Lecture 4:
Summary of Regular Languages

Valentine Kabanets

September 28, 2016

1 Some decidable properties of DFAs

We consider three natural questions about DFAs (phrased as languages):

1. $A_{DFA} = \{\langle M, w\rangle \mid$ DFA M accepts string $w\}$.
2. $E_{DFA} = \{\langle M\rangle \mid$ DFA M does not accept any string, i.e., $L(M) = \emptyset\}$.
3. $EQ_{DFA} = \{\langle M_1, M_2\rangle \mid$ DFAs M_1 and M_2 decide the same language, i.e., $L(M_1) = L(M_2)\}$.

The notation $\langle A\rangle$ means some encoding of an object A (say in binary).

We'll argue that each of these three languages is decidable, i.e., there is an algorithm for each of these three languages.

1. To decide A_{DFA}, we need an algorithm that takes as input a (description of a) DFA M and a (description of a) string w, and simulates M on w. We can simulate a DFA on a given input very easily: just keep track of the current state of M, and the current position on the input tape, and make transitions to a new state by following the δ function of the DFA M. It is not hard to see how to implement this simulation as a linear-time algorithm (say in Java).

2. We want to know if a given DFA M accepts some string. Think about the transition diagram for the DFA M. It is a directed graph, with edges labeled with symbols of the alphabet. Observe that

   > M accepts some string if and only if there is a way to reach an accepting state from the initial state, using the edges of our transition-diagram graph.

   Thus, it suffices to solve the reachability problem for a certain directed graph! We know how to do it (in linear time) using, say, breadth-first search!

3. Observe that $L(M_1) = L(M_2)$ if and only if the symmetric difference of the two languages is empty, i.e., iff
   $$L := (L(M_1) - L(M_2)) \cup (L(M_2) - L(M_1)) = \emptyset.$$
   We will argue that for two regular languages A and B, their difference $A-B$ is also regular. To this end, we take two DFAs $M_A = (Q_A, \Sigma, \delta_A, q_0, F_A)$ and $M_B = (Q_B, \Sigma, \delta_B, p_0, F_B)$ deciding A and B, respectively, and build a new DFA $M = (Q, \Sigma, \delta, r_0, F)$ deciding the language $A-B$.

The construction is similar to the one for the union of two regular languages that we did in one of the early lectures for the case of two DFAs. There, we essentially simulated the two automata "in parallel" by keeping track of the states of each of them as we read a given input string w. The only difference is what we call an accepting state for the case of $A - B$. If a given string w takes DFA M_A to an accepting state, and takes DFA M_B to a non-accepting state, then we should accept since $w \in A$ and $w \notin B$. Otherwise, we reject.

Formally, we take $Q = Q_A \times Q_B$, $r_0 = (q_0, p_0)$, $\delta((q,p),a) = (\delta_A(q,a), \delta_B(p,a))$, and $F = F_A \times (Q_B - F_B)$.

Now that we've shown that the difference of two regular labguages is also regular, and since we already know that the union of two regular language is also regular, we conclude that the language L defined above (as the symmetric difference of two regular languages) is regular. Hence, there is a DFA C deciding this L. Moreover, we can construct this DFA C from DFAs M_1 and M_2.

Thus, we want to know if $L(C) = \emptyset$. But this is exactly the problem E_{DFA} considered earlier, for which we already have an algotihm!

2 Minimization of FA and regular expressions

For *DFA*, there is an efficient (polynomial-time) algorithm for finding a minimal equivalent DFA, i.e., an equivalent DFA with a minimal number of states. In contrast, for *NFA* and for *regular expressions*, the task of finding a minimal equivalent NFA (or a minimal equivalent regular expression) is not known to be efficiently solvable, and it's conjectured that no such algorithm exists. (These two minimization problems are PSPACE-complete, the concept we'll talk about later in the course.)

3 Summary of Regular Languages

- Finite automata are a special case of Turing machine, with very limited computational power, but useful for many simple tasks (e.g., in compiler design).
- DFA and NFA accept the same class of languages (albeit NFA can have fewer states than DFA for the same language).
- Regular expressions correspond exactly to regular languages.
- There are simple languages that are *not* regular. The Pumping Lemma is a useful tool for arguing that a language is not regular.

We will skip Context-Free Languages, as well as (Context-Free) Grammars; you will see those in the Compilers course.

Now back to Turing machines!