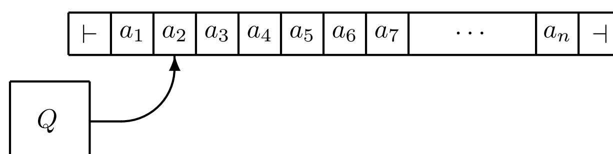
# Lecture 17

# Two-Way Finite Automata

Two-way finite automata are similar to the machines we have been studying, except that they can read the input string in either direction. We think of them as having a *read head*, which can move left or right over the input string. Like ordinary finite automata, they have a finite set $Q$ of *states* and can be either deterministic (2DFA) or nondeterministic (2NFA).

Although these automata appear much more powerful than one-way finite automata, in reality they are equivalent in the sense that they only accept regular sets. We will prove this result using the Myhill–Nerode theorem.

We think of the symbols of the input string as occupying cells of a finite tape, one symbol per cell. The input string is enclosed in left and right endmarkers $\vdash$ and $\dashv$, which are not elements of the input alphabet $\Sigma$. The read head may not move outside of the endmarkers.




Informally, the machine starts in its start state $s$ with its read head pointing to the left endmarker. At any point in time, the machine is in some state $q$ with its read head scanning some tape cell containing an input symbol $a_i$ or one of the endmarkers. Based on its current state and the symbol occupying

the tape cell it is currently scanning, it moves its read head either left or right one cell and enters a new state. It *accepts* by entering a special accept state $t$ and *rejects* by entering a special reject state $r$. The machine's action on a particular state and symbol is determined by a transition function $\delta$ that is part of the specification of the machine.

Example 17.1 Here is an informal description of a 2DFA accepting the set

$$A = \{x \in \{a,b\}^* \mid \#a(x) \text{ is a multiple of 3 and } \#b(x) \text{ is even}\}.$$

The machine starts in its start state scanning the left endmarker. It scans left to right over the input, counting the number of $a$'s mod 3 and ignoring the $b$'s. When it reaches the right endmarker $\dashv$, if the number of $a$'s it has seen is not a multiple of 3, it enters its reject state, thereby rejecting the input—the input string $x$ is not in the set $A$, since the first condition is not satisfied. Otherwise it scans right to left over the input, counting the number of $b$'s mod 2 and ignoring the $a$'s. When it reaches the left endmarker $\vdash$ again, if the number of $b$'s it has seen is odd, it enters its reject state; otherwise, it enters its accept state. □

Unlike ordinary finite automata, a 2DFA needs only a single accept state and a single reject state. We can think of it as halting immediately when it enters one of these two states, although formally it keeps running but remains in the accept or reject state. The machine need not read the entire input before accepting or rejecting. Indeed, it need not ever accept or reject at all, but may loop infinitely without ever entering its accept or reject state.

## Formal Definition of 2DFA

Formally, a 2DFA is an octuple

$$M = (Q,\ \Sigma,\ \vdash,\ \dashv,\ \delta,\ s,\ t,\ r),$$

where

- $Q$ is a finite set (the *states*),
- $\Sigma$ is a finite set (the *input alphabet*),
- $\vdash$ is the *left endmarker*, $\vdash \notin \Sigma$,
- $\dashv$ is the *right endmarker*, $\dashv \notin \Sigma$,
- $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to (Q \times \{L, R\})$ is the *transition function* ($L, R$ stand for left and right, respectively),
- $s \in Q$ is the *start state*,
- $t \in Q$ is the *accept state*, and

- $r \in Q$ is the *reject state*, $r \neq t$,

such that for all states $q$,

$$\begin{aligned} \delta(q, \vdash) &= (u, R) \quad \text{for some } u \in Q, \\ \delta(q, \dashv) &= (v, L) \quad \text{for some } v \in Q, \end{aligned} \tag{17.1}$$

and for all symbols $b \in \Sigma \cup \{\vdash\}$,

$$\begin{aligned} \delta(t, b) &= (t, R), \qquad & \delta(r, b) &= (r, R), \\ \delta(t, \dashv) &= (t, L), \qquad & \delta(r, \dashv) &= (r, L). \end{aligned} \tag{17.2}$$

Intuitively, the function $\delta$ takes a state and a symbol as arguments and returns a new state and a direction to move the head. If $\delta(p, b) = (q, d)$, then whenever the machine is in state $p$ and scanning a tape cell containing symbol $b$, it moves its head one cell in the direction $d$ and enters state $q$. The restrictions (17.1) prevent the machine from ever moving outside the input area. The restrictions (17.2) say that once the machine enters its accept or reject state, it stays in that state and moves its head all the way to the right of the tape. The octuple is not a legal 2DFA if its transition function $\delta$ does not satisfy these conditions.

Example 17.2 Here is a formal description of the 2DFA described informally in Example 17.1 above.

$$\begin{aligned} Q &= \{q_0, q_1, q_2, p_0, p_1, t, r\}, \\ \Sigma &= \{a, b\}. \end{aligned}$$

The start, accept, and reject states are $q_0$, $t$, and $r$, respectively. The transition function $\delta$ is given by the following table:

| | $\vdash$ | $a$ | $b$ | $\dashv$ |
|---|---|---|---|---|
| $q_0$ | $(q_0, R)$ | $(q_1, R)$ | $(q_0, R)$ | $(p_0, L)$ |
| $q_1$ | $-$ | $(q_2, R)$ | $(q_1, R)$ | $(r, L)$ |
| $q_2$ | $-$ | $(q_0, R)$ | $(q_2, R)$ | $(r, L)$ |
| $p_0$ | $(t, R)$ | $(p_0, L)$ | $(p_1, L)$ | $-$ |
| $p_1$ | $(r, R)$ | $(p_1, L)$ | $(p_0, L)$ | $-$ |
| $t$ | $(t, R)$ | $(t, R)$ | $(t, R)$ | $(t, L)$ |
| $r$ | $(r, R)$ | $(r, R)$ | $(r, R)$ | $(r, L)$ |

The entries marked $-$ will never occur in any computation, so it doesn't matter what we put here. The machine is in states $q_0$, $q_1$, or $q_2$ on the first pass over the input from left to right; it is in state $q_i$ if the number of $a$'s it has seen so far is $i$ mod 3. The machine is in state $p_0$ or $p_1$ on the second pass over the input from right to left, the index indicating the parity of the number of $b$'s it has seen so far. □

## Configurations and Acceptance

Fix an input $x \in \Sigma^*$, say $x = a_1 a_2 \cdots a_n$. Let $a_0 = \vdash$ and $a_{n+1} = \dashv$. Then

$$a_0 a_1 a_2 \cdots a_n a_{n+1} \quad = \quad \vdash x \dashv .$$

A *configuration* of the machine on input $x$ is a pair $(q, i)$ such that $q \in Q$ and $0 \le i \le n+1$. Informally, the pair $(q, i)$ gives a current state and current position of the read head. The *start configuration* is $(s, 0)$, meaning that the machine is in its start state $s$ and scanning the left endmarker.

A binary relation $\xrightarrow[x]{1}$, the *next configuration relation*, is defined on configurations as follows:

$$\begin{aligned} \delta(p, a_i) = (q, L) \quad &\Rightarrow \quad (p, i) \xrightarrow[x]{1} (q, i-1), \\ \delta(p, a_i) = (q, R) \quad &\Rightarrow \quad (p, i) \xrightarrow[x]{1} (q, i+1). \end{aligned}$$

The relation $\xrightarrow[x]{1}$ describes one step of the machine on input $x$. We define the relations $\xrightarrow[x]{n}$ inductively, $n \ge 0$:

- $(p, i) \xrightarrow[x]{0} (p, i)$; and
- if $(p, i) \xrightarrow[x]{n} (q, j)$ and $(q, j) \xrightarrow[x]{1} (u, k)$, then $(p, i) \xrightarrow[x]{n+1} (u, k)$.

The relation $\xrightarrow[x]{n}$ is just the $n$-fold composition of $\xrightarrow[x]{1}$. The relations $\xrightarrow[x]{n}$ are functions; that is, for any configuration $(p, i)$, there is exactly one configuration $(q, j)$ such that $(p, i) \xrightarrow[x]{n} (q, j)$. Now define

$$(p, i) \xrightarrow[x]{*} (q, j) \quad \stackrel{\text{def}}{\iff} \quad \exists n \ge 0 \; (p, i) \xrightarrow[x]{n} (q, j).$$

Note that the definitions of these relations depend on the input $x$. The machine is said to *accept* the input $x$ if

$$(s, 0) \xrightarrow[x]{*} (t, i) \quad \text{for some } i.$$

In other words, the machine enters its accept state at some point. The machine is said to *reject* the input $x$ if

$$(s, 0) \xrightarrow[x]{*} (r, i) \quad \text{for some } i.$$

In other words, the machine enters its reject state at some point. It cannot both accept and reject input $x$ by our assumption that $t \ne r$ and by properties (17.2). The machine is said to *halt* on input $x$ if it either accepts $x$ or rejects $x$. Note that this is a purely mathematical definition—the machine doesn't really grind to a halt! It is possible that the machine neither accepts nor rejects $x$, in which case it is said to *loop* on $x$. The set $L(M)$ is defined to be the set of strings accepted by $M$.

**Example 17.3** The 2DFA described in Example 17.2 goes through the following sequence of configurations on input *aababbb*, leading to acceptance:

$$(q_0, 0),\ (q_0, 1),\ (q_1, 2),\ (q_2, 3),\ (q_2, 4),\ (q_0, 5),\ (q_0, 6),\ (q_0, 7),\ (q_0, 8),$$
$$(p_0, 7),\ (p_1, 6),\ (p_0, 5),\ (p_1, 4),\ (p_1, 3),\ (p_0, 2),\ (p_0, 1),\ (p_0, 0),\ (t, 1).$$

It goes through the following sequence of configurations on input *aababa*, leading to rejection:

$$(q_0, 0),\ (q_0, 1),\ (q_1, 2),\ (q_2, 3),\ (q_2, 4),\ (q_0, 5),\ (q_0, 6),\ (q_1, 7),\ (r, 6).$$

It goes through the following sequence of configurations on input *aababb*, leading to rejection:

$$(q_0, 0),\ (q_0, 1),\ (q_1, 2),\ (q_2, 3),\ (q_2, 4),\ (q_0, 5),\ (q_0, 6),\ (q_0, 7),$$
$$(p_0, 6),\ (p_1, 5),\ (p_0, 4),\ (p_0, 3),\ (p_1, 2),\ (p_1, 1),\ (p_1, 0),\ (r, 1).$$

□

# Lecture 18

# 2DFAs and Regular Sets

In this lecture we show that 2DFAs are no more powerful than ordinary DFAs. Here is the idea. Consider a long input string broken up in an arbitrary place into two substrings $xz$. How much information about $x$ can the machine carry across the boundary from $x$ into $z$? Since the machine is two-way, it can cross the boundary between $x$ and $z$ several times. Each time it crosses the boundary moving from right to left, that is, from $z$ into $x$, it does so in some state $q$. When it crosses the boundary again moving from left to right (if ever), it comes out of $x$ in some state, say $p$. Now if it ever goes into $x$ in the future in state $q$ again, it will emerge again in state $p$, because its future action is completely determined by its current configuration (state and head position). Moreover, the state $p$ depends only on $q$ and $x$. We will write $T_x(q) = p$ to denote this relationship. We can keep track of all such information by means of a finite table

$$T_x : (Q \cup \{\bullet\}) \quad \rightarrow \quad (Q \cup \{\bot\}),$$

where $Q$ is the set of states of the 2DFA $M$, and $\bullet$ and $\bot$ are two other objects not in $Q$ whose purpose is described below.

On input $xz$, the machine $M$ starts in its start state scanning the left endmarker. As it computes, it moves its read head. The head may eventually cross the boundary moving left to right from $x$ into $z$. The first time it does so (if ever), it is in some state, which we will call $T_x(\bullet)$ (this is the purpose of $\bullet$). The machine may *never* emerge from $x$; in this case we

write $T_x(\bullet) = \perp$ (this is the purpose of $\perp$). The state $T_x(\bullet)$ gives some information about $x$, but only a finite amount of information, since there are only finitely many possibilities for $T_x(\bullet)$. Note also that $T_x(\bullet)$ depends only on $x$ and not on $z$: if the input were $xw$ instead of $xz$, the first time the machine passed the boundary from $x$ into $w$, it would also be in state $T_x(\bullet)$, because its action up to that point is determined only by $x$; it hasn't seen anything to the right of the boundary yet.

If $T_x(\bullet) = \perp$, $M$ must be in an infinite loop inside $x$ and will never accept or reject, by our assumption about moving all the way to the right endmarker whenever it accepts or rejects.

Suppose that the machine does emerge from $x$ into $z$. It may wander around in $z$ for a while, then later may move back into $x$ from right to left in state $q$. If this happens, then it will either

- eventually emerge from $x$ again in some state $p$, in which case we define $T_x(q) = p$; or
- never emerge, in which case we define $T_x(q) = \perp$.

Again, note that $T_x(q)$ depends only on $x$ and $q$ and not on $z$. If the machine entered $x$ from the right on input $xw$ in state $q$, then it would emerge again in state $T_x(q)$ (or never emerge, if $T_x(q) = \perp$), because $M$ is deterministic, and its behavior while inside $x$ is completely determined by $x$ and the state it entered $x$ in.

If we write down $T_x(q)$ for every state $q$ along with $T_x(\bullet)$, this gives all the information about $x$ one could ever hope to carry across the boundary from $x$ to $z$. One can imagine an agent sitting to the right of the boundary between $x$ and $z$, trying to obtain information about $x$. All it is allowed to do is observe the state $T_x(\bullet)$ the first time the machine emerges from $x$ (if ever) and later send probes into $x$ in various states $q$ to see what state $T_x(q)$ the machine comes out in (if at all). If $y$ is another string such that $T_y = T_x$, then $x$ and $y$ will be indistinguishable from the agent's point of view.

Now note that there are only finitely many possible tables

$$T : (Q \cup \{\bullet\}) \quad \to \quad (Q \cup \{\perp\}),$$

namely $(k+1)^{k+1}$, where $k$ is the size of $Q$. Thus there is only a finite amount of information about $x$ that can be passed across the boundary to the right of $x$, and it is all encoded in the table $T_x$.

Note also that if $T_x = T_y$ and $M$ accepts $xz$, then $M$ accepts $yz$. This is because the sequence of states the machine is in as it passes the boundary between $x$ and $z$ (or between $y$ and $z$) in either direction is completely determined by the table $T_x = T_y$ and $z$. To accept $xz$, the machine must at some point be scanning the right endmarker in its accept state $t$. Since

the sequence of states along the boundary is the same and the action when the machine is scanning $z$ is the same, this also must happen on input $yz$.

Now we can use the Myhill–Nerode theorem to show that $L(M)$ is regular. We have just argued that

$$\begin{aligned} T_x = T_y &\Rightarrow \forall z\ (M \text{ accepts } xz \iff M \text{ accepts } yz) \\ &\iff \forall z\ (xz \in L(M) \iff yz \in L(M)) \\ &\iff x \equiv_{L(M)} y, \end{aligned}$$

where $\equiv_{L(M)}$ is the relation first defined in Eq. (16.1) of Lecture 16. Thus if two strings have the same table, then they are equivalent under $\equiv_{L(M)}$. Since there are only finitely many tables, the relation $\equiv_{L(M)}$ has only finitely many equivalence classes, at most one for each table; therefore, $\equiv_{L(M)}$ is of finite index. By the Myhill–Nerode theorem, $L(M)$ is a regular set.

## Constructing a DFA

The argument above may be a bit unsatisfying, since it does not explicitly construct a DFA equivalent to a given 2DFA $M$. We can easily do so, however. Intuitively, we can build a DFA whose states correspond to the tables.

Formally, define

$$x \equiv y \quad \stackrel{\text{def}}{\iff} \quad T_x = T_y.$$

That is, call two strings in $\Sigma^*$ equivalent if they have the same table. There are only finitely many equivalence classes, at most one for each table; thus $\equiv$ is of finite index. We can also show the following:

(i) The table $T_{xa}$ is uniquely determined by $T_x$ and $a$; that is, if $T_x = T_y$, then $T_{xa} = T_{ya}$. This says that $\equiv$ is a right congruence.

(ii) Whether or not $x$ is accepted by $M$ is completely determined by $T_x$; that is, if $T_x = T_y$, then either both $x$ and $y$ are accepted by $M$ or neither is. This says that $\equiv$ refines $L(M)$.

These observations together say that $\equiv$ is a Myhill–Nerode relation for $L(M)$. Using the construction $\equiv \mapsto M_{\equiv}$ described in Lecture 15, we can obtain a DFA for $L(M)$ explicitly.

To show (i), we show how to construct $T_{xa}$ from $T_x$ and $a$.

- If $p_0, p_1, \ldots, p_k, q_0, q_1, \ldots, q_k \in Q$ such that $\delta(p_i, a) = (q_i, L)$ and $T_x(q_i) = p_{i+1}$, $0 \le i \le k-1$, and $\delta(p_k, a) = (q_k, R)$, then $T_{xa}(p_0) = q_k$.

- If $p_0, p_1, \ldots, p_k, q_0, q_1, \ldots, q_{k-1} \in Q$ such that $\delta(p_i, a) = (q_i, L)$ and $T_x(q_i) = p_{i+1}$, $0 \le i \le k-1$, and $p_k = p_i$, $i < k$, then $T_{xa}(p_0) = \bot$.

- If $p_0, p_1, \ldots, p_k, q_0, q_1, \ldots, q_k \in Q$ such that $\delta(p_i, a) = (q_i, L)$, $0 \le i \le k$, $T_x(q_i) = p_{i+1}$, $0 \le i \le k-1$, and $T_x(q_k) = \bot$, then $T_{xa}(p_0) = \bot$.

- If $T_x(\bullet) = \bot$, then $T_{xa}(\bullet) = \bot$.

- If $T_x(\bullet) = p$, then $T_{xa}(\bullet) = T_{xa}(p)$.

For (ii), suppose $T_x = T_y$ and consider the sequence of states $M$ is in as it crosses the boundary in either direction between the input string and the right endmarker $\dashv$. This sequence is the same on input $x$ as it is on input $y$, since it is completely determined by the table. Both strings $x$ and $y$ are accepted iff this sequence contains the accept state $t$.

We have shown that the relation $\equiv$ is a Myhill–Nerode relation for $L(M)$, where $M$ is an arbitrary 2DFA. The construction $\equiv \mapsto M_\equiv$ of Lecture 15 gives a DFA equivalent to $M$. Recall that in that construction, the states of the DFA correspond in a one-to-one fashion with the $\equiv$-classes; and here, each $\equiv$-class $[x]$ corresponds to a table $T_x : (Q \cup \{\bullet\}) \to (Q \cup \{\bot\})$.

If we wanted to, we could build a DFA $M'$ directly from the tables:

$$
\begin{aligned}
Q' &\stackrel{\text{def}}{=} \{T : (Q \cup \{\bullet\}) \to (Q \cup \{\bot\})\}, \\
s' &\stackrel{\text{def}}{=} T_\epsilon, \\
\delta'(T_x, a) &\stackrel{\text{def}}{=} T_{xa}, \\
F' &\stackrel{\text{def}}{=} \{T_x \mid x \in L(M)\}.
\end{aligned}
$$

The transition function $\delta'$ is well defined because of property (i), and $T_x \in F'$ iff $x \in L(M)$ by property (ii). As usual, one can prove by induction on $|y|$ that

$$\widehat{\delta}'(T_x, y) = T_{xy};$$

then

$$
\begin{aligned}
x \in L(M') &\iff \widehat{\delta}'(s', x) \in F' \\
&\iff \widehat{\delta}'(T_\epsilon, x) \in F' \\
&\iff T_x \in F' \\
&\iff x \in L(M).
\end{aligned}
$$

Thus $L(M') = L(M)$.

Another proof, due to Vardi [113], is given in Miscellaneous Exercise 61.

## Historical Notes

Two-way finite automata were first studied by Rabin and Scott [96] and Shepherdson [105]. Vardi [113] gave a shorter proof of equivalence to DFAs (Miscellaneous Exercise 61).