Notes on Finite Automata

Foundations of Computer Science

February 2, 2017

A *finite automaton* is a structure

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

where

- Q is a finite set of states
- Σ is a finite alphabet
- $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation: $\langle p, a, q \rangle \in \Delta$ when there is a transition from state p to state q labeled by the symbol a
- $s \in Q$ is the start state
- $F \subseteq Q$ is a set of final states.

Finite automaton M *accepts* string $u = a_1 \dots a_k$ if there is a path in M starting with s labeled by $a_1, a_2, \dots, a_k$, and ending up in a final state.

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

The language accepted by M is

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

Example: The following finite automaton accepts exactly the strings over $\{\mathtt{a}, \mathtt{b}\}$ of even length:

$$M_{even} = (\{1, 2\}, \{\mathtt{a}, \mathtt{b}\}, \Delta_{even}, 1, \{1\})$$

where the transitions go from state 1 to state 2 and back no matter the symbol:

$$\Delta_{even} = \{\langle 1, \mathtt{a}, 2\rangle, \langle 1, \mathtt{b}, 2\rangle, \langle 2, \mathtt{a}, 1\rangle, \langle 2, \mathtt{b}, 1\rangle\}$$

Example: The following finite automaton accepts exactly the strings over $\{\mathtt{a}, \mathtt{b}\}$ whose last two symbols are $\mathtt{a}$:

$$M_{last} = (\{1, 2, 3\}, \{\mathtt{a}, \mathtt{b}\}, \Delta_{last}, 1, \{3\})$$

where

$$\Delta_{last} = \{\langle 1, \mathtt{a}, 1\rangle, \langle 1, \mathtt{b}, 1\rangle, \langle 1, \mathtt{a}, 2\rangle, \langle 2, \mathtt{a}, 3\rangle\}$$

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

One direction of the equivalence is pretty easy, namely showing that when A is regular there is some finite automaton that accepts A.

Since A is regular, this means that there is a regular expression r with $L(r) = A$. We can define a recursive procedure that constructs a finite automaton from a regular expression and that accepts the language of the regular expression. The procedure will recurse 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 $L(\mathtt{a}) = \{\mathtt{a}\}$:

$$M_0 = (\{1\}, \Sigma, \varnothing, 1, \varnothing)$$

$$M_1 = (\{1\}, \Sigma, \varnothing, 1, \{1\})$$

$$M_{\mathtt{a}} = (\{1, 2\}, \Sigma, \{\langle 1, \mathtt{a}, 2\rangle\}, 1, \{2\})$$

For the recursive cases, first consider $r_1 + r_2$. 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. Without loss of generality, we can take Q_1 and Q_2 to have no states in common—we can simply rename states if needed, adjusting the transition relation and the state and final states. We can create a finite automaton $M_{r_1+r_2}$ that accepts $L(r_1 + r_2) = L(r_1) \cup L(r_2)$ by putting together M_1 and M_2 and creating a new start state that transitions to both M_1 and M_2. (More precisely, the new start state transitions to every state that the original start states s_1 and s_2 can transition to, with the same label.) Final states are those from M_1 and M_2, with the addition that the new start start will be final if any one of s_1 or s_2 is final. Formally:

$$M_{r_1+r_2} = (Q, \Sigma, \Delta, s_{new}, F)$$

where

- $Q = Q_1 \cup Q_2 \cup \{s_{new}\}$ where s_{new} is a new state not in Q_1 or Q_2
- $\Delta = \Delta_1 \cup \Delta_2 \cup \{\langle s_{new}, a, q\rangle \mid \langle s_1, a, q\rangle \in \Delta_1\} \cup \{\langle s_{new}, a, q\rangle \mid \langle s_2, a, q\rangle \in \Delta_2\}$
- F is $F_1 \cup F_2 \cup \{s_{new}\}$ if $s_1 \in F_1$ or $s_2 \in F_2$, and is $F_1 \cup F_2$ otherwise.

Consider $r_1 r_2$. Again, 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, where Q_1 and Q_2 have no states in common. We can create a finite automaton $M_{r_1 r_2}$ that accepts $L(r_1 r_2) = L(r_1) \cdot L(r_2)$ by putting together M_1 and M_2 and basically sending the final states of M_1 into M_2. (More precisely, every final state of M_1 transitions to every state that the original start states s_2 transitions to, with the same label.) The start state is the original start state s_1 of M_1. Final states are those from M_2, along with the final states of M_1 if the original start state s_2 is final in M_2 (why?). Formally:

$$M_{r_1 r_2} = (Q, \Sigma, \Delta, s_1, F)$$

where

- $Q = Q_1 \cup Q_2$
- $\Delta = \Delta_1 \cup \Delta_2 \cup \{\langle f, a, q\rangle \mid f \in F_1, \langle s_2, a, q\rangle \in \Delta_2\}$
- F is $F_1 \cup F_2$ if $s_2 \in F2$, and is F_2 otherwise.

Finally, consider ${r_1}^*$. Again, by recursion we can obtain a finite automaton $M_1 = (Q_1, \Sigma, \Delta_1, s_1, F_1)$ that accepts the language of r_1. We can create a finite automaton $M_{{r_1}^*}$ that accepts $L({r_1}^*) = L(r_1)^*$ by modifying M_1 to send the final states of M_1 back to the start. (More precisely, every final state of M_1 transitions to every state that the original start state s_1 transitions to, with the same label.) The one subtlety is the treatment of the empty string. Since $L(r_1)^*$ always contains the empty string, we need to make sure the resulting automaton accepts the empty string, which we cannot guarantee by simply taking the original start state of M_1 and making it final—if it wasn't final to start with, and there are transitions back to it, then making it final might make it possible to accept more strings than we want. (Why?) The fix is to create a new start state like we did for $M_{r_1+r_2}$, that transitions to every state that s_1 transitions to, and that is indeed final. Final states are those from M_1, along with the new start state. Formally:

$$M_{{r_1}^*} = (Q, \Sigma, \Delta, s_{new}, F_1 \cup \{s_{new}\})$$

where

- $Q = Q_1 \cup \{s_{new}\}$ where s_{new} is a new state not in Q_1

- $\Delta = \Delta_1 \cup \{\langle f, a, q\rangle \mid f \in F_1, \langle s_1, a, q\rangle \in \Delta_1\} \cup \{\langle s_{new}, a, q\rangle \mid \langle s_1, a, q\rangle \in \Delta_1\}$.

Proving the other direction of the theorem, that the language of a finite automaton is always regular, is a bit more painful. The simplest way is to create a regular expression from a description of the finite automaton, but the construction is awkward. I've added some notes about it in the supplemental material for this lecture.

In the special case when a finite automaton has, for every symbol of the alphabet, at most one transition out of every state labeled with symbol, we say the finite automaton is *deterministic*. The transition relation Δ for a deterministic finite automaton (DFA) is in fact a partial function (a function that is not defined on all elements of its domain) from $Q \times \Sigma$ to Q. 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. If you don't get stuck on the way and reach a final state after consuming all symbols from the input string, you accept the input string.

(When we want to emphasize that an automaton may not be deterministic, we often use the term nondeterministic.)

Example: Finite automaton M_{even} from earlier is a deterministic finite automaton. Finite automaton M_{last} is a nondeterministic finite automaton.

Even though DFAs seem more restrictive than nondeterministic finite automata, 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 regular. But if A is regular, while we know it can be accepted by a finite automaton, it's not clear it can be accepted by a *deterministic* finite automaton.

Consider the following way to "execute" a nondeterministic finite automaton M. The idea is to look at the possible transitions that can be taken for a given symbol of the input all at once. Intuitively, we begin in the start state, and for every symbol of the input string in order, we keep track of the possible states we can reach by following a transition labeled with that symbol from any of the possible states we have already reached. Thus, at every step in this process, we maintain a set of possible states that M can be in. If at the end of the string one of the possible states is final, then M accepts the string.

This process, where we consider *sets* of states of M, in fact describes the execution of a deterministic finite automaton, in which states are sets of states of M, and there is a transtion from a set X of states of M to a set Y of states of M labeled a exactly when the transitions of M labeled a from any state in X lead to the states in Y.

Formally, for every finite automaton M, there exists a deterministic finite automaton $\widehat{M}$ that accepts the same language as M, constructed as follows. If $M = (Q, \Sigma, \Delta, s, F)$, construct

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

where:

- $\widehat{Q} = \{X \mid X \subseteq Q\}$

- $\widehat{\Delta} = \{\langle X, a, Y\rangle \mid X \subseteq Q, Y = \{q \in Q \mid \langle p, a, q\rangle \in \Delta \text{ for some } p \in X\}\}$
- $\widehat{s} = \{s\}$
- $\widehat{F} = \{X \subseteq Q \mid X \cap F \neq \varnothing\}$

This construction is called the *subset construction*.

We can see that $\widehat{M}$ is deterministic because any subset $X \subseteq Q$ uniquely determines a subset Y to transition to in the definition of $\widehat{\Delta}$.