# Finite State Automata (Finite State Machines)

Finite State Automata are graphs in which
- the vertices represent the states of an object (or system)
- one state is designated the "start state"
- zero or more states are designated "final states"
- each edge is directed from one state to another
- each edge may be labeled:
    * Typically with an event/reaction pair.
    * Sometimes in other ways.

Example: Asynchronous System
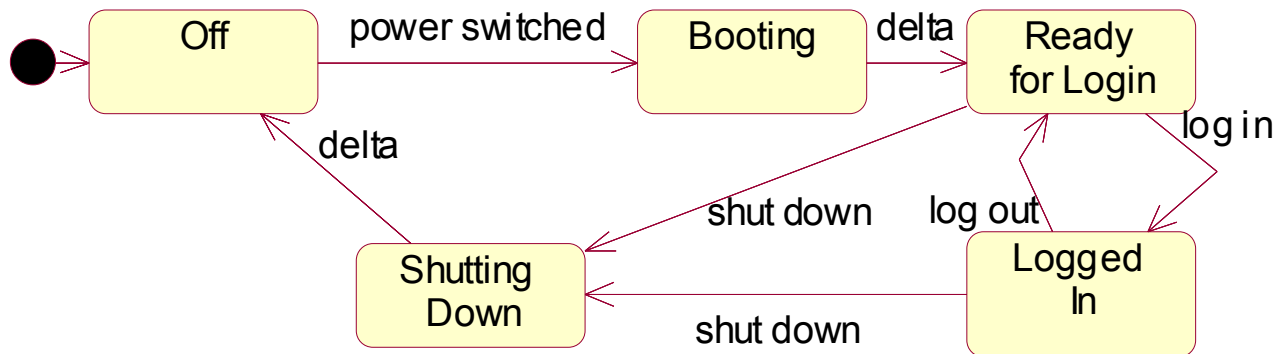
Here

- The start state is named "Off"

- There are no final states

- Only events appear

- The delta event represents the passage of some amount of time

- The above is an "asynchronous" system
  - ∗ Events may happen at variable intervals.
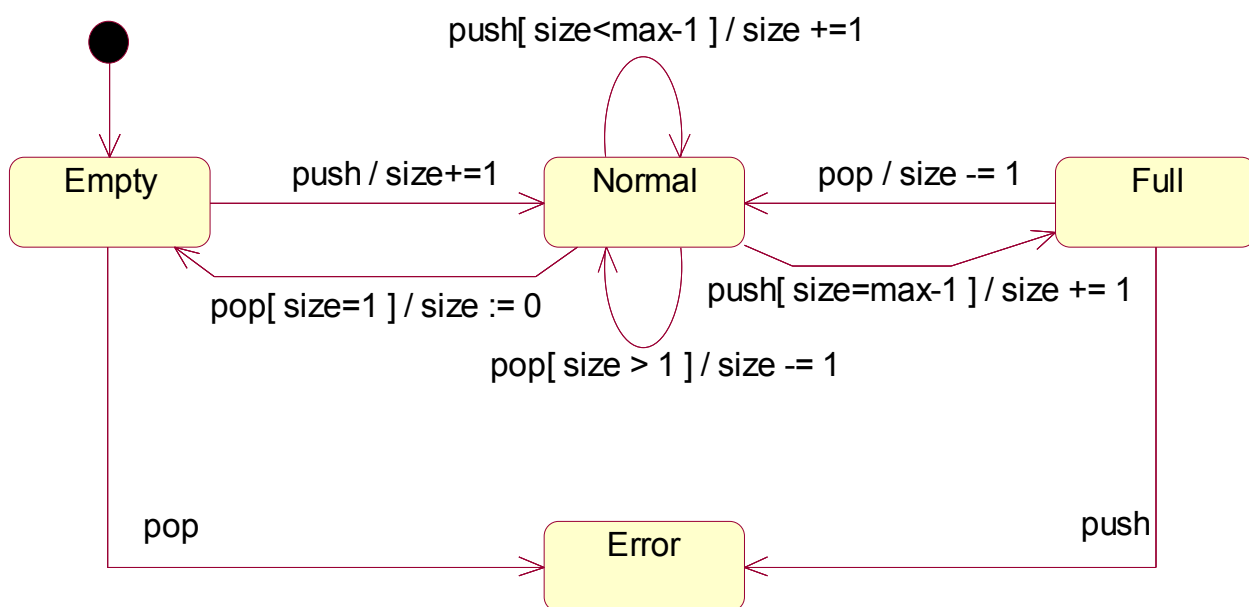
2

Example: Synchronous System

As you know, state machines can model synchronous digital hardware

 ● events are the inputs at each clock

 ● reactions are the outputs at each clock
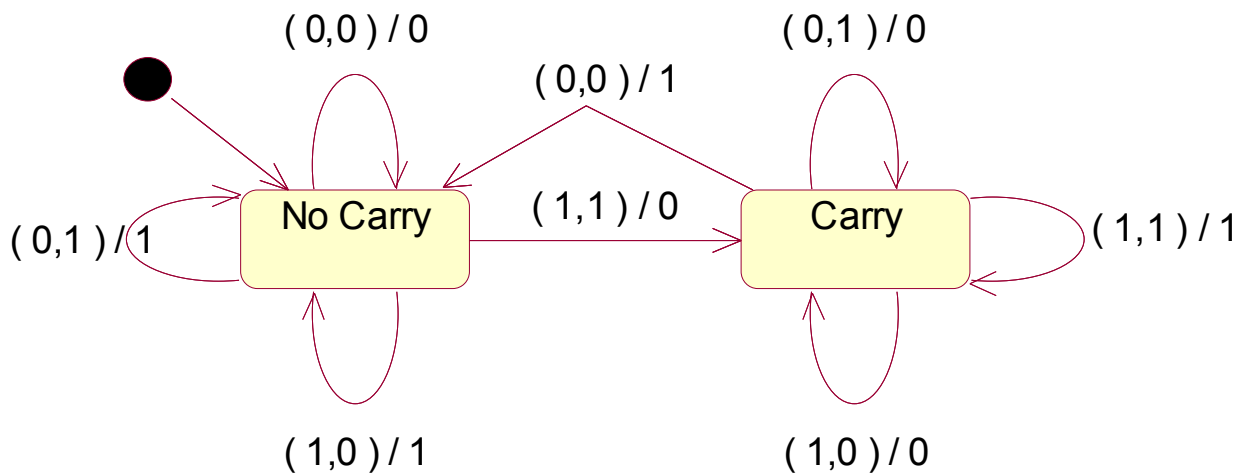
A serial adder.

Example: Software objects

A finite state model of a *Stack* class.

- vertices represent some aspects of an object's state.

- events are calls to the object's public functions.

- In [square brackets] are conditions restricting transitions.

- reactions modify aspects of the object's state not captured by vertices, specify new events, specify return values.
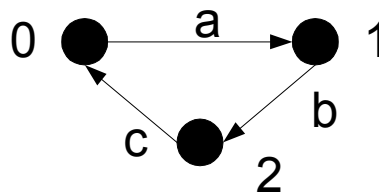
4

Formalizing Finite State Automata

Directed Graphs (Reminder)

Defn: A *directed graph*, $D = (V, E, \phi)$, consists of

- A set of vertices V

- A set of edges E

- A function $\phi : E \to (V \times V)$

Example: $V = \{0, 1, 2\}$ $E = \{a, b, c\}$ and $\phi(a) = (0, 1)$, $\phi(b) = (1, 2)$, $\phi(c) = (2, 0)$

Defn: A *walk* in a directed graph, $D = (V, E, \phi)$, is an alternating sequence of vertices and edges
$$(v_0, e_1, v_1, e_2, ..., e_n, v_n)$$
where $\phi(e_i) = (v_{i-1}, v_i)$, for $i \in \{1, 2, ..., n\}$.

Finite State Automata

Defn: A *finite state automaton (FSA)*, $A = (V, E, \phi, s, F, \Sigma, \Pi, \lambda)$ consists of

- (V, E, ϕ) is a finite directed graph

- $s \in V$ is a *start state*

- $F \subseteq V$ is a set of *final states*

- Σ is a set of *input symbols* (events)

- Π is a set of *output symbols* (reactions)

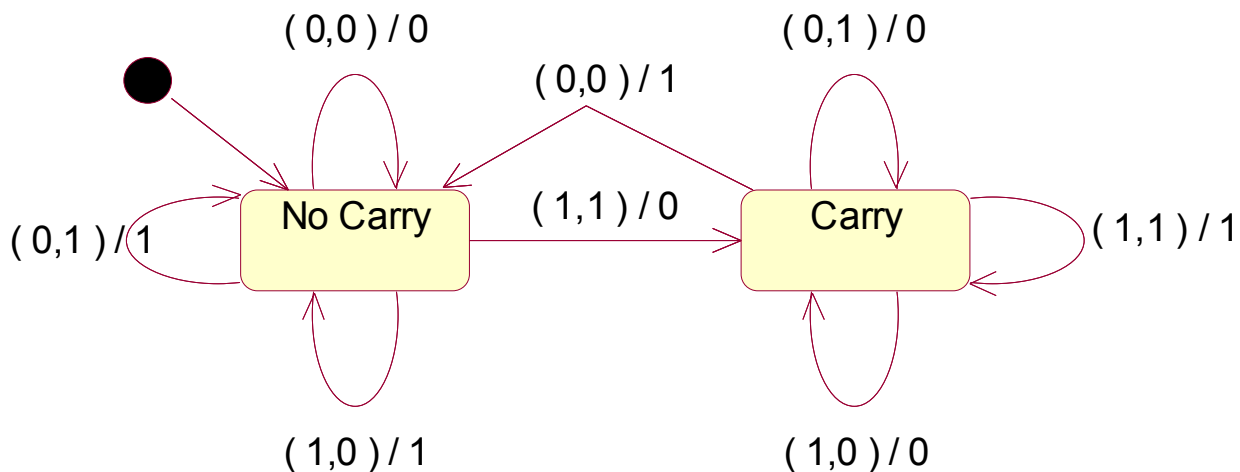- $\lambda : E \to (\Sigma \times \Pi)$ is a *labelling function*

Each walk
$$(v_0, e_1, v_1, e_2, ..., e_n, v_n)$$
in the graph $D = (V, E, \phi)$ where $v_0 = s$ gives rise to a *behaviour* in the automaton
$$(\sigma_1, \pi_1, \sigma_2, \pi_2, ..., \sigma_n, \pi_n)$$
where $\lambda(e_i) = (\sigma_i, \pi_i)$, for each $i \in \{1, 2, ..., n\}$.

6

Example: The serial adder again.

(0,0) / 0   (0,1) / 0

(0,0) / 1

No Carry   (1,1) / 0   Carry   (1,1) / 1

(0,1) / 1

(1,0) / 1   (1,0) / 0

For the serial adder, one behaviour is

$$((1,1), 0, (1,0), 0, (0,0), 1)$$

This corresponds to

$$\begin{array}{r} 11 \\ +\ 01 \\ \hline 100 \end{array}$$

Defining Languages

Sometimes we want to define some behaviours as complete and others as incomplete.

For example, suppose we want to define the set of valid C++ floating point constants.

Rules for floating point constants

- Basic form is ±DDD.DDDe±DDD

- Signs are optional in both mantissa and exponent

- Exponent part is optional.

- Decimal point is optional.

- Must have either exponent part or decimal point.

- At least one digit must appear before or after the decimal point

- At least one digit must appear before the exponent part.

- At least one digit must appear in the exponent part if any

The 3 final states are marked with arrows to double circles

We use only input symbols, not output symbols. Symbol D stands for any digit.
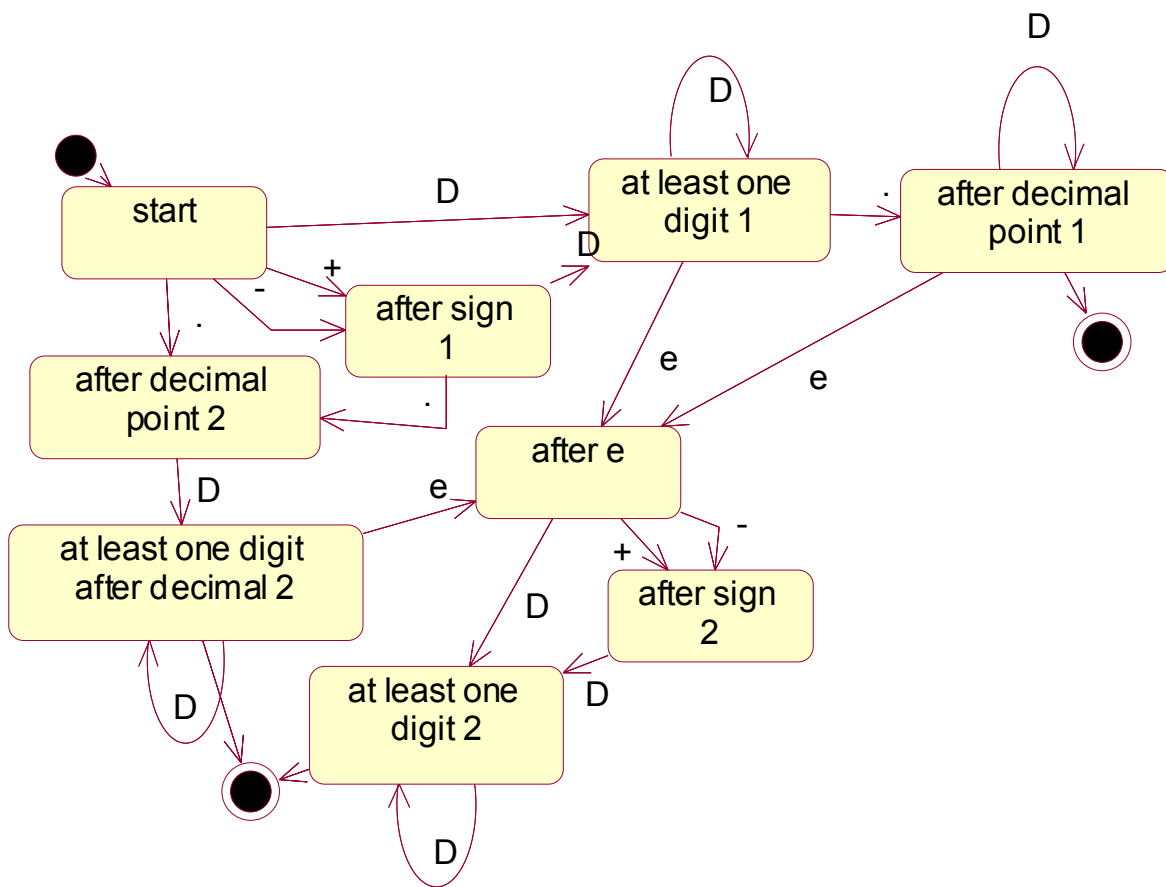
A *complete behaviour* is one that arises from a walk that starts at the start state and ends at a final state.

Thus

$$DD.DDe + DD$$

is a complete behaviour, and

$$DD.DD$$

is a complete behaviour, but

$$DD.DDe+$$

is not.

Furthermore

$$DD.DD.DD \qquad, + DD. + DD \qquad, \text{ and } \qquad eDD$$

are not behaviours at all.

A *language* is a set of finite sequences.

The *language* of an FSA is its set of complete behaviours.

Not all languages can be defined by some FSA.

An interesting problem. Given an automaton describing a language, what automaton describing the same language has the fewest states.

Because FSAs are well defined mathematically and are finite, they are well suited to modelling systems and to (automated) proving of properties of systems.