3.7 Finite State Automata With Output: Transducers

So far, we have only considered automata that recognize languages, i.e., automata that do not produce any output on any input (except "accept" or "reject").

It is interesting and useful to consider input/output finite state machines. Such automata are called *transducers*. They compute functions or relations. First, we define a deterministic kind of transducer.

Definition 3.12. A *general sequential machine (gsm)* is a sextuple $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where

(1) Q is a finite set of *states*,

(2) Σ is a finite *input alphabet*,

(3) Δ is a finite *output alphabet*,

(4) $\delta\colon Q \times \Sigma \to Q$ is the *transition function*,

(5) $\lambda\colon Q \times \Sigma \to \Delta^*$ is the *output function* and

(6) q_0 is the *initial* (or *start*) *state*.

If $\lambda(p, a) \neq \epsilon$, for all $p \in Q$ and all $a \in \Sigma$, then M is *nonerasing*. If $\lambda(p, a) \in \Delta$ for all $p \in Q$ and all $a \in \Sigma$, we say that M is a *complete sequential machine (csm)*.

An example of a gsm for which $\Sigma = \{a, b\}$ and $\Delta = \{0, 1, 2\}$ is shown in Figure 3.11. For example aab is converted to 102001.

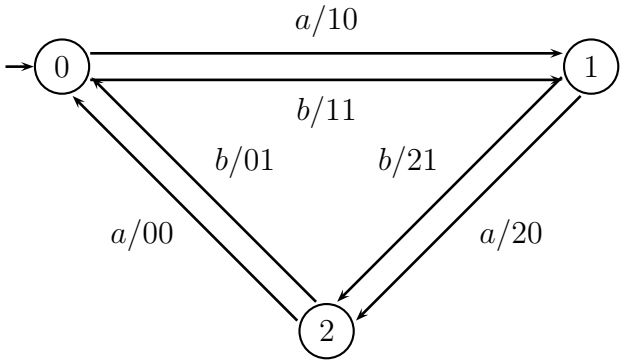

Figure 3.11: Example of a gsm

In order to define how a gsm works, we extend the transition and the output functions. We define $\delta^*\colon Q\times\Sigma^* \to Q$ and $\lambda^*\colon Q \times \Sigma^* \to \Delta^*$ recursively as follows: For all $p \in Q$, all $u \in \Sigma^*$ and all $a \in \Sigma$

$$\begin{aligned}\delta^*(p,\epsilon) &= p\\ \delta^*(p,ua) &= \delta(\delta^*(p,u),a)\\ \lambda^*(p,\epsilon) &= \epsilon\\ \lambda^*(p,ua) &= \lambda^*(p,u)\lambda(\delta^*(p,u),a).\end{aligned}$$

For any $w \in \Sigma^*$, we let

$$M(w) = \lambda^*(q_0,w)$$

and for any $L \subseteq \Sigma^*$ and $L' \subseteq \Delta^*$, let

$$M(L) = \{\lambda^*(q_0,w) \mid w \in L\}$$

and

$$M^{-1}(L') = \{w \in \Sigma^* \mid \lambda^*(q_0,w) \in L'\}.$$

Note that if M is a csm, then $|M(w)| = |w|$ for all $w \in \Sigma^*$. Also, a homomorphism is a special kind of gsm—it can be realized by a gsm with one state.

We can use gsm's and csm's to compute certain kinds of functions.

Definition 3.13. A function $f\colon \Sigma^* \to \Delta^*$ is a *gsm* (resp. *csm*) *mapping* iff there is a gsm (resp. csm) M so that $M(w) = f(w)$, for all $w \in \Sigma^*$.

Remark: Ginsburg and Rose (1966) characterized gsm mappings as follows:

A function $f\colon \Sigma^* \to \Delta^*$ is a gsm mapping iff

(a) f preserves prefixes, i.e., $f(x)$ is a prefix of $f(xy)$;

(b) There is an integer, m, such that for all $w \in \Sigma^*$ and all $a \in \Sigma$, we have $|f(wa)| - |f(w)| \leq m$;

(c) $f(\epsilon) = \epsilon$;

(d) For every regular language, $R \subseteq \Delta^*$, the language $f^{-1}(R) = \{w \in \Sigma^* \mid f(w) \in R\}$ is regular.

A function $f\colon \Sigma^* \to \Delta^*$ is a csm mapping iff f satisfies (a) and (d), and for all $w \in \Sigma^*$, $|f(w)| = |w|$.

The following proposition is left as a homework problem.

Proposition 3.2. *The family of regular languages (over an alphabet Σ) is closed under both gsm and inverse gsm mappings.*

We can generalize the gsm model so that

(1) the device is nondeterministic,

(2) the device has a set of accepting states,

(3) transitions are allowed to occur without new input being processed,

(4) transitions are defined for input strings instead of individual letters.

Here is the definition of such a model, the *a-transducer*. A much more powerful model of transducer will be investigated later: the *Turing machine*.

Definition 3.14. An *a-transducer* (or *nondeterministic sequential transducer with accepting states*) is a sextuple $M = (K, \Sigma, \Delta, \lambda, q_0, F)$, where

(1) K is a finite set of *states*,

(2) Σ is a finite *input alphabet*,

(3) Δ is a finite *output alphabet*,

(4) $q_0 \in K$ is the *start* (or *initial*) *state*,

(5) $F \subseteq K$ is the set of *accepting* (of *final*) *states* and

(6) $\lambda \subseteq K \times \Sigma^* \times \Delta^* \times K$ is a finite set of quadruples called the *transition function* of M.

If $\lambda \subseteq K \times \Sigma^* \times \Delta^+ \times K$, then M is *ϵ-free*

Clearly, a gsm is a special kind of a-transducer.

An a-transducer defines a binary relation between Σ^* and Δ^*, or equivalently, a function $M : \Sigma^* \to 2^{\Delta^*}$.

We can explain what this function is by describing how an a-transducer makes a sequence of moves from configurations to configurations.

The current *configuration* of an a-transducer is described by a triple

$$(p, u, v) \in K \times \Sigma^* \times \Delta^*,$$

where p is the current state, u is the remaining input, and v is some ouput produced so far.

We define the binary relation $\vdash_M$ on $K \times \Sigma^* \times \Delta^*$ as follows: For all $p, q \in K$, $u, \alpha \in \Sigma^*$, $\beta, v \in \Delta^*$, if $(p, u, v, q) \in \lambda$, then

$$(p,\ u\alpha,\ \beta) \vdash_M (q,\ \alpha,\ \beta v).$$

Let $\vdash_M^*$ be the transitive and reflexive closure of $\vdash_M$.

The function $M\colon \Sigma^* \to 2^{\Delta^*}$ is defined such that for every $w \in \Sigma^*$,

$$M(w) = \{y \in \Delta^* \mid (q_0,\, w,\, \epsilon) \vdash_M^* (f,\, \epsilon,\, y),\ f \in F\}.$$

For any language $L \subseteq \Sigma^*$ let

$$M(L) = \bigcup_{w \in L} M(w).$$

For any $y \in \Delta^*$, let

$$M^{-1}(y) = \{w \in \Sigma^* \mid y \in M(w)\}$$

and for any language $L' \subseteq \Delta^*$, let

$$M^{-1}(L') = \bigcup_{y \in L'} M^{-1}(y).$$

Remark: Notice that if $w \in M^{-1}(L')$, then there exists some $y \in L'$ such that $w \in M^{-1}(y)$, i.e.,
$y \in M(w)$. This **does not** imply that $M(w) \subseteq L'$, only that $M(w) \cap L' \neq \emptyset$.

One should realize that for any $L' \subseteq \Delta^*$ and any a-transducer, M, there is some a-transducer, M', (from Δ^* to 2^{Σ^*}) so that $M'(L') = M^{-1}(L')$.

The following proposition is left as a homework problem:

Proposition 3.3. *The family of regular languages (over an alphabet Σ) is closed under both a-transductions and inverse a-transductions.*

3.8 An Application of NFA's: Text Search

A common problem in the age of the Web (and on-line text repositories) is the following:

Given a set of words, called the *keywords*, find all the documents that contain one (or all) of those words.

Search engines are a popular example of this process. Search engines use *inverted indexes* (for each word appearing on the Web, a list of all the places where that word occurs is stored).

However, there are applications that are unsuited for inverted indexes, but are good for automaton-based techniques.

Some text-processing programs, such as advanced forms of the UNIX `grep` command (such as `egrep` or `fgrep`) are based on automaton-based techniques.

The characteristics that make an application suitable for searches that use automata are:

(1) The repository on which the search is conducted is rapidly changing.

(2) The documents to be searched cannot be catalogued. For example, Amazon.com creates pages "on the fly" in response to queries.

We can use an NFA to find occurrences of a set of keywords in a text. This NFA signals by entering a final state that it has seen one of the keywords. The form of such an NFA is special.

(1) There is a start state, q_0, with a transition to itself on every input symbol from the alphabet, Σ.

(2) For each keyword, $w = w_1 \cdots w_k$ (with $w_i \in \Sigma$), there are k states, $q_1^{(w)}, \ldots, q_k^{(w)}$, and there is a transition from q_0 to $q_1^{(w)}$ on input w_1, a transition from $q_1^{(w)}$ to $q_2^{(w)}$ on input w_2, and so on, until a transition from $q_{k-1}^{(w)}$ to $q_k^{(w)}$ on input w_k. The state $q_k^{(w)}$ is an accepting state and indicates that the keyword $w = w_1 \cdots w_k$ has been found.

The NFA constructed above can then be converted to a DFA using the subset construction.

Here is an example where $\Sigma = \{a, b\}$ and the set of keywords is

$$\{aba,\ \ ab,\ \ ba\}.$$

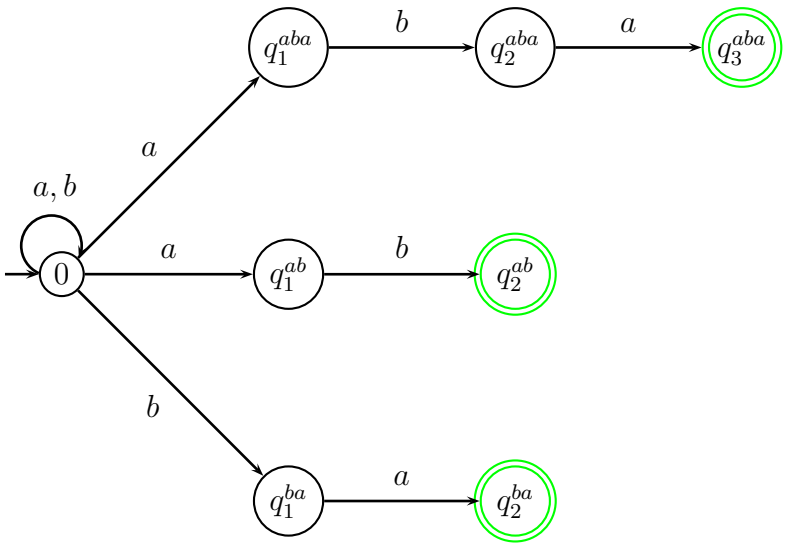

Figure 3.12: NFA for the keywords aba, ab, ba.

Applying the subset construction to the NFA, we obtain the DFA whose transition table is:

		a	b
0	0	1	2
1	$0, q_1^{aba}, q_1^{ab}$	1	3
2	$0, q_1^{ba}$	4	2
3	$0, q_1^{ba}, q_2^{aba}, q_2^{ab}$	5	2
4	$0, q_1^{aba}, q_1^{ab}, q_2^{ba}$	1	3
5	$0, q_1^{aba}, q_1^{ab}, q_2^{ba}, q_3^{aba}$	1	3

The final states are: 3, 4, 5.

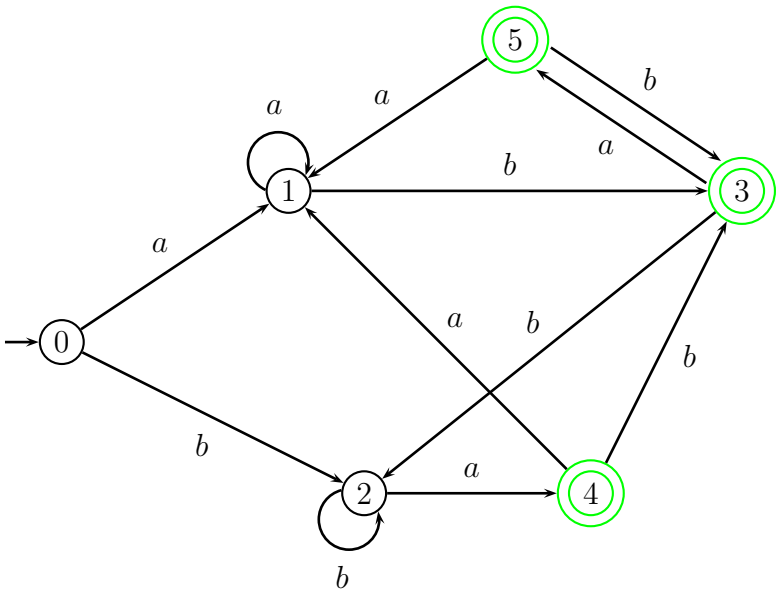

Figure 3.13: DFA for the keywords aba, ab, ba.

The good news news is that, due to the very special structure of the NFA, the number of states of the corresponding DFA is *at most* the number of states of the original NFA!

We find that the states of the DFA are (check it yourself!):

(1) The set $\{q_0\}$, associated with the start state q_0 of the NFA.

(2) For any state $p \neq q_0$ of the NFA reached from q_0 along a path corresponding to a string $u = u_1 \cdots u_m$, the set consisting of:

   (a) q_0

   (b) p

   (c) The set of all states q of the NFA reachable from q_0 by following a path whose symbols form a nonempty suffix of u, i.e., a string of the form $u_j u_{j+1} \cdots u_m$.

As a consequence, we get an efficient (w.r.t. time and space) method to recognize a set of keywords. In fact, this DFA recognizes leftmost occurrences of keywords in a text (we can stop as soon as we enter a final state).