Notes on Finite Automata

Foundations of Computer Science

February 4, 2016

Recall that a language $A \subseteq \Sigma^*$ is *regular* if there exists a regular expression r such that $L(r) = A$.

From this definition and from the definition of the language of a regular expression, it is easy to see that:

- Every finite language (i.e., language with a finite number of strings) is regular, including the empty language $\varnothing$. (Why?)
- if A and B are regular languages, then $A \cup B$ and $A \cdot B$ are regular languages. (For example, if A and B are regular, then there are regular expressions r_A and r_B such that $L(r_A) = A$ and $L(r_B) = B$, and it's immediate that $L(r1 + r2) = A \cup B$, so $r_1 + r_2$ is the regular expression showing that $A \cup B$ is regular.)
- If A is regular, then A^* is regular. (In particular, if we consider Σ as a language (one string per alphabet symbol) which is regular because it is finite, then Σ^*, the set of all strings over Σ, is regular.)

All that comes from the definition of regular expressions. It also turns out that when A is regular, then $\overline{A}$ is regular, and so is, for example, $rev(A) = \{\sigma^R \mid \sigma \in A\}$, where σ^R is the reverse of string σ. We can't show that just yet. (It's not particular easy using regular expressions, though not impossible.)

Not every language is regular. The set of all *palindromes* over an alphabet Σ is not regular, for instance. Again, we can't show that just yet.

Finite Automata

A (nondeterministic) *finite automaton* is a tuple

$$M = (Q, \Sigma, \Delta, s, F)$$

where

- Q is a finite set of states

- Σ is a finite alphabet
- $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, made up of triples (p, a, q) indicating that there is a transition from state p to state q labeled by the symbol a.
- $s \in Q$ is the start state
- $F \subset Q$ is a set of accepting (or final) states.

When given an input string $\sigma = a_1 \dots a_k$, we say that automaton M *accepts* σ if there is a way to start from the start state s, follow transitions labeled by $a_1, a_2, \dots, a_k$, and end up in an accepting state. If there is no such way, we say that the automaton *rejects* σ.

Formally: $M = (Q, \Sigma, \Delta, s, F)$ accept $\sigma = a_1 \dots a_k$ if there exists $q_0, q_1, \dots, q_k \in Q$ such that $q_0 = s$, $q_k \in F$, and $(q_{i-1}, a_i, q_i) \in \Delta$ for all $1 \leq i \leq k$.

The language accepted by M is

$$L(M) = \{\sigma \mid M \text{ accepts } \sigma\}$$

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

One direction is pretty easy: if A is regular, then there is a finite automaton that accepts A. We can do it by constructing an automaton from the regular expression that witnesses that A is regular. The other direction will be done in tutorial: if A is accepted by a finite automaton M, then we can construct a regular expressions whose language is A.

We can define a recursive procedure that constructs a finite automaton from a regular expression that accepts the language of the regular expression. It's going to be a recursive procedure that recurses over the structure of the regular expression.

For the bases case of the recursion, it is easy to construct finite automata for $L(0) = \varnothing$, $L(1) = \{\epsilon\}$ and $L(a) = \{a\}$.

If r_1 and r_2 are regular expressions, by recursion we can obtain finite automata $M_1 = (Q_1, \Sigma, \Delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \Delta_2, s_2, F_2)$ that accept the languages of r_1 and r_2 respectively. Assume that Q_1 and Q_2 have no states in common—you can do that by renaming states if needed, adjusting the transition relation in consequence. We can create a finite automaton M that accepts $L(r_1 + r_2) = L(r_1) \cup L(r_2)$ as follows:

$$M = (Q, \Sigma, \Delta, s, F)$$

where

- $Q = Q_1 \cup Q_2 \cup \{s\}$ where s is a new state not in Q_1 or Q_2
- $\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, a, q) \mid \exists a, q \text{ such that } (s_1, a, q) \in \Delta_1\} \cup \{(s, a, q) \mid \exists a, q \text{ such that } (s_2, a, q) \in \Delta_2\}$

- $F = F_1 \cup F_2$

The construction for $r_1 r_2$ was given in class, and the construction for r^* was left as an exercise.

A *deterministic finite automaton* (DFA) is a special case of a finite automaton where there is exactly one transition out of every state labeled with a symbol from the alphabet, for every symbol of the alphabet. In order words, for a DFA, it suffices to start from the start state and follow the one and only path that follows the symbols in the input string. Reaching an accepting state mean you accept the input string, otherwise, you reject it.

A DFA is a finite automaton where the transition relation Δ is in fact a function. To emphasize that, I will sometimes use $\delta : Q \times \Sigma \longrightarrow Q$ to denote that transition function.

Even though DFAs seem more restrictive than general finite automaton, it turns out that DFAs accept exactly the regular languages. Clearly, since every DFA is a finite automaton, if a DFA accepts a language A, then A is accepted by a finite automaton, and by the result above, A is regular. If A is regular, then we know it can be accepted by a finite automaton, but it's not clear it can be accepted by a *deterministic* finite automaton.

It turns out that yes, because deterministic finite automata can *simulate* general finite automata. The idea is to consider all possible paths through a finite automaton as it's trying to accept a string, and take those sets of states that the finite automaton gets to as states of the deterministic finite automaton.

Formally, for every finite automaton M, there exists a deterministic finite automaton N that accepts the same language as M, constructed as follows. If $M = (Q_M, \Sigma, \Delta_M, s_M, F_M)$, construct

$$N = (Q, \Sigma, \delta, s, F)$$

(remember, we're constructing a deterministic finite automaton, so we need a transition function δ):

- $Q = \{X \mid X \subseteq Q_M\}$
- $\delta(X, a) = \{q \in Q_M \mid \exists (p, a, q) \in \Delta_M \text{ for some } p \in X\}$
- $s = \{s_M\}$
- $F = \{X \subseteq Q_M \mid X \cap F_M \neq \varnothing\}$

(This is called the *subset construction*.)