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

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

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 $T_x(\bullet) = \bot$

(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

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

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

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

That is, call two strings in Σ^* 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 $T_x(q_i) = p_{i+1}$, $0 \leq i \leq k-1$, 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 $T_x(q_i) = p_{i+1}$, $0 \leq i \leq k-1$, and $p_k = p_i$, $i < k$, then $T_{xa}(p_0) = \perp$.
- If $p_0, p_1, \ldots, p_k, q_0, q_1, \ldots, q_k \in Q$ such that $\delta(p_i, a) = (q_i, L)$, $0 \leq i \leq k$, $T_x(q_i) = p_{i+1}$, $0 \leq i \leq k-1$, and $T_x(q_k) = \perp$, then $T_{xa}(p_0) = \perp$.
- If $T_x(\bullet) = \perp$, then $T_{xa}(\bullet) = \perp$.
- 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 $T_x : (Q \cup \{\bullet\}) \to (Q \cup \{\perp\})$.

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

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

The transition function δ' 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 [122], is given in Miscellaneous Exercise 61.

Historical Notes

Two-way finite automata were first studied by Rabin and Scott [102] and Shepherdson [114]. Vardi [122] gave a shorter proof of equivalence to DFAs (Miscellaneous Exercise 61).