Supplementary Notes on Finite Automata

Foundations of Computer Science

Spring 2017

The Language of a Finite Automaton is Regular

We stated in class the following result:

Theorem: *A language A is accepted by some finite automaton M if and only if A is regular.*

We showed one direction of this equivalence: when A is regular, we can construct a finite automaton which accepts A.

To argue the other direction, that if a language is accepted by a finite automaton, then it is regular, we give a process for constructing a regular expression which denotes the same language as that accepted by any given finite automaton.

Let $M = (Q, \Sigma, \Delta, s, F)$ be a finite automaton.

We define a recursive function $R(p, q, X)$ that returns a regular expressions representing all strings that take automaton M from state p to state q through transitions going only through states in X, with the possible exception of p and q which need not lie in X.

Here is the recursive function definition. It recurses on the size of X.

When X is empty, let $a_1, \ldots, a_k$ be all the symbols in Σ such that $\langle p, a_i, q\rangle \in \Delta$, that is, that make the automaton transition from p to q. We have two cases: when $p \neq q$,

$$R(p, q, \varnothing) = \begin{cases} a_1 + \cdots + a_k & \text{if } k \geq 1 \\ 0 & \text{if } k = 0 \end{cases}$$

and when $p = q$,

$$R(p, p, \varnothing) = \begin{cases} a_1 + \cdots + a_k + 1 & \text{if } k \geq 0 \\ 1 & \text{if } k = 0. \end{cases}$$

For the recursive case, choose any element $v \in X$. (Different choices will yield different regular expressions at end, but all of those different regular expressions will denote the same language.) Take:

$$R(p, q, X) = R(p, q, X \backslash \{v\}) + R(p, v, X \backslash \{v\})(R(v, v, X \backslash \{v\})^* R(v, q, X \backslash \{v\})$$

To understand this definition, observe that any string that takes the automaton from p to q with all intermediate states in X either:

- never takes the automaton through state v, and those strings are given by this part of the regular expression:
$$R(p, q, X\backslash\{v\})$$
- or makes the automaton reach state v a first time, given by this part of the regular expression:
$$R(p, v, X\backslash\{v\})$$
followed by a finite number of loops (possibly zero) from v back to itself without going through v in between and always staying in X, give by this part:
$$(R(v, v, X\backslash\{v\}))^*$$
followed by taking the automaton from v to q without going through v, given by this part:
$$R(v, q, X\backslash\{v\}).$$

Once we have such a function R, we can construct the regular expression whose language is the language accepted by M, namely all the strings taking M from the start state to any of the final states $f_1, \ldots, f_k$ through any state of M by:

$$R(s, f_1, Q) + \cdots + R(s, f_k, Q).$$

The result is usually a *huge* unreadable regular expression. Thankfully, it can be simplified because regular expression obey algebraic-like simplification rules. But that's beyond the scope of these notes.

Complementation of Regular Languages

Using finite automata, we can show that the complement of a regular language is regular.

Let A be a regular language. That means that there is a deterministic finite automaton $M = (Q, \Sigma, \Delta, s, F)$ that accepts A. (The determinism is important.)

The first thing we do is *complete* M, in such a way that for every symbol in Σ and for every $q \in Q$, there is a $p \in Q$ such that $\langle q, a, p\rangle \in \Delta$ — in other words, there is a transition at every state for every symbol. We do so by introducing a new *deadend* state to which we transition whenever the original M doesn't have a suitable transition.

Let $M' = (Q', \Sigma, \Delta', s, F)$ by taking

- $Q' = Q \cup \{s_{deadend}\}$ where $s_{deadend}$ is a new state not in Q

- $\Delta' = \Delta \cup \{\langle q, a, s_{deadend}\rangle \mid q \in Q, a \in \Sigma, \text{and there is no } p \in Q \text{ with } \langle q, a, p\rangle \in \Delta\}$

It is not difficult to argue that M' accepts the same language as M. Now, take M', and construct the finite automaton $\overline{M'} = (Q', \Sigma, \Delta', s, Q' - F)$. The language accepted by $\overline{M'}$ is $\overline{A}$, and therefore $\overline{A}$ is regular.

Non-Regular Languages

Finite automata also let us argue that some languages are not regular.

Let $A = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$. That is, A is the language of all strings made up of a sequence of $\mathtt{a}$s followed by a sequence of the same number of $\mathtt{b}$s. I claim A is not regular.

Let's argue by contradiction. Let's assume A *is* in fact regular, and derive a contradiction. This means that our assumption cannot be, and thus A cannot be regular.

If A is regular, then there exists a deterministic finite automaton $M = (Q, \{\mathtt{a}, \mathtt{b}\}, \Delta, s, F)$ that accepts A. Let $N = |Q|$, the number of states in M.

Consider the string $u = \mathtt{a}^{2N}\mathtt{b}^{2N}$, which is a string in A. Therefore M accepts u. Since M has only N states and string u has length $4N$, any sequence of transitions in M that accepts u must hit one state, call it q, at least twice. Moreover, because the first $2N$ symbols of u are all $\mathtt{a}$, that state q must be reached from s after $k < 2N$ transitions labeled $\mathtt{a}$ and also after $k + m < 2N$ transitions labeled $\mathtt{a}$ for some $m > 0$.

In other words, there is a loop of length m from q to q with transitions labeled $\mathtt{a}$. The string $u = \mathtt{a}^{2N}\mathtt{b}^{2N} = \mathtt{a}^k\mathtt{a}^m\mathtt{a}^{2N-k-m}\mathtt{b}^{2N}$ is accepted by the automaton by first going from s to q (via $\mathtt{a}^k$) and then from q to q (via $\mathtt{a}^m$) and then from q to a final state (via $\mathtt{a}^{2N-k-m}\mathtt{b}^{2N}$).

But consider the string $v = \mathtt{a}^k\mathtt{a}^m\mathtt{a}^m\mathtt{a}^{2N-k-m}\mathtt{b}^{2N}$. It also must be accepted by A! Indeed, $\mathtt{a}^k$ takes M from s to q, $\mathtt{a}^m$ takes M from q to q, $\mathtt{a}^m$ takes M from q to q again (second time around the loop!), and finally $\mathtt{a}^{2N-k-m}\mathtt{b}^{2N}$ takes M from q to a final state. So M accepts v. But v is of the form $\mathtt{a}^m\mathtt{a}^{2N}\mathtt{b}^{2N}$ and since $m > 0$, $v \notin A$. So A cannot be the language accepted by M. But we chose M specifically to have language A. That's our contradiction.

Our initial assumption cannot be: A is not regular.

A similar argument shows that $Pal(\Sigma)$, the set of all palindromes over Σ, is not regular.

A general form of the argument is formalized as the *Pumping Lemma for Regular Languages*:

Theorem (Pumping Lemma): *If A is regular language, then there is a number p with the property that any string $u \in A$ of length at least p can be written as $u = xyz$ for x, y, z satisfying:*

(i) *for every $i \geq 0$, $xy^iz \in A$,*

(ii) $|y| > 0$, *and*

(iii) $|xy| \leq p$.