Final State Machine (Automata)

Acceptor / Recognizer:

For \((E, S, S_0, \delta, F)\):

- \(E\): input alphabet
- \(S\): set of states
- \(S_0\): initial state
- \(\delta\): state transitions \(\delta : S \times E \rightarrow S\)
- \(F\): set of final states

Acceptor / Recognizer outputs yes or no if an accepting state \((F)\) is reached at end of input \((E)\).

Example: Detector for input pattern \(a b c\)

- \(E = \{a, b, c\}\)
- \(S = \{s_1, s_2, s_3, s_{no}, s_{yes}\}\)
- \(S_0 = s_1\)
- \(F = s_{yes}\)

State transitions:

- \(\delta(s_1, a) = s_2\)
- \(\delta(s_1, b) = s_{no}\)
- \(\delta(s_1, c) = s_2\no\)
- \(\delta(s_2, a) = s_3\)
- \(\delta(s_2, b) = s_{yes}\)
- \(\delta(s_2, c) = s_{no}\)
- \(\delta(s_3, a) = s_{no}\)
- \(\delta(s_3, b) = s_2\)
- \(\delta(s_3, c) = s_{yes}\)
- \(\delta(s_{yes}, a) = s_{no}\)
- \(\delta(s_{yes}, b) = s_{no}\)
- \(\delta(s_{yes}, c) = s_{yes}\)
- Provide a formal definition of our desired system behavior.
- Formalism can be used to evaluate PSM.
- Helps us to build tools that can easily manipulate PSM.

PSM Properties

1. **Only one condition should be true**
   - Verify: AND of every pair of conditions = 0

2. **One condition should be true**
   - OR of all transitions should be 1

   **Prop 1:** \( a + b \neq 0 \)
   **Prop 2:** \( a + b \neq 1 \)

   What if \( ab = 1 \)?

   - Prop 1: \( a + a'b \neq 0 \)
     - \( a + a'b \neq 1 \)
     - Property 1: \( a'b + a = a'b + a = a + a'b = 0 \)
     - Property 2: \( a'b + a + a'b = a + a'b \neq 0 = a + a'b \neq 1 \)

What if \( ab = 0 \)?

   - Prop 1: \( a + a'b = 0 \)
     - Property 1: \( a'b + a = a'b + a = a + a'b = 0 \)
     - Property 2: \( a'b + a + a'b = a + a'b \neq 0 = a + a'b \neq 1 \)
Transducer:

$\text{PSN} \left( \Sigma, \Gamma, S, s_0, \delta, \omega \right)$

$\Sigma$: inputs
$\Gamma$: outputs
$S$: set of states
$s_0$: initial state
$\delta$: state transitions $\delta: S \times \Sigma \rightarrow S$
$\omega$: output functions

Moore: $\omega: S \rightarrow \Gamma$ => output dependent only on state

Mealy: $\omega: S \times \Sigma \rightarrow \Gamma$ => outputs dependent on state and input

Moore:

\[ \begin{array}{c}
\delta(s_0, b) = s_0 \\
\delta(s_0, b') = s_0 \\
\delta(s_1, b) = s_1 \\
\delta(s_1, b') = s_0 \\
x(s_0, b) = 0 \\
x(s_0, b') = 0 \\
x(s_1, b) = 1 \\
x(s_1, b') = 1 \\
\end{array} \]

Mealy:

\[ \begin{array}{c}
\delta(s_0, b) = s_0 \\
\delta(s_0, b') = s_0 \\
\delta(s_1, b) = s_1 \\
\delta(s_1, b') = s_0 \\
x(s_0, b) = 0 \\
x(s_0, b') = 0 \\
x(s_1, b) = 1 \\
x(s_1, b') = 1 \\
\end{array} \]
Register Transfer Level (RTL) Design

- Specifies computation as transfers of data between registers.
- Often modeled using FSMD (Finite State Machine with data).
- FSMD: Extension of FSM modeling concepts to allow for efficient data representation with precise definition.

FSMD:

FSMD(Σ, Γ, R, S, S₀, δ, w)

- Σ: Inputs (Boolean and multi-bit data)
- Γ: Outputs (Boolean and multi-bit data)
- R: Set of internal registers
- S: Set of states
- S₀: Initial state
- δ: State transitions
  \[ δ: S × Σ × R \rightarrow S \]  ⇒ allows for arbitrary logical expressions input and register to define next state

- w: Output and register transfer functions
  \[ w: S × Σ × R \rightarrow \{Γ', R'\} \]  ⇒ allows for use of arbitrary computations to define outputs and registers
Example: Linear distance measure

Input: B, S
Output: L, D
Register: Ostr

\[ S = \{ B, S \} \]
\[ F = \{ L, D \} \]
\[ R = \{ Ostr \} \]
\[ S = \{ S_0, S_1, S_2, S_3, S_4 \} \]
\[ S_0 = \{ S_0 \} \]
\[ f: S(S_0) = S_1 \quad w: Ostr(S_0) = O \]
\[ f(S_1, B) = S_2 \quad Ostr(S_1) = D \]
\[ f(S_2, B) = S_3 \quad Ostr(S_2, Ostr) = Ostr + 1 \]
\[ f(S_3, B) = S_4 \quad Ostr(S_3) = Ostr \]
\[ f(S_4, B) = S_1 \quad Ostr(S_4) = Ostr / 2 \]

Question: What is the value of Ostr in S2 and S4?
What is the value of D in S1, S2, and S3?

* Registers maintain last value stored until the register is updated (i.e., a register transfer is defined for that register)
Example:

\[ \text{State-based Mode Timing Alternatives} \]

1. Input-based timing (finite state acceptor)
2. Event-driven (behavior defined by discrete events)
3. Cycle-accurate (state and registers updated on clock edges)

Cycle-accurate (or even approximate)

Regular transfers and state transitions are updated at same time (both on clock edges)

- Implies that in FSM model, registers have a "current" and "next" value within a given state
  - transitions are based on current value
  - next value will be updated on next clock edge (i.e., next state transition)

Question: What is the behavior of the above example?