Reminder: systems

- System: atomic system | composite system
- Atomic system: state + dynamics (+ inputs/outputs)
- Composite system: set of subsystems + composition
- Dynamics: rules defining how state evolves in time
- Composition: rules defining how subsystems interact
Classes of systems/models considered in this course

- Continuous: differential equations, …
- Discrete: state machines, transition systems, …
- Timed: discrete-event, timed automata, …
- Dataflow: process networks, SDF, …
- Probabilistic: Markov chains, …
Discrete systems

Automata, state machines, transition systems, …

• States
• Transitions: discrete moves from one state to the next

• “logical” time = order of transitions
• As opposed to quantitative, “real-time” models such as differential equations or timed automata (we will see those later).

Finite State Machines

Machines of type **Moore** or **Mealy**

Main application: digital circuits
Moore machines

States: \{q_0, q_1, q_2, q_3\}
Initial state: q_0
Input symbols: \{x, y, z\}
Output symbols: \{a, b, c\}
Output function:
\[
\text{out} : \text{States} \to \text{Outputs}
\]
Transition function:
\[
\text{next} : \text{States} \times \text{Inputs} \to \text{States}
\]
Mealy machines

States: {S0, S1, S2}
Initial state: S0
Input symbols: {0,1}
Output symbols: {0,1}
Output function:

\[ \text{out} : \text{States} \times \text{Inputs} \rightarrow \text{Outputs} \]

Transition function:

\[ \text{next} : \text{States} \times \text{Inputs} \rightarrow \text{States} \]
Finite State Machines – Formal Definition

An FSM is a tuple

\[(I, O, S, s_0, \delta, \lambda)\]

- \(I\): set of inputs
- \(O\): set of outputs
- \(S\): set of states
- \(s_0 \in S\): initial state
- \(\delta : S \times I \rightarrow S\): transition function
- \(\lambda\): output function
  - If the FSM is of type **Moore**: 
    \[\lambda : S \rightarrow O\]
  - If the FSM is of type **Mealy**: 
    \[\lambda : S \times I \rightarrow O\]

Example: Mealy Machine

structure:

\[
\begin{align*}
\text{in1} & \in \{0, 1\} \\
\text{in2} & \in \{0, 1\} \\
\text{out} & \in \{0, 1, 2\}
\end{align*}
\]

behavior:
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\(^1\)):

- 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.
- Memory elements often have more inputs (e.g., resets to initialize state).

\(^1\)More accurate description of timing behavior in timing analysis lecture.

Is the D flip-flop a state machine?
Are 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?

From Circuits to State Machines

Is this a state machine?
From Circuits to 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?

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):

![Arbiter diagram]

Better drawing:

![Better arbiter diagram]

Modeling and Implementation/Synthesis

What we have done / what we will do next:

![Modeling and Implementation/Synthesis diagram]
From FSMs to Circuits

“Brute-force” implementation:
- \( \log n \) flip-flops, where \( n = |S| \) = number of states of the FSM.
- \( \log k \) input wires, where \( k = |I| \) = number of input symbols.
- \( \log m \) output wires, where \( m = |O| \) = number of output symbols.
- Multiplexers to implement transition and output functions.

More efficient implementations: the logic synthesis problem. Several subproblems:
- State encoding (or state assignment)
- Logic minimization
- ...

Let’s implement this FSM (on whiteboard):
From FSMs to Circuits

Several combinatorial optimization problems. E.g., state assignment (state encoding): how to encode the states of a given FSM as boolean vectors. Which of the many possible encodings to choose?

Example (taken from [Kohavi, 1978]):

<table>
<thead>
<tr>
<th>Table 12.1 Machine $M_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NS$</td>
</tr>
<tr>
<td>$PS$ $x = 0$ $x = 1$</td>
</tr>
<tr>
<td>$x = 0$ $x = 1$</td>
</tr>
<tr>
<td>$A$ $A$ $D$ 0 1</td>
</tr>
<tr>
<td>$B$ $A$ $C$ 0 0</td>
</tr>
<tr>
<td>$C$ $C$ $B$ 0 0</td>
</tr>
<tr>
<td>$D$ $C$ $A$ 0 1</td>
</tr>
</tbody>
</table>

(a) Assignment $\alpha$

<table>
<thead>
<tr>
<th>$Y_1Y_2$ $x = 0$ $x = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$ $x = 0$ $x = 1$</td>
</tr>
<tr>
<td>$A$ 10 00 10 0 1</td>
</tr>
<tr>
<td>$B$ 11 11 01 0 0</td>
</tr>
<tr>
<td>$C$ 10 11 00 0 1</td>
</tr>
<tr>
<td>$D$ 10 11 00 0 1</td>
</tr>
</tbody>
</table>

(b) Assignment $\beta$

<table>
<thead>
<tr>
<th>$Y_1Y_2$ $x = 0$ $x = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$ $x = 0$ $x = 1$</td>
</tr>
<tr>
<td>$A$ 00 00 11 0 1</td>
</tr>
<tr>
<td>$B$ 01 00 10 0 0</td>
</tr>
<tr>
<td>$C$ 10 10 01 0 0</td>
</tr>
<tr>
<td>$D$ 11 10 00 0 1</td>
</tr>
</tbody>
</table>

From FSMs to Circuits

The two state encodings result in two very different circuits:

Fig. 12.1 First realization of $M_1$.

Fig. 12.2 Second realization of $M_1$.

Figures taken from [Kohavi, 1978].
An elegant notation for (not necessarily finite) state machines: Lustre

A program in the synchronous language Lustre [Halbwachs et al., 1991]:

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

Can you guess its meaning?

\[
E_0 = false \\
E_{k+1} = X_{k+1} \land \neg X_k
\]

Quiz: write a counter in Lustre.

Bibliography