Lecture 3:
Nondeterministic Finite Automata

Valentine Kabanets

September 20, 2016

Notation: when we have missing transition arrows in our FA diagram, we mean that they all go to a "trap state", which is a non-final state such that once entered by the FA, it will never change to a new state.

1 Equivalence between NFA and DFA

Obviously, a DFA is a special case of an NFA. Hence, every regular language is also an NFA language. To prove the converse, we will show how to take a given NFA $N = (Q, \Sigma, \delta, q_0, F)$ and build an equivalent DFA M.

The idea is to keep track of a subset of states that the NFA could be in at a given point in time. Thus, the states of the DFA to be constructed will be all subsets of states of the original NFA. One complication is ϵ-transitions. To deal with those, we introduce the ϵ-closure of a set R of states of the original NFA defined by

$E(R) = \{q \in Q \mid q$ is reachable from some $r \in R$ by a possibly empty path of ϵ-transitions$\}$.

Formally, we define the DFA $M = (Q', \Sigma, \delta', q_0', F')$, where

- $Q' = \mathcal{P}(Q)$,
- $q_0' = E(\{q_0\})$,
- $F' = \{R \in Q' \mid R \cap F \neq \emptyset\}$, and
- for each $R \in Q'$, and $a \in \Sigma$, define $\delta(R, a) = \cup_{r \in R} E(\delta(r, a))$.

It is easy to see that M accepts exactly the same language as N.

Remark 1. *The number of states of M is* exponentially *bigger than that of N. This is unavoidable. For "small" size NFA N, the given construction can be used to build an equivalent DFA. But it becomes impractical for reasonably "large" NFA (say NFA on 100 states). The time to build a DFA from a given NFA is dominated by the function $O(2^{|Q|})$, where $|Q|$ is the number of states of the given NFA.*

For practical purposes, it is better to keep your NFA as is, and just simulate it on a given input string by following the ideas of the DFA construction above (i.e., start at q_0', and make transitions while keeping track of the current set of states).

2 Regular expressions

We have the following inductive definition of regular expressions.

- The following are regular expressions: $\emptyset$, ϵ, and a for every $a \in \Sigma$.
- If R_1 and R_2 are regular expressions, then so are $(R_1 \cup R_2)$, $(R_1 \circ R_2)$, and (R_1^*).

The regular expressions are intended to represent languages. The basic expressions $\emptyset$, ϵ, and a represent the languages $\emptyset$, $\{\epsilon\}$, and $\{a\}$, respectively. The operations on R_1 and R_2 naturally correspond to the same operations on the languages represented by R_1 and R_2.

We will see that the class of languages represented by regular expressions is exactly the class of regular languages (i.e., the class of DFA languages). This leads to the following efficient way to test if a given string w has the pattern described by a given regular expression R. Take R and build an equivalent DFA (or NFA) for it; then run the constructed FA on the input string w.

3 Regular expressions and regular languages

We show that the two are equivalent.

Theorem 1. *A language L is regular iff $L = L(R)$ for some regular expression R.*

Proof. In one direction, we take a regular expression and construct an NFA for the language corresponding to that expression. The construction is by induction on the structure of the regular expression. For $\emptyset$, ϵ, and a (for $a \in \Sigma$), we have simple NFA (in fact, DFA).

For $R = R_1 \cup R_2$, we by induction construct the NFA for R_1 and R_2, and then construct the NFA for the union of the languages, as discussed last time. The case of $R = R_1 \circ R_2$ and $R = (R_1)^*$ are similar.

For the other direction, we take a DFA, and extract from it a regular expression. The idea is to define a *generalized* NFA whose arrows are labeled with regular expressions. We remove one state at a time, preserving the language accepted. When we have only two states left, the arrow connecting them will be labeled by the regular expression we want.

More precisely, we start with a DFA, and transform it to an automaton with a new start state (with no incoming arrows) and a single accept state (without any outgoing arrows). We label all arrows by the regular expressions, including the "missing" arrows which get labeled by $\emptyset$. Then we pick one state to rip out, say q_{rip}; this can be any state other than the start or the accept states. For every pair of states p, q (where p may equal q), we update the label of the arrow $p \to q$ in the new automaton (where we removed q_{rip}) as follows: Let R_4 be the label of $p \to q$, R_1 of $p \to q_{rip}$, R_2 of q_{rip} to q_{rip}, and R_3 of q_{rip} to q. The new label for $p \to q$ will get the label $(R_4) \cup (R_1)(R_2^*)(R_3)$. The meaning is: to get from p to q, you can either go directly using a string of the type R_4, or go via q_{rip} using a string of the type $(R_1)(R_2^*)R_3$. □

4 Non-regular languages

Consider the language $L = \{0^n 1^n \mid n \geq 0\}$. Intuitively, L cannot be accepted by any DFA since the DFA cannot count (as they have only a constant number of states). More formally, we will use the Pumping Lemma to argue that certain languages are not regular.

Lemma 1 (Pumping Lemma)**.** *Let L be any regular language, and let A be a DFA on p states accepting L, for some $p \geq 1$. Then every $w \in L$ with $|w| \geq p$ has the form $w = xyz$ such that*

- $|y| > 0$,
- $|xy| \leq p$, *and*
- $xy^i z \in L$, *for every* $i \geq 0$.

Proof. The idea is very simple. Consider any $w \in L$ whose length is $n \geq p$. During the accepting computation of DFA A on w, the automaton goes through a sequence of $n + 1 > p$ states $q_0, q_1, q_2, \ldots, q_n$. So, by the pigeonhole principle, there must be a repeated state in the sequence. Let q_i be the first occurrence of the repeated state, and let q_j be the second occurrence of that same state. Let x be the string that takes the DFA from q_0 to q_i; y be the string that takes it from q_i to q_j; and z be the string that takes it from q_j to q_n. It is easy to see that $|xy| \leq p$ (as the first repeated occurrence must happen within the first p transitions of the DFA having p states only). Also, y cannot be empty as q_i and q_j are in distinct time steps, and so take some non-empty string to go from one to the other. Finally, any repetition of y results in a string in L since the DFA can simply loop for a number of times, going from q_i to q_j. □

To argue that a given language is not regular, we assume that it were regular, then pick a long enough string w in the language, and show that the "pumped" version of w cannot be in the language. This contradicts the Pumping Lemma, and so the original language cannot be regular.

Specifically, take $L = \{0^n 1^n \mid n \geq 0\}$.

Claim 1. *L is not regular.*

Proof. Suppose L is regular. Let A be a DFA accepting L, and let p be the number of states of DFA A.

Consider $w = 0^p 1^p$. Since $|w| \geq p$, we should have by the Pumping Lemma that $w = xyz$, with y nonempty, $|xy| \leq p$, and $xy^i z \in L$ for every $i \geq 0$.

There are 3 cases:

1. y consists entirely of 0s. Then $xyyz$ cannot be in L as it has too many 0s.
2. y consists entirely of 1s. Then $xyyz$ cannot be in L as it has too many 1s.
3. y contains both 0s and 1s. Then $xyyz$ cannot be in L as it has 0s and 1s out of order.

This shows that w cannot be "pumped", which contradicts the Pumping Lemma. We conclude that L is not a regular language. (Actually, if we're careful, we'll see that cases (2) and (3) cannot happen: we know by the Pumping Lemma that $|xy| \leq p$, and so y will fall into the region of w that consists entirely of 0s. So we only need to consider case (1) above.) □