

CS 3719 (Theory of Computation and Algorithms) – Lecture 4

Antonina Kolokolova*

January 13, 2011

1 Regular Languages

Recall that a language is recognized by a finite automaton if each string in the language and no string not in the language are accepted by this automaton. There can be several different automata accepting the same language.

Definition 3. *A language is called regular if there exists a finite automaton that accepts it. We will use the notation $\mathcal{L}(N)$ to mean the language accepted by the automaton N .*

We will show soon that this is the same “regular” as in “regular expressions”. In particular, the operations used to build regular expressions inductively out of smaller ones are called regular operations:

Definition 4. *Let A, B be languages. We define three regular operations on them:*

- **Union** $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Concatenations** $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
- **Star** $A^* = \{x_1 \dots x_k \mid k \geq 0 \text{ and } \forall 1 \leq i \leq k, x_i \in A\}$

We are building up to showing that the class of languages expressible by regular expressions is the same as the class of languages accepted by finite automata: that is, in both cases it is the class of regular languages.

Theorem 1. *The class of regular languages is closed under regular operations.*

*The material in this set of notes came from many sources, in particular “Introduction to Theory of Computation” by Sipser and course notes of U. of Toronto CS 364.

Corollary 2. *For every regular expression, there is a finite automaton that recognizes the same language as this regular expression generates.*

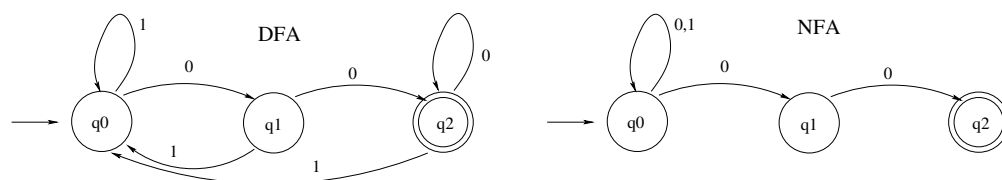
We will postpone the proof of this theorem and the corollary slightly, to introduce a version of finite automata which will make the proofs easier. This version looks more powerful; however, we will show that the class of languages recognized by this model is the same as the class of languages recognized by automata we considered so far (although not efficiently).

Definition 5. *A non-deterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0 and F are as in the case of deterministic finite automaton (Q is the set of states, Σ an alphabet, q_0 is a start state and F is a set of accepting states), but the transition function δ is $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q)$. Here, $\mathcal{P}(Q)$ is the powerset (set of all subsets) of Q .*

A non-deterministic finite automaton accepts a string $w = w_1 \dots w_m$ if there exists a sequence of states r_0, \dots, r_m such that $r_0 = q_0, r_m \in F$ and $\forall i, 0 \leq i < m, r_{i+1} \in \delta(r_i, w_i)$.

There are two differences between this definition of δ and the deterministic case. First, δ gives a (possibly empty) set of states, as opposed to exactly one state. Second, there can be a transition without seeing any symbols, on empty string ϵ . Also, now the acceptance condition is that there exists a good sequence of states leading to acceptance: there can be many computational paths, some of which would lead to rejecting states, some would “die” with no next state to go to, and, if the string is in the language, at least one will finish in an accepting state.

Example 1. Consider the following two automata accepting a language of all strings ending with 00. The first one is deterministic, the second non-deterministic. Notice how in the non-deterministic automaton there are two arrows on 0 from q_0 (so, $\delta(q_0, 0) = \{q_0, q_1\}$), no arrows on 1 from q_2 and no arrows at all from q_2 (so $\delta(q_2, 0) = \delta(q_2, 1) = \emptyset$.) This example does not have ϵ -arrows.

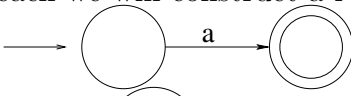
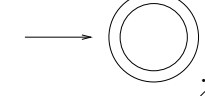



Consider possible executions of this NFA on string 101000. From q_0 on 1 there is just one choice: stay at q_0 . On the second symbol, there are two choices: stay at q_0 or go to q_1 . The computational branch that goes to q_1 dies at the next step: there is nowhere to go from q_1 on 1. The first computational branch survives, there the automaton stays in q_0 on the 3rd symbol. On the 4th symbol, 0, there is a choice again: you can see that moving to q_1 and then to q_2 will make it get stuck on the last 0. Similarly, not going to q_1 on the 5th symbol will make it finish in a reject state (either q_0 or q_1). Finally, an accepting sequence of states for string 101000 is: $q_0, q_0, q_0, q_0, q_0, q_1, q_2$.

Now, with this definition of NFA, we are ready to prove the Corollary 2.

Proof. Recall the recursive definition of regular expressions: there were three base cases, and for the recursive step the three rules were union, concatenation and star. The theorem 1 (which we will prove later) gives us the recursive step. More precisely, if regular expressions R_1 and R_2 generate the same languages as recognized by finite automata N_1 and N_2 , respectively, then the closure under union operation gives us an automaton for a language $\mathcal{L}(N_1) \cup \mathcal{L}(N_2)$, which is the same set of strings as $R_1 \cup R_2$. Similarly, the theorem gives us automata for recognizing $R_1 \circ R_2$ and R_1^* given automata recognizing R_1 and R_2 . So all that is left to prove is the base case.

There are three parts of the base case, and for each we will construct a NFA accepting it.

- 1) $R = a$, for some $a \in \Sigma$. Take the NFA 
- 2) $R = \epsilon$ Define the corresponding NFA as 
- 3) $R = \emptyset$ Then, take an NFA accepting nothing. 

The proof follows by structural induction. Formally, suppose R is a regular expression; we will show how to construct a NFA recognizing the same language. If R is of the form a or ϵ or \emptyset then use the corresponding automaton from the previous paragraph. Otherwise $R = R_1 \cup R_2$ or $R = R_1 \circ R_2$ or $R = R_1^*$. By induction hypothesis, there exist NFAs N_1 and N_2 recognizing languages of R_1 and R_2 respectively. Now, by the theorem 1 there is an NFA N with $\mathcal{L}(N) = \mathcal{L}(N_1) \cup \mathcal{L}(N_2)$. Thus, a regular expression $R = R_1 \cup R_2$ generates a language $\mathcal{L}(N)$. The arguments for $R_1 \circ R_2$ and R_1^* similarly follow from theorem 1. \square

To give an intuition for the proof of theorem 1 for the case of union, as well as illustrate the use of ϵ -arrows, consider the following example.

Example 2. Consider the language of strings over $\{0, 1\}$ starting with either 00 or 11. Here, the ϵ -arrows at the beginning allow us to start either at the automaton recognizing strings starting with 00 or at the automaton recognizing strings starting with 11.

