Statecharts preamble

The “finite state model” has been used in many contexts, both conceptual and technical. In it’s original conception, it arose in the mid-1950s to model simple machines having “memory” — that is, responding differently to the same stimulus (or input) at different times, but in a consistent, history sensitive manner. The intuitive idea is that a stimulus may alter the internal configuration of a machine so that if it is presented with the same input again, it responds differently. This is in contrast to “stateless” devices (e.g., an or-gate) that provides a unique result for each stimulus.

Statecharts are an ingenious and substantial extension of this basic model. To establish a common starting point, a few examples of traditional “sequential machines” (as they are usually called) are presented here.

Example I.

This example illustrates a sequential machine with input and output alphabet {a, b} and which copies its input to the output while deleting the second occurrence (if any) of the letter 'a'. As with acceptors we may use either a tabular or a diagrammatic presentation of specific sequential machines. In the state diagram, we need each edge label to indicate both the input and the output — we write this with the notation λ/x, where $\lambda \in \Sigma$ is the input and $x \in \Delta^*$ is the output. Usually intuition is fostered by the state diagram presentation as illustrated in Figure IA, and this is normally all we will subsequently provide.

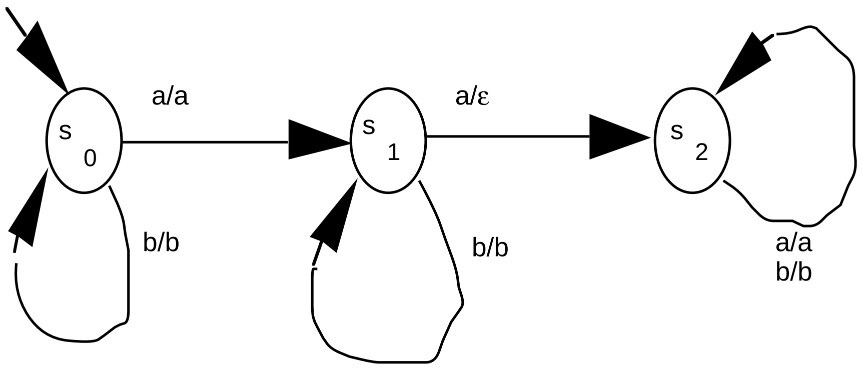

Figure IA.

This machine copies 'b's in state s_0 until the first 'a' occurs, then copies it and transfers to state s_1. Then in state s_1, the machine continues to copy 'b's until the second 'a' occurs. This 'a' places nothing in the output sequence and transfers to state s_2. In s_2 the machine simply copies both 'a's and 'b's. So for instance, this machine for the input 'abbab' would yield output 'abbb' via the run shown in Figure II.

state	s_0	s_1	s_1	s_1	s_2	s_2
input	a	b	b	a	b	
output	a	b	b	ε	b	

Figure IB.

Note the need for permitting an output of length 0 in defining this sequential machine.

Example II.

In this example we have $\Sigma = \Delta = \{0, 1, \#\}$ and we provide a sequential machine that will “compute odd parity” — that is, it will transform an input of the form x# where $x \in \{0, 1\}^*$ into the output $x\pi\#$ where $\pi \in \{0, 1\}$ and $x\pi$ has odd parity (i.e., an odd number of '1's).

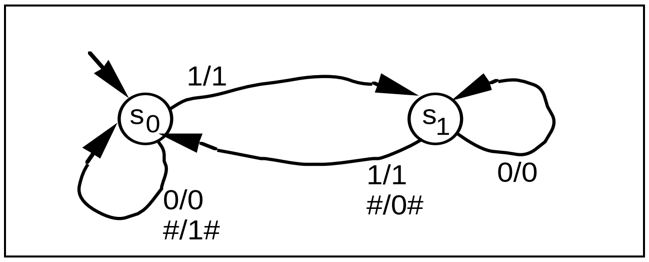

Figure II.

This machine is in state s_0 whenever the input has even parity, and state s_1 whenever the input has odd parity. The output when the “end marker” '#' is encountered then adds '1' in state s_0 and adds '0' when in state s_1 resulting in an odd parity output for all inputs Note the need in this machine for permitting an output of length greater than one for some input letters.

Example III.

We illustrate a sequential machine realization of the unit delay function.

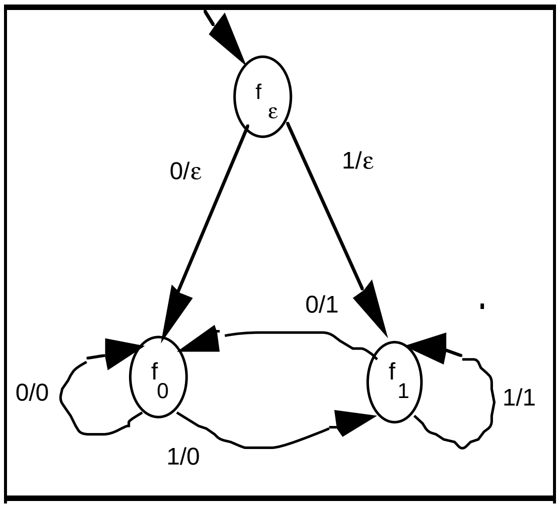

Figure III.

For this machine we see the following input/output correspondence

state	f_ε	f_1	f_0	f_0	f_1	f_0
input	1	0	0	1	0	1
output		1	0	0	1	0

Example IV.

To more clearly illustrate the challenge of understanding sequential machines, consider the sequential machine depicted in Figure IV.

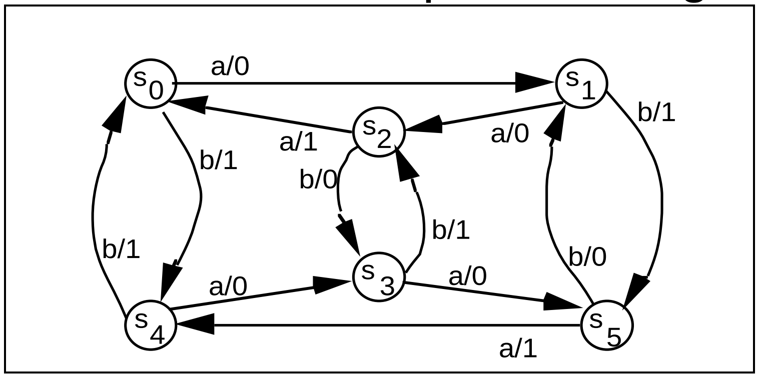

Figure IV.

It can be seen by direct inspection that states s_2 and s_5 each provides an output of '1' for an 'a' input and an output of '0' for a 'b' input. But state s_0 differs from s_2 (and s_5) since its output is '1' for input 'b'. Since there are finitely many combinations of outputs possible for the input letters, it is a matter of simple inspection to determine the equivalence behavior for any input of length one. It is much more challenging to determine equivalent behavior for *all* inputs.

It can be seen that s_2 and s_5 agree not only for length one inputs, they are *equivalent*. However this is not so easy to verify.

Example V.

We observed in Example IV that certain states are equivalent. In fact, there is a reduced machine for this sequential machine obtained by identifying classes of equivalent states, and associating the input/output behavior of the states in a class to the entire class object. If you review the state diagram in Figure IV and in your mind's eye coalesce the states as indicated, you will see exactly the machine depicted in Figure V.

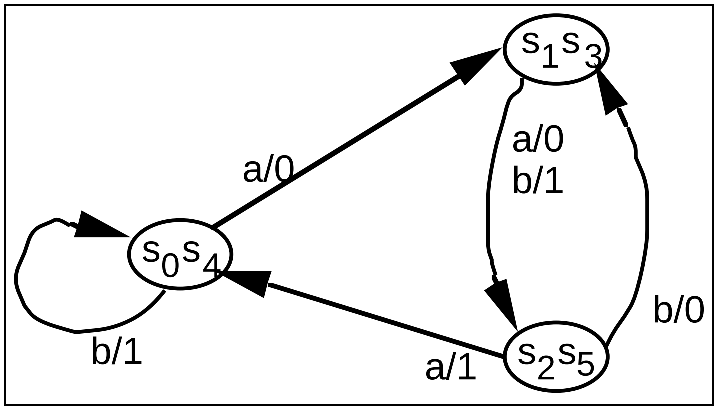

Figure V.