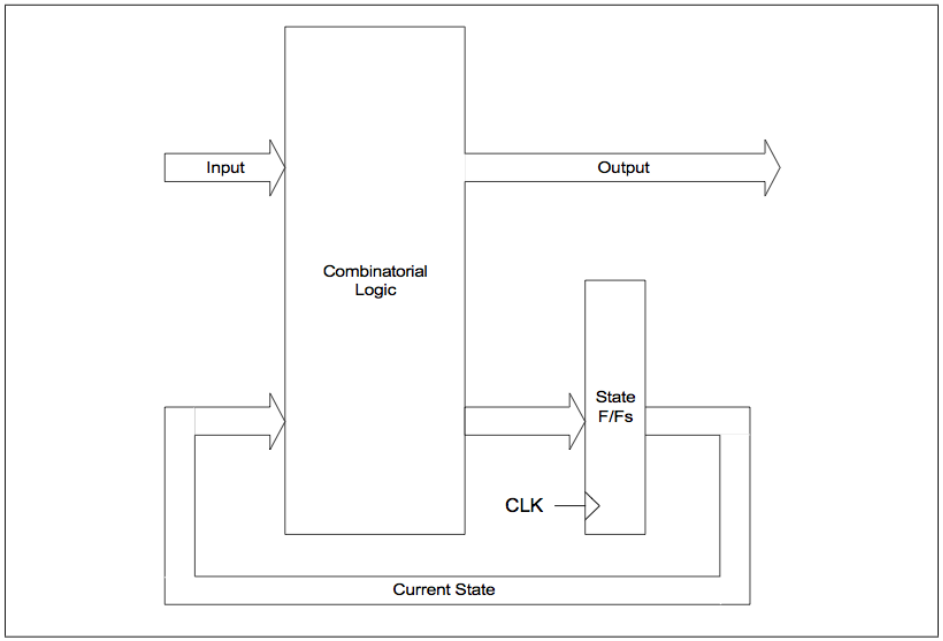


behavior:



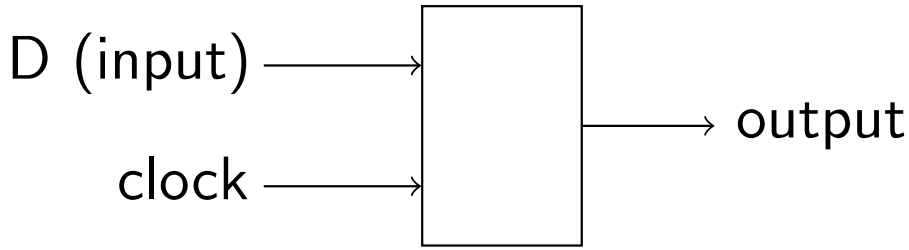
DIGITAL CIRCUITS

Synchronous Circuits – Generic structural view:



- Combinational logic part: a network of logical gates (AND, OR, NOT, XOR, ...).
- Memory/state of the circuit: some type of digital memory element (e.g., D-type flip-flop).
- Synchronous: clock arriving conceptually synchronously (simultaneously) at all flip-flops.
- Circuit: a network of connected gates and flip-flops ("netlist").

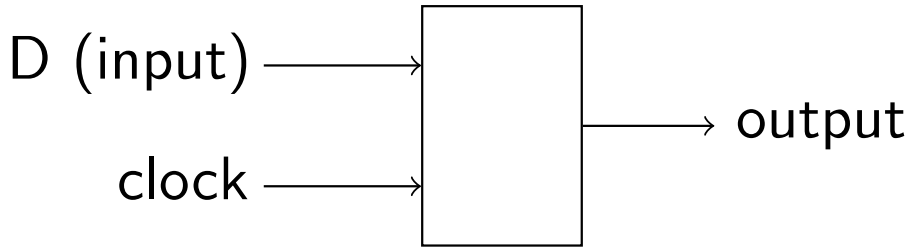
Memory element: D flip-flop



Behavior (simplified):

- Clock input defines a set of times $t_1, t_2, t_3, \ldots$ (e.g., up-edges of a periodic pulse).
- The value of output remains constant during the interval $[t_k, t_{k+1})$ and equal to the value of the input D at t_k.
- "Door-opening" metaphor.

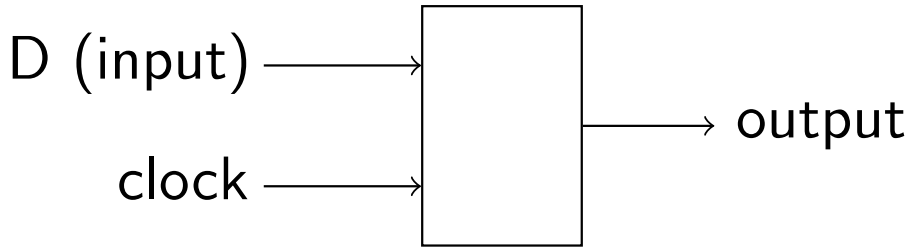
Memory element: D flip-flop



Behavior (simplified):

- Clock input defines a set of times $t_1, t_2, t_3, \ldots$ (e.g., up-edges of a periodic pulse).
- The value of output remains constant during the interval $[t_k, t_{k+1})$ and equal to the value of the input D at t_k.
- "Door-opening" metaphor.
- In real life memory elements often have more inputs (e.g., resets to initialize state).

Memory element: D flip-flop

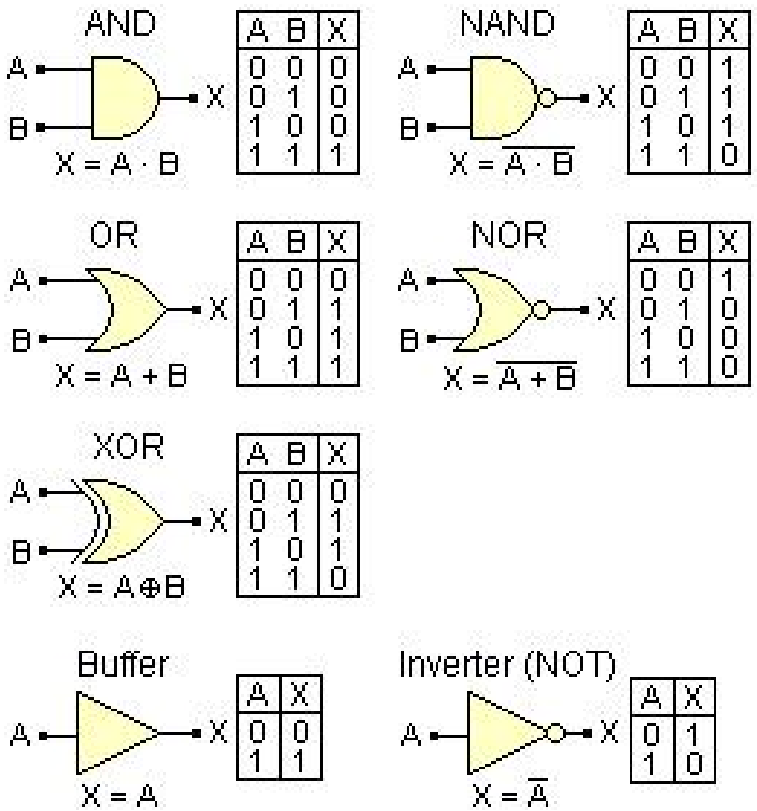


Behavior (simplified):

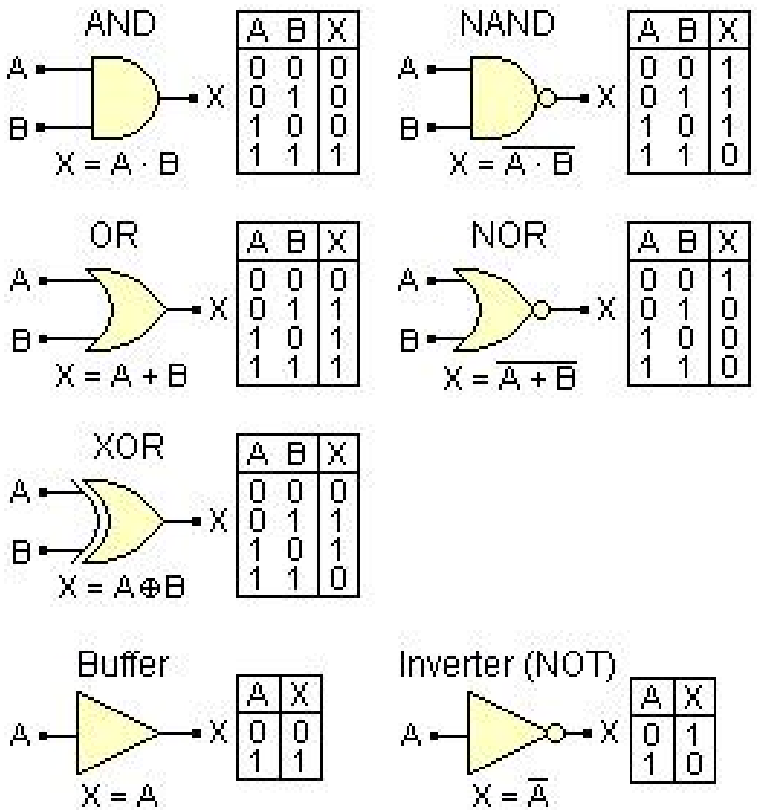
- Clock input defines a set of times $t_1, t_2, t_3, ...$ (e.g., up-edges of a periodic pulse).
- The value of output remains constant during the interval $[t_k, t_{k+1})$ and equal to the value of the input D at t_k.
- "Door-opening" metaphor.
- In real life memory elements often have more inputs (e.g., resets to initialize state).

Is the D flip-flop a state machine?

Combinational logic gates



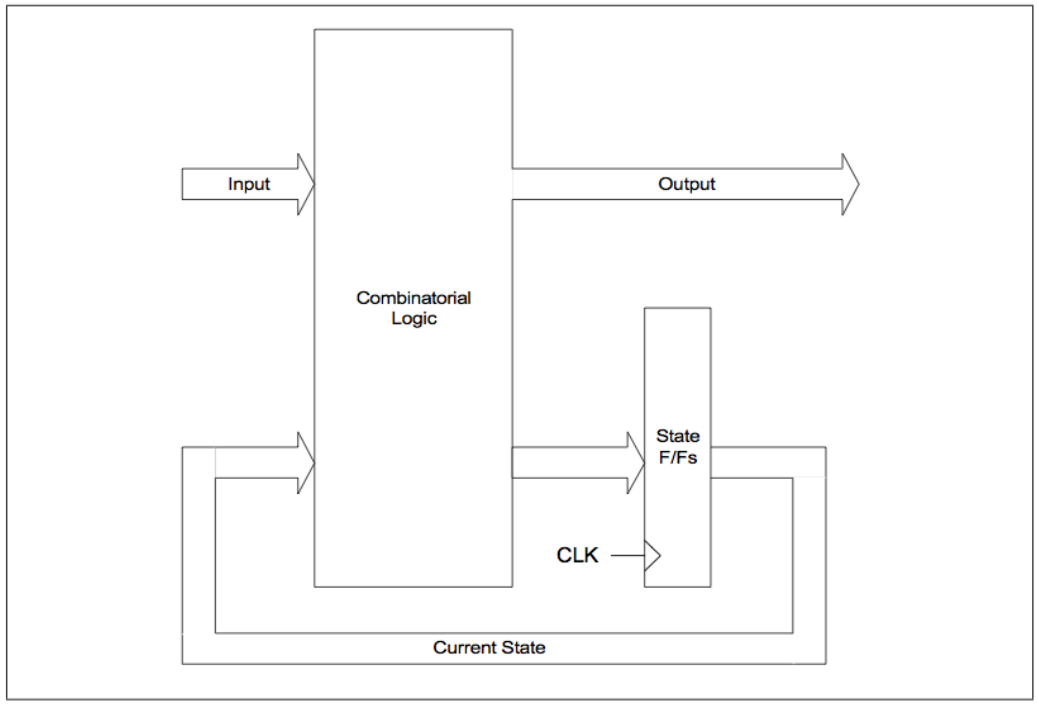
Combinational logic gates



Are combinational logic gates state machines?

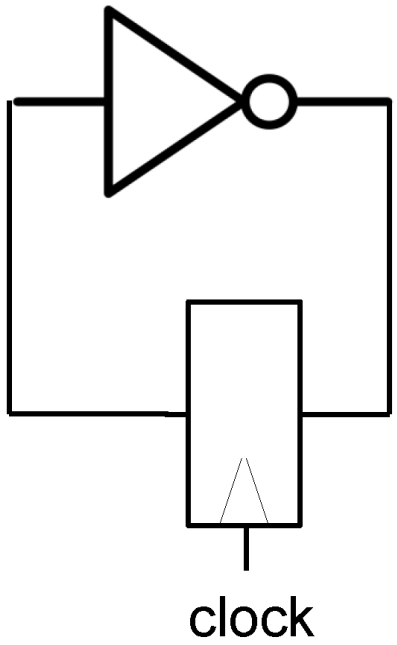
Digital Circuits: Networks of Flip-Flops and Logic Gates

For now, we consider **acyclic** circuits: they **can** have feedback, but any feedback loops are "broken" by flip-flops:



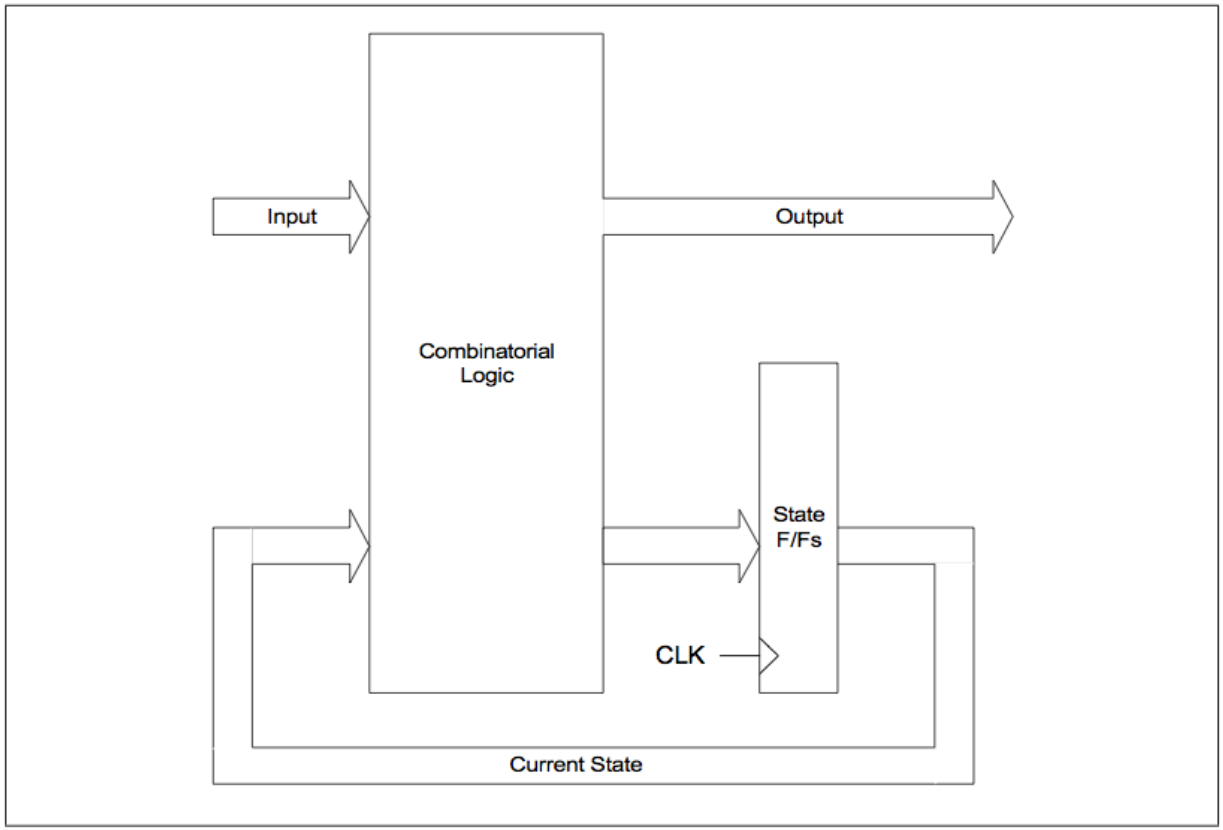
Are the dynamics of such circuits well-defined? How?

Modeling Circuits as State Machines



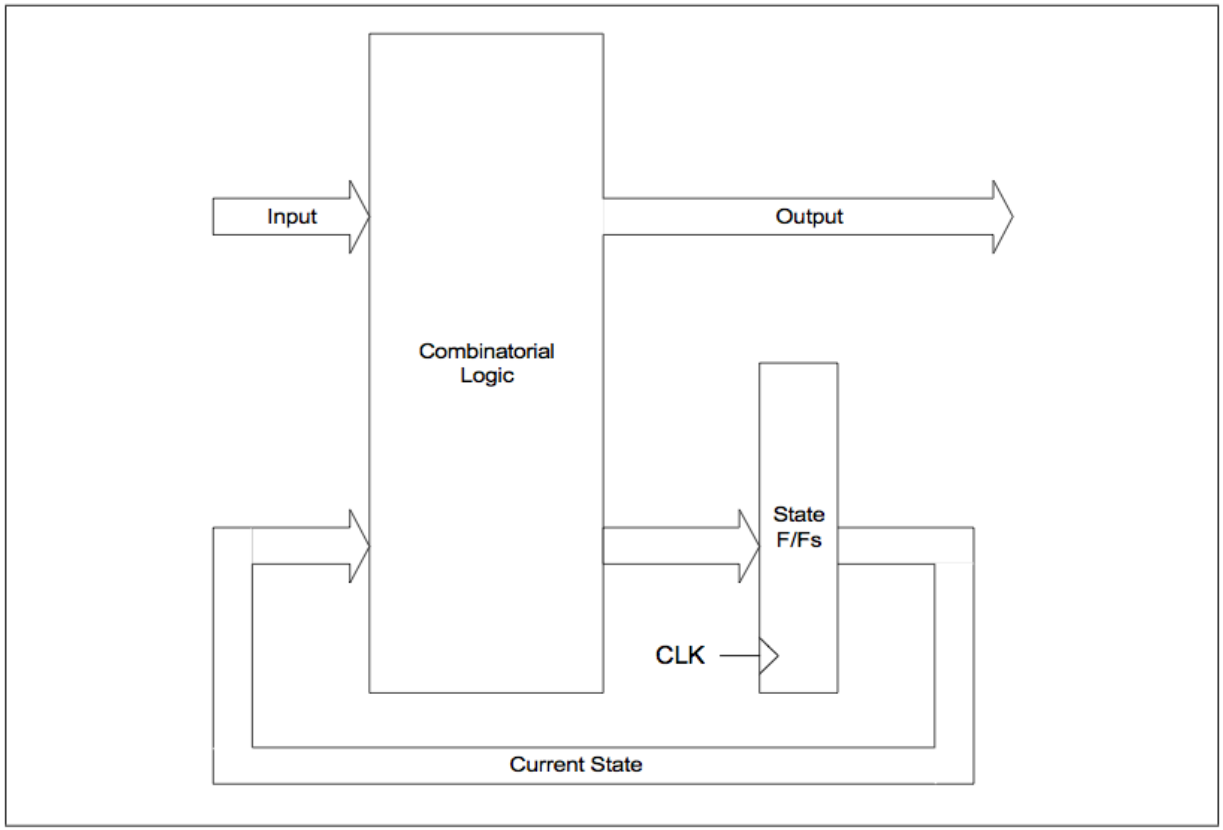
Is this a state machine?

Modeling Circuits as State Machines



Is this a state machine? Is it a Mealy or Moore machine?
How are $(I, O, S, s_0, \delta, \lambda)$ defined?

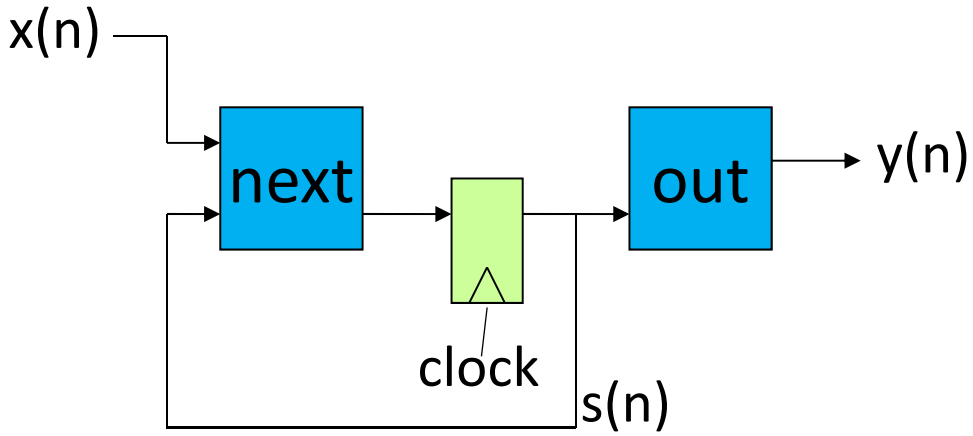
Modeling Circuits as State Machines



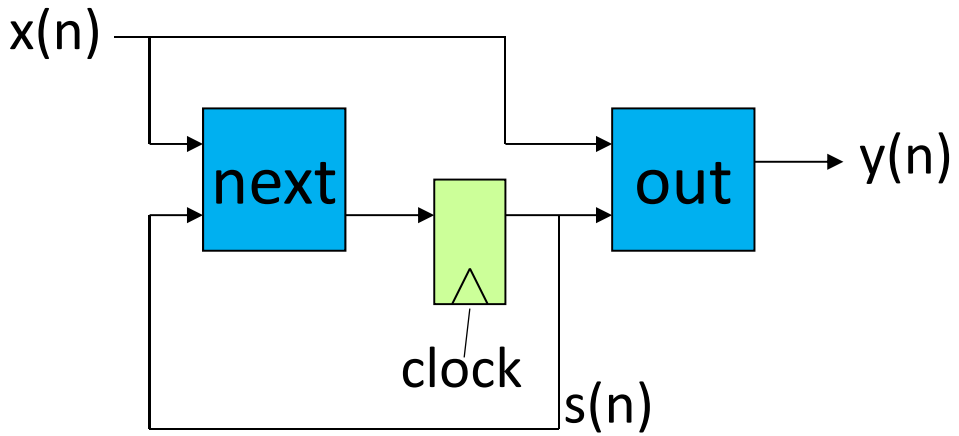
Is this a state machine? Is it a Mealy or Moore machine?
How are $(I, O, S, s_0, \delta, \lambda)$ defined?
What would a Moore Machine look like?

Moore and Mealy machines viewed as digital circuits

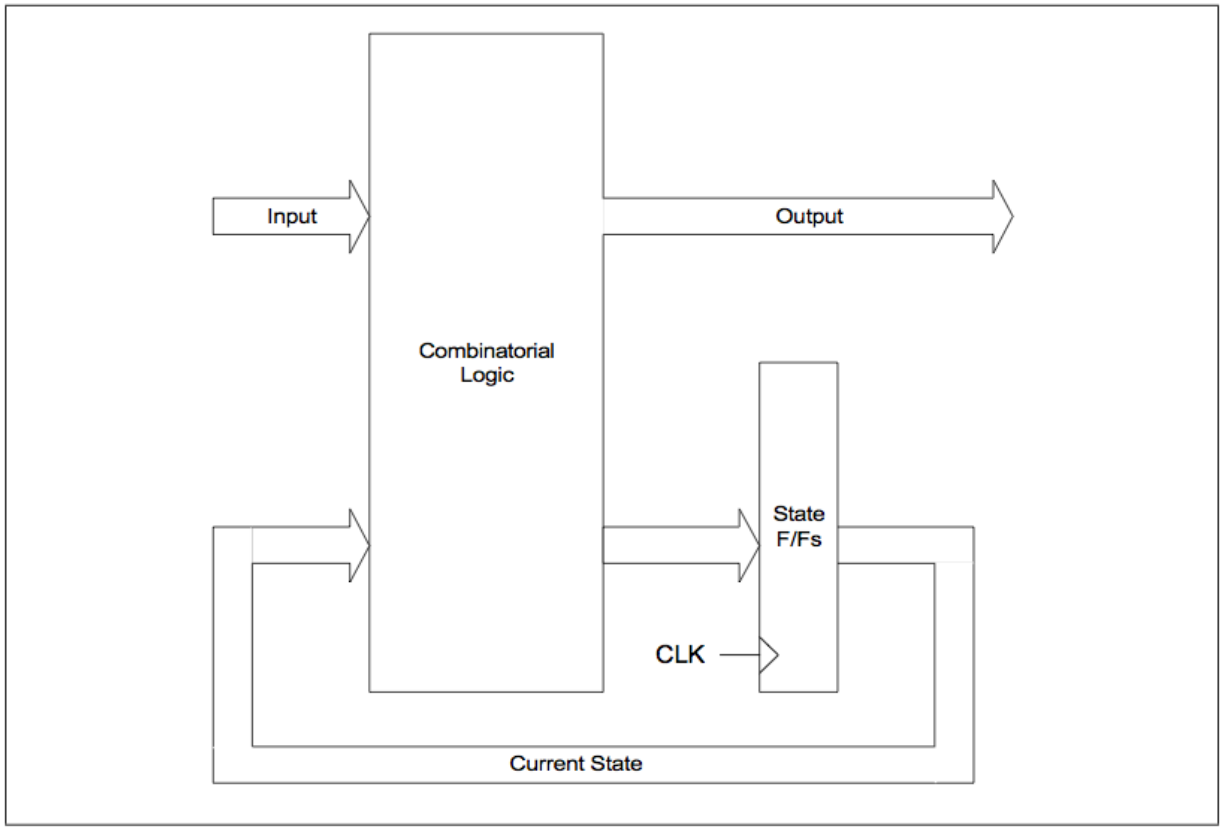
Moore machine:



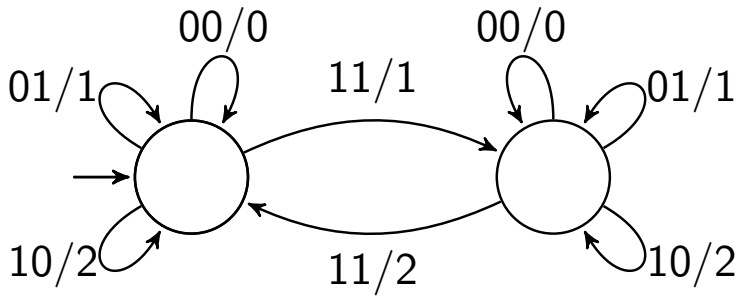
Mealy machine:



State Machines and Synchronous Circuits

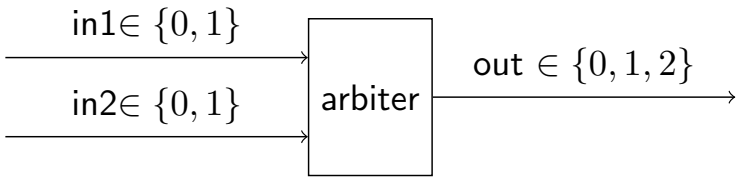


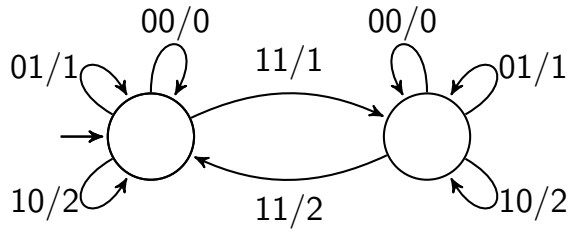
Is this a good drawing?



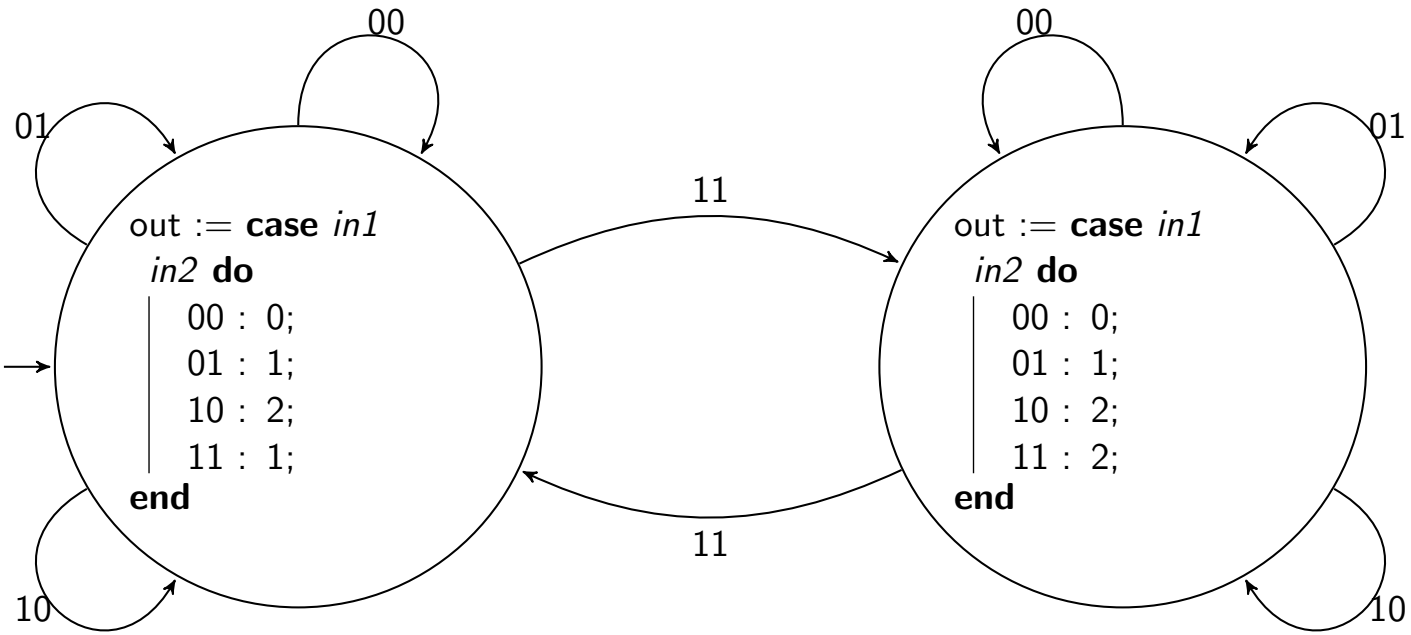
Drawing Mealy Machines Correctly

Traditional drawing mixes transition and output functions, although these are independent (this matters in the case of circuits, for instance, where outputs might change multiple times before stabilizing – c.f. discussion on circuits that follows):



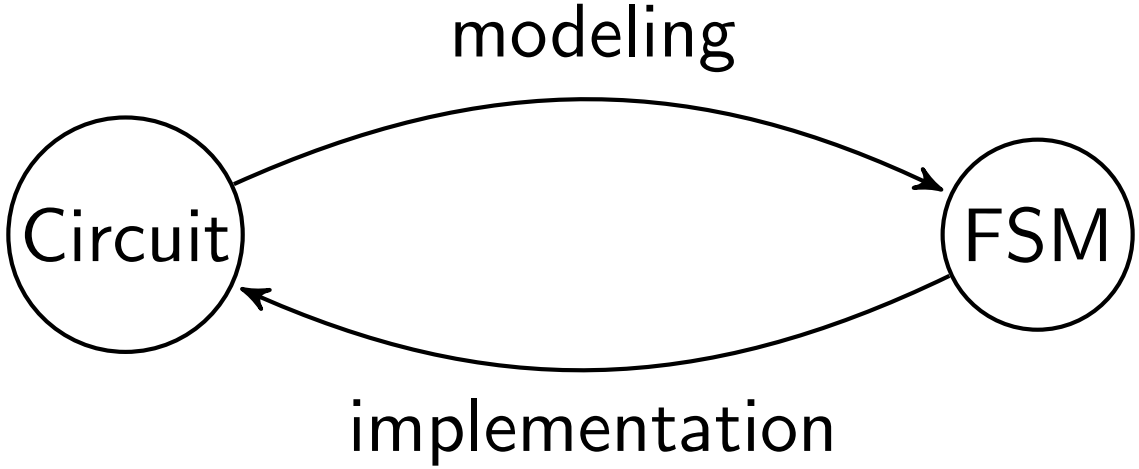


Better drawing:



Modeling vs Implementation (Logic Synthesis)

We have done the "modeling" part:



The "implementation" part: logic synthesis (part of EDA).

The synchronous language Lustre [Halbwachs et al., 1991]

An elegant way to model state machines.
A simple program in Lustre:

```
node Edge (X : bool) returns (E : bool);
let
  E = false -> X and not pre X ;
tel
```

Can you guess its meaning?

The synchronous language Lustre [Halbwachs et al., 1991]

An elegant way to model state machines.
A simple program in Lustre:

```
node Edge (X : bool) returns (E : bool);
let
  E = false -> X and not pre X ;
tel
```

Can you guess its meaning?

$$\begin{aligned} E_0 &= false \\ E_{k+1} &= X_{k+1} \wedge \neg X_k \end{aligned}$$

The synchronous language Lustre [Halbwachs et al., 1991]

An elegant way to model state machines.
A simple program in Lustre:

```
node Edge (X : bool) returns (E : bool);
let
  E = false -> X and not pre X ;
tel
```

Can you guess its meaning?

$$\begin{aligned} E_0 &= false \\ E_{k+1} &= X_{k+1} \wedge \neg X_k \end{aligned}$$

Quiz: draw the corresponding state machine.

Infinite state machines

In the program below, all variables are Boolean, therefore range over a **finite** domain:

```
node Edge (X : bool) returns (E : bool);
let
  E = false -> X and not pre X ;
tel
```

This need not be the case: in Lustre, we can have variables of type `integer`, `real`, etc.

Infinite state machines

In the program below, all variables are Boolean, therefore range over a **finite** domain:

```
node Edge (X : bool) returns (E : bool);
let
  E = false -> X and not pre X ;
tel
```

This need not be the case: in Lustre, we can have variables of type `integer`, `real`, etc.

Formally, the tuple $(I, O, S, s_0, \delta, \lambda)$ can model also infinite state machines, by allowing sets I, O, S to be infinite.

Infinite state machines

In the program below, all variables are Boolean, therefore range over a **finite** domain:

```
node Edge (X : bool) returns (E : bool);
let
  E = false -> X and not pre X ;
tel
```

This need not be the case: in Lustre, we can have variables of type `integer`, `real`, etc.

Formally, the tuple $(I, O, S, s_0, \delta, \lambda)$ can model also infinite state machines, by allowing sets I, O, S to be infinite.

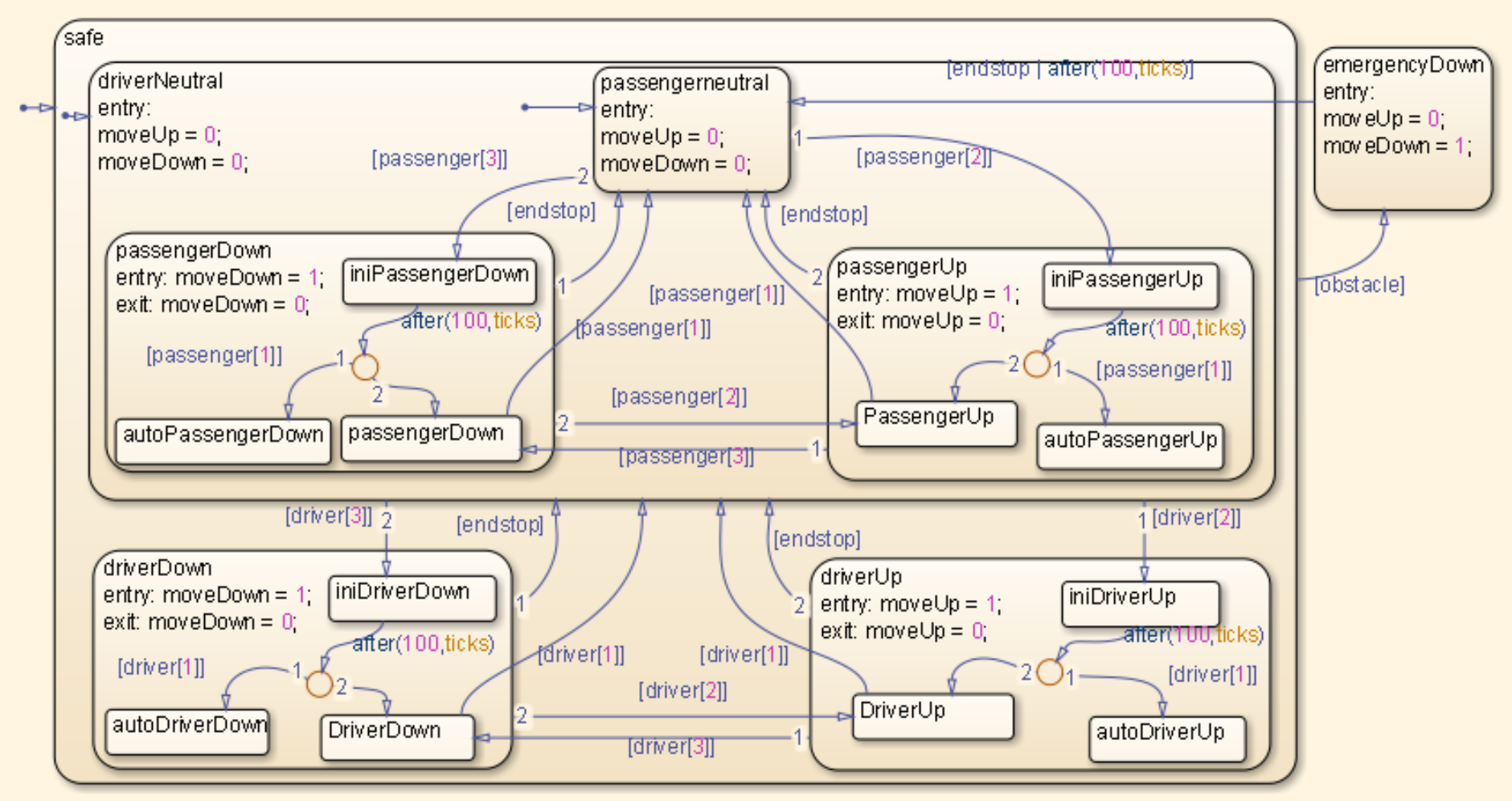
Quiz: write a counter in Lustre.

A counter in Lustre

Extended state machines

- State machines + additional state variables
- State variables can be of infinite type (integers, reals, lists, ...).
- Transitions can have **guards** (conditions on state variables which specify when can a certain transition occur) and **updates** of state variables.
- Such ESMs generally have an infinite number of states $\Rightarrow$ problems such as reachability, termination, ..., are usually **undecidable**.
- Can also have other features, such as message receptions/transmissions, for communication with other machines.
- Can also be **hierarchical**.
- Widely used in the industry: Stateflow, UML/SysML, ...

Example: a hierarchical extended state machine in Stateflow



nuXmv

The model-checker nuXmv

- Widespread symbolic model checker
- Long history, starting from SMV (Symbolic Model Verifier) by Ken McMillan [McMillan, 1993]
- Synchronous systems (mostly)

Modeling FSMs in nuXmv

```
MODULE main
VAR
  b : boolean;

ASSIGN
  init(b) := TRUE;
  next(b) := !b;

INVARSPEC
  b;
```

Modeling FSMs in nuXmv

```
MODULE main
VAR
  i : {0, 1, 2, 3};

ASSIGN
  init(i) := 0;
  next(i) := case
                i<3 : i+1;
                TRUE : 0;
             esac;

INVARSPEC
  i>=0 & i<=3;
```

Modeling FSMs in nuXmv

```
MODULE main
VAR
  bit0 : counter_cell(TRUE);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);
SPEC
  AG AF bit2.carry_out

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := FALSE;
  next(value) := value xor carry_in;
DEFINE
  carry_out := value & carry_in;
```

SMV model counter.smv taken from `http://nusmv.fbk.eu/examples/examples.html`

Bibliography


Baier, C. and Katoen, J.-P. (2008).
Principles of Model Checking.
MIT Press.


Bozzano, M., Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., and Tonetta, S.
nuXmv 1.1.1 User Manual.


Clarke, E., Grumberg, O., and Peled, D. (2000).
Model Checking.
MIT Press.


Hachtel, G. D. and Somenzi, F. (1996).
Logic Synthesis and Verification Algorithms.
Kluwer.


Halbwachs, N., Caspi, P., Raymond, P., and Pilaud, D. (1991).
The synchronous dataflow programming language Lustre.
Proceedings of the IEEE, 79(9):1305–1320.


Kohavi, Z. (1978).
Switching and finite automata theory.
McGraw-Hill, 2 edition.


McMillan, K. (1993).
Symbolic model checking: an approach to the state-explosion problem.
Kluwer.