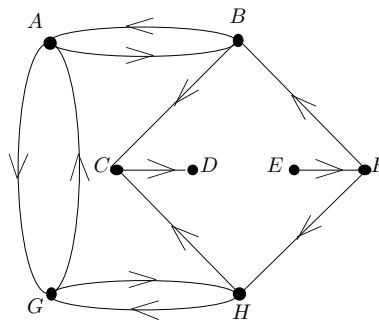


Definition A **directed graph** is a graph in which the edges have been given an orientation to allow travel along a given edge only in the specified direction. The edges of a directed graph are also called **arcs**.

Example 1



In this graph some possible paths are $G A G H C D$, $E F H G A B C$, $F B C D$, for example. But there is no path which ends at E and there is no path which begins at D .

The internal memory of a computer is a system which can exist in only a finite number of different states. (There are many other “systems”, both electrical and mechanical which have this property.) Such systems are known as Finite State Machines (FSMs) or Automata.

Such a system typically accepts inputs of some type and, as a result of the input, enters into a new (possibly unchanged) state. The new state normally depends on two things:

- (a) the input, and
- (b) the state of the system prior to receiving the input.

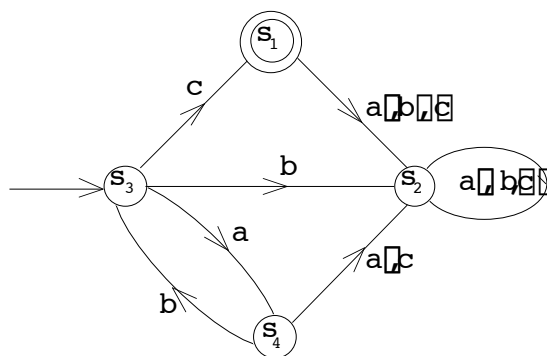
Such a situation can be modelled using directed graphs.

Definition A **finite state machine** (FSM) consists of

- (1) a finite set S , consisting of the **states** of the machine,
- (2) a particular state $s_0 \in S$, called the **initial state** of the machine,
- (3) a subset $F \subseteq S$, called the **final states** (or **accepting states**) of the machine,
- (4) a finite set Σ consisting of the inputs for the machine (Σ is called the **alphabet** for the machine),
- (5) a function $f: S \times \Sigma \rightarrow S$, called the **transition function**.

Example 2

Consider the following state digram of an FSM.



Here we have the state set $S = \{s_1, s_2, s_3, s_4\}$. The initial state is s_3 . The only accepting state is s_1 . The input alphabet is $\Sigma = \{a, b, c\}$. Note that rather than putting three loops labelled a , b and c at s_2 we put a single loop and label it a, b, c .

The transition, or state table for this machine is

state table	<i>a</i>	<i>b</i>	<i>c</i>
<i>s</i> ₁	<i>s</i> ₂	<i>s</i> ₂	<i>s</i> ₂
<i>s</i> ₂	<i>s</i> ₂	<i>s</i> ₂	<i>s</i> ₂
<i>s</i> ₃	<i>s</i> ₄	<i>s</i> ₂	<i>s</i> ₁
<i>s</i> ₄	<i>s</i> ₂	<i>s</i> ₃	<i>s</i> ₂

What happens if we input the “word” *abcac*? The FSM starts in state *s*₃ and then goes through the following sequence of states:

$$s_3 \xrightarrow{a} s_4 \xrightarrow{b} s_3 \xrightarrow{c} s_1 \xrightarrow{a} s_2 \xrightarrow{c} s_2.$$

Since the machine ends up in the non-accepting state *s*₂, the word *abcac* is not accepted by the machine. Note that the state *s*₂ is a **black hole (or dump state)** since once the machine enters this state it can never again leave it.

What happens if we input the “word” *ababc*? The FSM starts in state *s*₃ and then goes through the following sequence of states:

$$s_3 \xrightarrow{a} s_4 \xrightarrow{b} s_3 \xrightarrow{a} s_4 \xrightarrow{b} s_3 \xrightarrow{c} s_1.$$

Since the machine ends up in the accepting state *s*₁, the word *ababc* is accepted by the machine.

After a bit of experimentation it seems that the words accepted by this machine are precisely those of the form $(ab)^n c$ for some $n \geq 0$.

Definition Consider a finite state machine.

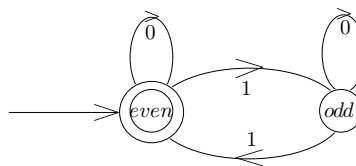
- (1) A finite sequence of elements from the alphabet Σ is called a **word** or **string** over Σ .
- (2) The **empty word** over Σ is denoted by θ and is the word with no symbols.
- (3) The set of all words over Σ is denoted Σ^* and any subset of Σ^* is called a **language** over Σ .

Thus a FSM reads a word and having read the word ends up in a certain state. If that state is a final state the word is **accepted** by the machine. If not, we say that the word is **rejected**.

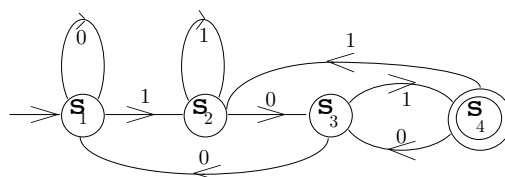
With this in mind, two natural questions arise.

- (a) Given a FSM, what is the set of words accepted by the machine? (This set of words is called the **language** of the machine.)
- (b) How can one design a FSM to accept a given language? (It should be noted that there are certain languages which will be accepted by no finite state machine.)

Example 3 The parity check machine. This machine determines whether or not a binary word contains an even number of “1”s. It accepts the word if there is an even number of “1”s and rejects the word otherwise.



Example 4 We wish to construct a FSM which accepts a binary word if and only if it ends in 101. (That is, a FSM whose language is the set of binary words ending in 101.)



Definition The **extended transition function** is the function

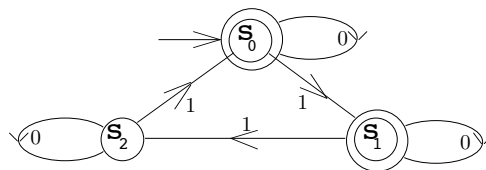
$$f: S \times \Sigma^* \rightarrow S \text{ defined by}$$

- (a) $f(s, \theta) = s$ (recall θ is the empty word), and
 (b) if W is a word ending with the symbol i (i.e. W has the form $W = W'i$ where W' is also a word), then $f(s, W) = f(s, W'i) = f(f(s, W'), i)$.

This is a recursive definition. For the machine of Example 4, we have (reading the word 01101).

$$\begin{aligned} f(s_1, 01101) &= f(f(s_1, 0110), 1) \\ &= f(f(f(s_1, 011), 0), 1) \\ &= f(f(f(f(s_1, 01), 1), 0), 1) \\ &= f(f(f(f(f(s_1, 0), 1), 1), 0), 1) \\ &= f(f(f(f(s_1, 1), 1), 0), 1) \\ &= f(f(f(s_2, 1), 0), 1) \\ &= f(f(s_2, 0), 1) \\ &= f(s_3, 1) \\ &= s_4 \end{aligned}$$

Example 5 To determine the language accepted by



Note that on reading a zero the machine does not change state so the language depends on the number of "1"s in the word.

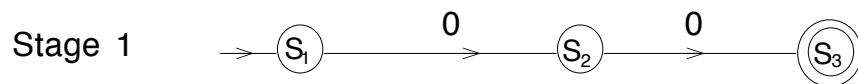
Whenever this machine reads three "1"s (regardless of any "0"s between the "1"s), it will cycle through the three states and return to the state it was in before it read the three "1"s. Since our initial state is an acceptor state, the machine will therefore accept any binary word containing 0, 3, 6, 9, 12, ... "1"s (regardless of the number of zeros).

Since s_1 is also an acceptor, the machine will also accept any word containing 1, 4, 7, 10, ... "1"s (regardless of the number of zeros).

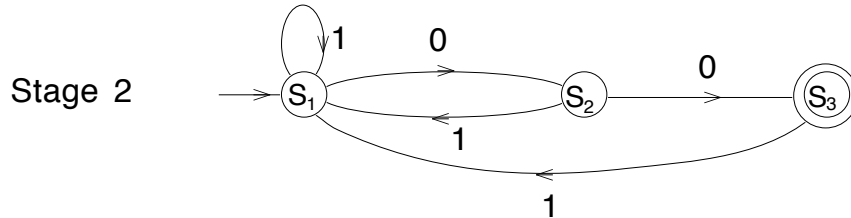
Any word containing 2, 5, 8, 11, ... "1"s will leave the machine in state s_2 and so the machine will reject the word.

Thus the language of the machine is any word where the number of "1"s when divided by 3 gives a remainder of 0 or 1.

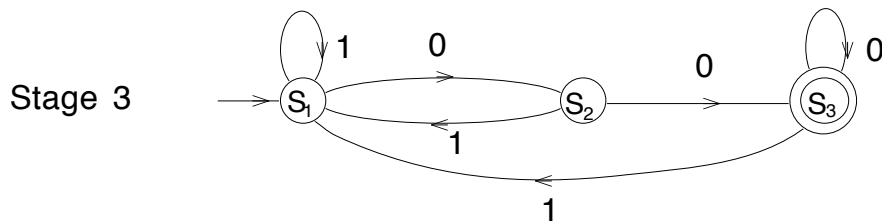
Example 6 To create a recognition machine which accepts words which end with at least two 0's
 The most basic word which will be accepted is 00



Reading a 1 at any stage should lead back to the initial state.

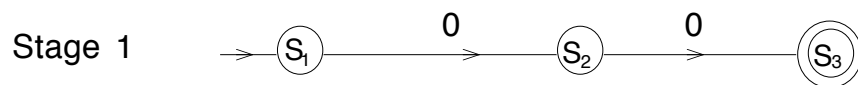


Finally, reading a 0 when at the acceptor state the state should not change.

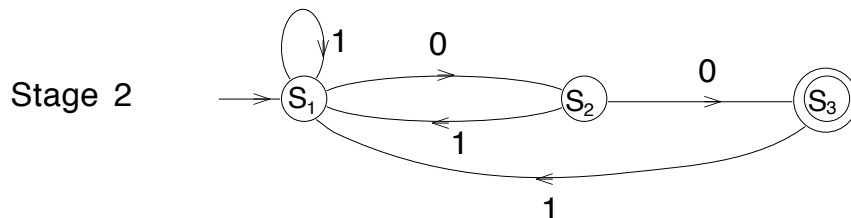


Example 7 To create a recognition machine which accepts words which end with at least two 0's, and not more than two 0's

Stage 1 is the same as above.



Stage 2 is the same as above.



Now, however, reading 0 at the acceptor state, (means that 000 has been read) leads to a new "waiting state" and at S_4 , any more 0's result in no change, but a 1 read at S_4 leads back to the initial state.

