Defining Signals and Systems

Last time we
- Found ways to define functions and systems
- Defined many example systems

Today we will
- Introduce a particular system called a finite state machine
- Talk about the parts and behaviors of state machines
- Find ways to define state machines
- Discuss the concept of state
- Concentrate on deterministic state machines
- Revisit last slide from last lecture

Defining Systems: Block Diagrams

Suppose I want the computer to automatically make the digital recording louder if it senses that my voice is too soft.

A feedback signal called Volume could be added to tell the computer to scale the digital samples to reach a specified average volume:

Voice \rightarrow \text{Microphone} \rightarrow \text{MicOutput} \rightarrow \text{Volume} \rightarrow \text{Computer} \rightarrow \text{DigitizedSound} \rightarrow \text{Volume Adjustor}

DigitizedSound cannot be written directly as a function of Voice. An implicit definition like the differential equation can be used to define DigitizedSound in terms of the input Voice and the systems:

$$\text{DigitizedSound} = \text{Computer}(\text{Microphone}(\text{Voice}), \text{Volume Adjustor}(\text{DigitizedSound}))$$
Defining Systems: State Machines

Consider a system which takes a sequence of elements as input, and provides a sequence of elements as output.

To define the system, we must specify the sequence of outputs produced by a given sequence of inputs.

It is often convenient to define such a system using the concept of system state.

The state of a system represents the history of input elements.

A state machine tells us which output element is produced, given an input element & the system state (history of input elements).

State Machines: Operation

Roughly speaking, a state machine operates as follows:

1) An element from the input sequence, x(n), is considered.
2) Based on x(n) and the current state, s(n), the state machine generates an output y(n).
3) Based on x(n) and the current state, s(n), the state machine generates a new state s(n+1). This is called a state transition.
4) Return to step 1), consider x(n+1).

In this way, the state machine specifies a sequence of outputs for a sequence of inputs.
**State Machines: Parts**

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 × Inputs → States × 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:

\[(States, Inputs, Outputs, update, initialState)\]

---

**State Machine: Stuttering**

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

When \(x(n) = absent\):

- the state does not change, so \(s(n+1) = s(n)\);
- the output is also “nothing”, i.e. \(y(n) = absent\).

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

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

\[\text{absent} \in Inputs \quad \text{absent} \in Outputs\]
Defining State Machines

To define a state machine, define all 5 elements of the 5-tuple

\[(\text{States}, \text{Inputs}, \text{Outputs}, \text{update}, \text{initialState})\]

Example: Suppose I am playing a game by tossing a coin.
If I toss 3 heads in a row, I win.
If I toss a tail before tossing three heads, I lose.

\[\text{States} = \{0\_\text{heads}, 1\_\text{head}, 2\_\text{heads}\}\]

\[\text{Inputs} = \]

\[\text{Outputs} = \]

\[\text{initialState} = \]

Defining State Machines: Table

The function \textit{update} can be defined using a table:

\[
\begin{array}{|c|c|c|}
\hline
(s(n+1), y(n)) & = & \text{update}(s(n), x(n)) \\
\hline
x(n) = \text{head} & x(n) = \text{tail} \\
\hline
s(n) = 0\_\text{heads} & & \\
\hline
s(n) = 1\_\text{head} & & \\
\hline
s(n) = 2\_\text{heads} & & \\
\hline
\end{array}
\]

There is a more visually appealing way to define the \textit{update} function: a state diagram.
**Defining State Machines: State Diagram**

To create a state diagram for a state machine, first draw circles representing the states.

For each combination of input and state, draw an arrow from the current state to the next state. Label the arrow with the input and output that create the transition as shown: “input/output”

**State Machines: State Response**

The state response is sequence of states resulting from a particular input sequence.

Example: Find the state response and output sequence for

\[ x = \text{tail} \text{ head} \text{ head} \text{ tail} \]
**Facts About State Machines**

- 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).
- 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 transition will be defined for every possible combination of input and current state.
- If a transition is not shown for a particular input, assume the transition is to the same state and the output is absent.

---

Consider the transition labels, “input/output”. If more than one input leads to the same transition and output, the label may contain a set of inputs.

- In the transition label “input/output”, the set of inputs is called a guard. The input must match the guard for the transition to take place.
- For the state machines studied in this lecture, there is exactly one possible transition for each combination of current state and input. These state machines are called deterministic.
- Sometimes it is useful to model a system using a state machine that has more than one possible transition for each combination of current state and input. These state machines are called non-deterministic.