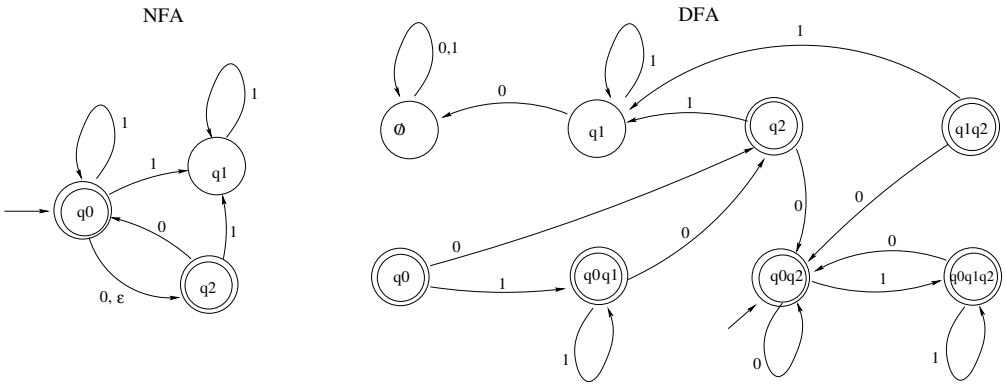
CS 3719 (Theory of Computation and Algorithms) – Lecture 7

Antonina Kolokolova*

January 20, 2011

Example 1. Here is an example of converting a 3-state NFA into a DFA, following the procedure from the last class. The alphabet is $\Sigma = \{0, 1\}$. The states of the DFA are $Q = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$. Out of them, all except $\emptyset$ and $\{q_1\}$ are accepting states. The starting state is $\{q_0, q_2\}$.



1 From automata to regular expressions

In this lecture, we will show the other direction of a theorem that the class of languages recognized by automata is the same as the class generated by regular expressions. Moreover, our proof will be constructive: given an automaton, we will show how to convert it into an equivalent regular expression.

Theorem 5. *For every NFA N there is a regular expression R generating $\mathcal{L}(N)$.*

Before we start the proof, let us play with the definition of an automaton a little more. It would be very convenient for the proof if the transitions in the automaton were regular

*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.

expressions: after all, a symbol is a regular expression, so is ϵ, a "0,1" is just $(0 \cup 1)$ and no arrow is like an $\emptyset$ expression. Also, let's have a unique accepting state and make it and the start state special: there will be no incoming arrows into the start state and no outgoing arrows to the accept state. We will call such automata Generalized Nondeterministic Finite Automata (GNFA).

Definition 6. A Generalized Nondeterministic Finite Automata (GNFA) $N = (Q, \Sigma, \delta, q_{start}, q_{accept})$ *is a 5-tuple where Q is the set of states, Σ an alphabet, q_{start} a start state, q_{accept} a (unique) accept state and $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \to \mathcal{R}$ a transition function, where $\mathcal{R}$ is the set of all regular expressions over Σ. A GNFA accepts a string w if there is a way to break w into $w_1 \dots w_{m'}$ (some possibly empty) in Σ^*, where there exists a sequence of states $r_0, \dots, r_{m'}$ starting with q_{start} and ending with q_{accept} such that $\forall i, 0 \leq i < m', w_i \in \mathcal{L}(\delta(r_i, r_{i+1}))$.*

Note that here the transition function, rather than outputting a state, outputs a label of the transition. The reason for defining it this way is that there is just one transition (possibly labeled $\emptyset$) for every 2 states for the total of $n^2 + 2n$ transitions in a GNFA with n internal states; however there are infinitely many possible regular expressions. In our definition, δ can still be described by a finite table.

Lemma 6. *Any DFA can be converted into a GNFA.*

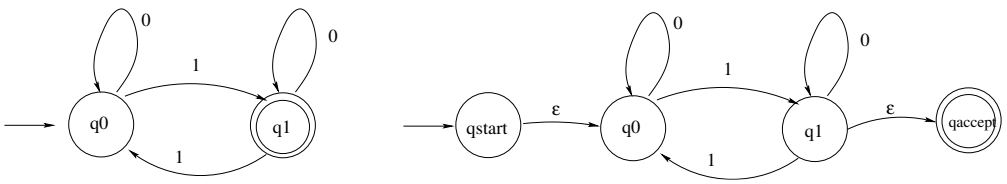
Proof. To convert a DFA $D = (Q, \Sigma, \delta, q_0, F)$ into a GNFA $N = (Q', \Sigma, \delta', q_{start}, q_{accept}$, add two states q_{accept} and q_{start} to Q (getting Q'). The new transition function δ' will consist of transitions in δ in the new format together with transitions for the two new states.

$$\delta'(q_i, q_j) = \begin{cases} \bigcup_k \{a_k\}, & \{a_k \in \Sigma \cup \{\epsilon\} | \delta(q_i, a_k) = q_j\} \\ \epsilon, & (q_i = q_{start} \text{ and } q_j = q_0) \text{ or } (q_i \in F \text{ and } q_j = q_{accept}) \\ \emptyset, & \text{otherwise} \end{cases}$$

The notation $\bigcup_k S_k$ means a union of sets S_k; here, $\bigcup_k \{a_k\}$ just means a set of all symbols of Σ on which there is a transition from q_i to q_j in D. If there are no such symbols, $\bigcup_k \{a_k\} = \emptyset$.

Now, N is a GNFA, and $\mathcal{L}(N) = \mathcal{L}(D)$. To check the latter statement, notice that N starts reading its input in the state q_0 since there are no transitions other than ϵ to q_0 from q_{start}, and that whenever the computation ends in one of the states in F, it has an ϵ-transition to q_{accept}.Since the two new states do not participate in the computation otherwise, for every computational path in D there is the same computational path in N preceded with q_{start} and followed by q_{accept} and vice versa. □

Example 2. Here a simple 2-state DFA accepting strings with odd number of 1s is converted into a corresponding GNFA.



Proof of theorem 5. Now we will show how, given a GNFA $N = (Q, \Sigma, \delta, q_{start}, q_{accept})$, to compute a corresponding regular expression. The proof is by induction on the number of states in N.

For the base case, there are only two states in N, and they are q_{start} and q_{accept}. There is just one transition in this NFA, $\delta(q_{start}, q_{accept}) = R$. Then, $\mathcal{L}(N) = \mathcal{L}(R)$. In the acceptance condition of the GNFA, $m' = 1$, $r_0 = q_{start}$, $r_1 = q_{accept}$ and $w \in \mathcal{L}(R)$.

Now suppose that we know how to obtain a regular expression from a GNFA with $k \geq 2$ states. We will show how to convert a GNFA $N = (Q, \Sigma, \delta, q_a, q_s)$ on $k+1$ states to a GNFA $N = (Q', \Sigma, \delta', q_a, q_s)$ on k states accepting the same language. Here, q_s and q_a are just a short notation for q_{start} and q_{accept}.

Pick an arbitrary state in $Q - \{q_s, q_a\}$; call it q_r ("q_{rip}" in Sipser's book). Set $Q' = Q - \{q_r\}$. Now, let q_i and q_j be two other states in Q (recall that there are at least 3 states in Q). Since we are removing the state q_r we need to adjust δ to take into account all computation paths that were going via q_r.

Look at two ways to get from q_i to q_j: via q_r or direct. The path via q_r consists of 3 steps: $\delta(q_i, q_r) = R_1$, then a possible loop on q_r, $\delta(q_r, q_r) = R_2$, and then q_r to q_j step on $\delta(q_r, q_j) = R_3$. A direct path from q_i to q_j is $\delta(q_i, q_j) = R_4$. Now, note that a path via q_r is a concatenation of the three steps with 0 or more repetitions of the loop; thus, a regular expression $R_1 R_2^* R_3$ encodes it. Finally, since there are two possible paths from q_i to q_j, via q_r or direct, we take the union of them. The label on the q_i, q_j arrow in the GNFA without q_r thus becomes $\delta'(q_i, q_j) = R_1 R_2^* R_3 \cup R_4$. You can check yourself that for any substring w_m on which N went from q_i to q_j either direct or via q_r, N' will also go from q_i to q_j, directly this time.

Therefore, by induction hypothesis, we can extract a regular expression from N', since $\mathcal{L}(N') = \mathcal{L}(N)$ and N' contains only k states.

In other words, to convert a GNFA to a regular expression repeat the procedue of removing a state and adjusting δ $k-2$ times. Then, return $\delta(q_a, q_s)$ of the resulting 2-state GNFA.

Now, starting from a finite automaton (assiming, without loss of generality, that it is a DFA) construct a GNFA and then obtain a regular expression by doing the removing-states procedure until only 2 states are left. □

Example 3. Here are the two steps of obtaining a regular expression from the GNFA in the previous example. Noting that $\epsilon R = R$, $\emptyset \cup R = R$ and $R\emptyset = \emptyset$, obtain $R = 0^*1(10^*1 \cup 0)^*$