Deterministic finite state acceptors

Another method of describing sets of strings is via deterministic finite acceptors. These are automata that are like Turing machines with no tape. A deterministic finite acceptor (or dfa) has a finite alphabet of characters, a finite set of states, a start state, a set of accepting states, and a transition function. The transition function is defined for pairs consisting of a state and a character, and produces a state.

As an example, we specify a dfa over the alphabet $\{a, b\}$ that has states $\{1, 2\}$, with the start state being state 1 and the set of accepting states being $\{2\}$, and a transition function δ given by

$$\delta(1, a) = 2$$
$$\delta(1, b) = 1$$
$$\delta(2, a) = 1$$
$$\delta(2, b) = 2$$

This information can also be conveyed by a diagram in which each state is represented by a circle and each transition by an arrow. For example, the transition for $\delta(1, a) = 2$ would be shown as an arrow from state 1 to state 2 labelled by a, and the transition for $\delta(1, b) = 1$ would be shown as an arrow from state 1 to state 1 labelled by b. The initial state is indicated by a single incoming arrow, (not coming from another state) and the accepting states are represented by placing another circle around the state symbol. (Breathe easy, there will be no ASCII drawings of deterministic finite state acceptors.)

For each string of input symbols, we determine whether the dfa accepts the string as follows. Starting at the start state, we follow the transition corresponding to each symbol in the string, from left to right, until we have used up all the symbols in the string. If this process ends in an accepting state, the string is accepted. Otherwise, the string is not accepted.

For example, for the dfa we defined above, if we consider the string *abbab*, we start in state 1, follow the transition to state 2 for the first symbol (a), follow the transition to the state 2 for the next symbol (b), follow the transition to the state 2 for the next symbol (b), follow the transition to the state 1 for the next symbol (a), and follow the transition to the state 1 for the final symbol (b). Since this process ends in the state 1, which is a not an accepting state of this dfa, the string *abbab* is not accepted. Similar reasoning leads us to the conclusion that the string *ababab* is accepted, because starting in the start state, we end in state 2. Experimenting with other strings, we come to the conclusion that this dfa accepts a string of a's and b's if and only if it contains an odd number of a's.

Regular sets

A set of strings is called *regular* if it can be denoted by a regular expression. It is a theorem that a set is regular if and only if it is accepted by a deterministic finite acceptor. The proof is not difficult, though we do not have time for it in this course. This means that for every regular expression, there is an equivalent dfa and vice versa. As an exercise, try to find a regular expression to denote the set of strings of a's and b's that contain an odd number of a's.

(Spoiler: one possible answer to this is the expression $b^*a(b^*|ab^*a)^*$. To see how this works, an initial sequence of b's keeps us in state 1 and an a moves us to state 2. In state 2, either a sequence of b's or an a followed by a sequence of b's followed by an a takes us back to state 2. This can be repeated at will to return to state 2, which is accepting.)

Conversely, suppose we want to construct a dfa to accept the set of strings denoted by the regular expression $c(a|d)(a|d)^*r$. We start with state 1, and let the character c move to state 2. In state 2, either

a or d moves to state 3. In state 3, either a or d stays in state 3, while r moves to state 4, which is the single accepting state in the machine. All of the transitions we haven't yet mentioned should move to state 5, which is a non-accepting state for which a, c, d, and r all return to state 5. State 5 is a "dead state" – once it is reached, no extension of the string will be accepted. Putting all this together, our acceptor has alphabet $\{a, c, d, r\}$, states $\{1, 2, 3, 4, 5\}$, initial state 1, final states $\{4\}$, and a transition function defined as follows.

$$\delta(1, a) = 5, \delta(1, c) = 2, \delta(1, d) = 5, \delta(1, r) = 5,$$
$$\delta(2, a) = 3, \delta(2, c) = 5, \delta(2, d) = 3, \delta(2, r) = 5,$$
$$\delta(3, a) = 3, \delta(3, c) = 5, \delta(3, d) = 3, \delta(3, r) = 4,$$
$$\delta(4, a) = 5, \delta(4, c) = 5, \delta(4, d) = 5, \delta(4, r) = 5,$$
$$\delta(5, a) = 5, \delta(5, c) = 5, \delta(5, d) = 5, \delta(5, r) = 5.$$

For example, to accept the string *cadr*, the sequence of states we visit is: 1, 2, 3, 3, 4.

Palindromes are not a regular set

Next we exhibit a relatively simple set of strings that is not a regular set. A string is a palindrome if it is equal to its reverse. Over the alphabet $\{a, b\}$ this includes strings such as the following:

$$\lambda, a, b, aa, bb, aaa, aba, bab, bbb, aaaa, abba, baab, bbbb, \ldots$$

The set of strings over the alphabet $\{a, b\}$ that are palindromes is not regular, in other words, it cannot be denoted by a regular expression or accepted by a deterministic finite acceptor.

Why not? We sketch the idea of a proof. Suppose for the sake of contradiction that the set of palindromes over $\{a, b\}$ is accepted by a deterministic finite acceptor M. This means that every palindrome is accepted by M and every non-palindrome is rejected by M. Let n denote the number of states in M.

Suppose we begin in the start state of M and feed it n a's. Because M has only n states, it must visit at least one state twice in this process. Suppose M arrives at state k after a string of i a's, and again arrives at state k after a string of $i + j$ a's, where $j > 0$. We use the notation a^i for a string of i a's. Using this notation, M's finite state memory cannot distinguish between a^i and a^{i+j}. If from the state k we feed M the string ba^i, we must reach an accepting state of M, because a^iba^i is a palindrome, and M accepts all palindromes. But this means that M also accepts the string $a^{i+j}ba^i$, which is NOT a palindrome (because $j > 0$.) This is a contradiction, showing that there is no deterministic finite acceptor that accepts the set of palindromes.