Tik-79.148 **Spring 2001**
Introduction to Theoretical Computer Science
Tutorial 5
Answers to Demonstration Exercises

4. The construction works by going through all possible paths from the initial state to all final states. At start we give each state a unique number and then use the following two recursive rules:

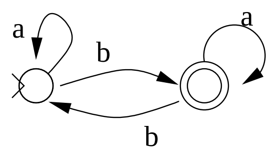
$$R(i,j,1) = \begin{cases} \{\sigma \in \Sigma \mid \delta(q_i,\sigma) = q_j\} & i \neq j \\ \{e\} \cup \{\sigma \in \Sigma \mid \delta(q_i,\sigma) = q_j\} & i = j \end{cases}$$
$$R(i,j,k+1) = R(i,j,k) \cup R(i,k,k)R(k,k,k)^* R(k,j,k)$$

The notation $R(i,j,k)$ denotes the set of computations that lead from the state q_i to q_j that do not visit state q_k or higher.

The intuition of the first rule is that if we may not visit any state on our route, we mush take a direct transition from q_i to q_j.

The second rule asserts that we can divide all routes from q_i to q_j without visiting q_{k+1} or higher into two partitions: we either do not visit q_k enroute or we go to q_k, potentially visit it again, and finally go to q_j.

We want to find the regular expression corresponding to the following finite state automaton:



Using the rules, we get:

$$\begin{aligned} L(M) &= R(1,2,3) \\ R(1,2,3) &= R(1,2,2) \cup R(1,2,2)R(2,2,2)^* R(2,2,2) \\ R(1,2,2) &= R(1,2,1) \cup R(1,1,1)R(1,1,1)^* R(1,2,1) \\ R(2,2,2) &= R(2,2,1) \cup R(2,1,1)R(1,1,1)^* R(1,2,1) \end{aligned}$$

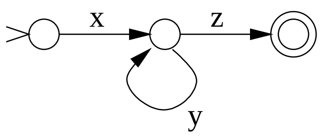
$$\begin{aligned} R(1,2,1) &= R(2,1,1) = b \\ R(1,1,1) &= R(2,2,1) = e \cup a \end{aligned}$$

Finally, we substitute the simple cases to the earlier equations to get:

$$\begin{aligned} R(1,2,2) &= b \cup (e \cup a)(e \cup a)^* b = a^* b \\ R(2,2,2) &= (e \cup a) \cup b(e \cup a)^* b = e \cup a \cup ba^* b \\ R(1,2,3) &= a^* b \cup a^* b(e \cup a \cup ba^* b)^* (e \cup a \cup ba^* b) \\ &= a^* b(a \cup ba^* b)^* \end{aligned}$$

5. The pumping theorem for regular languages states that for each infinite regular language L we can find some $k \geq 0$ such that all strings w in the language that are longer than that ($|w| > k$) may be divided into three parts x, y, and z ($y \neq e$) such that $xy^nz \in L$ for all $n \geq 0$.

   The intuition behind the pumping theorem is that there is only a finite number states in an automaton. If the accepted language is infinite, there has to be a cycle in the automaton. This cycle may be traversed zero, one, or an arbitrary number of times. The following picture illustrates this situation:

   We have to prove that $L = \{ww^R \mid w \in \{a, b\}^*\}$ is not regular. We start by defining the language $L' = L \cap (ab)^*(ba)^*$. Since the class of regular languages is closed under intersection, L may not be regular if L' is not regular.

   By examining the words in L we notice that:

   $$L' = (ab)^n(ba)^n, n \geq 0 \ .$$

   Consider the word $w = (ab)^k(ba)^k$ where k is an arbitrarily large integer. If L' is regular, we can divide w in the three parts x, y, and z. We now go through every possible choice of y and show that the pumping theorem cannot be satisfied so L' has to be irregular.

   1° $y = (ab)^r$ or $y = (ba)^r, r > 0$. Since all words in L' have an equal number of ab and ba parts, the words resulting from adding y don't belong to L' anymore.

   2° $y = bb, y = a(ba)^r$ tai $(ba)^r b, r \geq 0$. Since all words in L' have an equal number of a and b letters, these choices are not possible.

   3° $y = a(ba)^r bb(ab)^r b$ is not possible since in this case the resulting words will have a substring ba before the last ab.

   Since no satisfying partition could be found, L' is not regular so L must also be irregular.

6. We can use the pumping theorem to prove that a language is *not* regular but not the other way. There are languages that fullfill the conditions of the theorem but that are not regular. For example, the language of balanced parenthesis (for example, (()()) and (((()))) belong to it) is not regular but the condition is satisfied since we may add the string () an arbitrary number of times to all words of the language.

   The easiest way to prove that a language is regular is to use the closure properties of regular languages; the class of regular languages is closed under the union, concatenation, Kleene star, complementation, and the intersection.

In the language we are given a regular language L and we define a new language by it:

$$L' = \{xy \mid x \in L \text{ ja } y \notin L\}$$

The language L' is the concatenation of languages L and its complement $\overline{L}$ ($y \notin L \rightarrow y \in \overline{L}$). Since regular languages are closed under complementation and concatenation, L' is regular.

We may also construct a finite-state automaton that decides the language L'. Since L is regular, there is some deterministic automaton M that decides it. We now construct an automaton $\overline{M}$ that is otherwise similar to M except that all accepting states are made rejecting and vice verse. Now $\overline{M}$ accepts the complement of L. We combine these two machines into a new non-deterministic automaton M' by adding a non-deterministic e-transition from all accepting states of M to the initial state of $\overline{M}$. Now M' decides L' so L' is regular.