Notes on Finite State Automata

Scribe: Colton Hart   Lecturer/Editor: Chris Eppolito

29 April 2020

Recall that a *finite state machine* is a sextuple $M = (S, I, O, t, w, s_0)$ with...

1. a finite set S of *states* with a distinguished *initial state* $s_0 \in S$,
2. finite sets I and O of *input characters* and *output characters* respectively,
3. a *transition function* $t\colon S \times I \to S$,
4. a *write function* $w\colon S \times I \to O$.

Machines allows both reading and writing. Our next model can answer inclusion questions about nice sets of strings; rather than writing as we read, this model will allow us to output a single boolean value (i.e. $True$ or $False$) after applying the machine to an input string. First we recall some terminology.

Definition. A *language* is a set of *strings* or *words* in some finite *alphabet*.

Example 1. Consider the alphabet $A = \{0, 1\}$. The *bit strings* are the language $L = A^*$.

In the alphabet A of lowercase English letters, both *abracadabra* and *xkcd* are words in A^*. They are not words in the language $L = \{x : x \text{ is an English word}\}$.

For $A = \{0\} \cup [9]$, the set $L = \{x : x \text{ is a weakly increasing word in } A\}$ is a language in A.

The above example demonstrates that our languages can have a syntax, but not necessarily a semantics. We now describe a theory of computation for recognizing strings of *regular languages* (a special class).

Definition. A *(deterministic) finite state automaton* is a quintuple $M = (S, s_0, F, A, t)$ with...

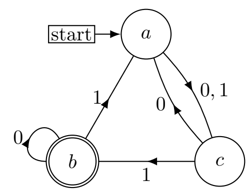
1. a finite set S of *states* with a distinguished *initial state* s_0 and a set $F \subseteq S$ of *final* or *accepting* states,
2. an *input alphabet* A,
3. a *transition function* $t\colon S \times A \to S$.

We often always abbreviate this as "automaton" (or pluralized as "automata").

Remark. As before we draw diagraph pictures of our automata. Our conventions are the same as for state machines except each arrow has one label, and the final states are double circled; to make our diagrams more concise, we often give an arrow more than one label rather than produce multiple arrows.

Example 2. The following picture describes the same automation $M = (S, s_0, F, A, t)$ where $S = a, b, c$, $s_0 = a$, $A = \{0, 1\}$, $F = \{b\}$, and transition function t (given as a table below).

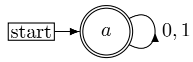
t	0	1
a	c	c
b	b	a
c	a	b

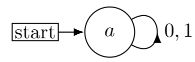


Definition. An automation $M = (S, s_0, F, A, t)$ yields a function $f_M\colon A^* \to \{True, False\}$, where $f_M(w)$ is $True$ if and only if reading w into M ends at a final state. A word w is *accepted* by M when $f_M(w) = True$.

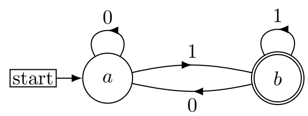
Thus every automaton M determines an *accepted language* $L_M = \{w \in A_M^* : f_M(w) = True\}$. Below we exhibit many examples of automata and the languages they accept.

Example 3. The automata below accept all bit strings and the empty language (respectively).

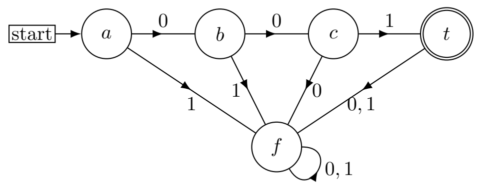




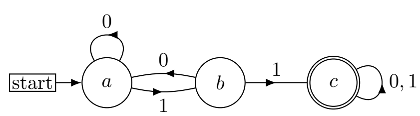
Example 4. The automaton below has accepted language $L = \{w \in \{0,1\}^* : w \text{ ends with a } 1\}$.



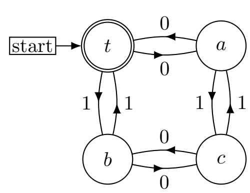
Example 5. We give an automaton which accepts only the bit string 001 below.



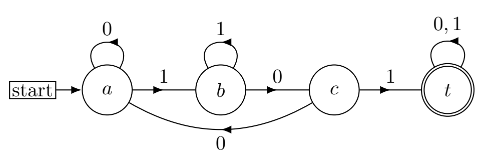
Example 6. We give an automaton that accepts any bit string with 11 as a substring below.



Example 7. The automaton below has $L_M = \{w \in \{0,1\}^* : w \text{ has an even number of each symbol } 0, 1\}$.



Example 8. The following automaton accepts bit strings with a substring 101.



Given automata M and N with $A_M = A_N$, we can construct an automaton which accepts $L_M \cap L_N$.

Proposition. *Given automata M and N with $A = A_M = A_N$, there is an automaton $M \times N$ satisfying $L_{M\times N} = L_M \cap L_N$. In particular, $M \times N$ is given by*

$$M \times N = (S_M \times S_N, (s_0(M), s_0(N)), F_M \times F_N, A, t_{M\times N}),$$

where $t_{M\times N}((m,n),a) = (t_M(m,a), t_N(n,a))$ for all $(m,n,a) \in S_M \times S_N \times A$.

Proof. Notice that $M \times N$ simulates M in its first component and N in its second component. Thus $w \in L_{M\times N}$ if and only if $f_M(w) = True = f_N(w)$. Hence $L_{M\times N} = L_M \cap L_N$ as desired. $\square$