Theory of Computation - CSE 105, Fall 1998

Regular Languages

Sample Problems and Solutions

Notes: The alphabet in all problems is $\{0, 1\}$ unless explicitly mentioned otherwise.

1 Deterministic Finite Automata (DFA)

Problem 1 - Designing DFAs

Define the language L_1 to be

$$L_1 = \{w : w \text{ contains either } 000 \text{ or } 010 \text{ as a consecutive substring}\}$$

Give the state diagram of a DFA with *at most five states* that recognizes the language L_1, and give a *formal* proof of the correctness of your construction.

Strategy: We first try to determine what we need to remember about a string as we are reading it. In this case we would need the following states: a start state, a state in which we have just seen a 0. Then states that have just seen 00 and 01. And finally a state that has seen either a 000 or a 010. This will be our accept state. Note that it suffices to have one state represent both 000 and 010, since all that matters is that we need to accept both.

Construction: The following DFA recognizes L_1.

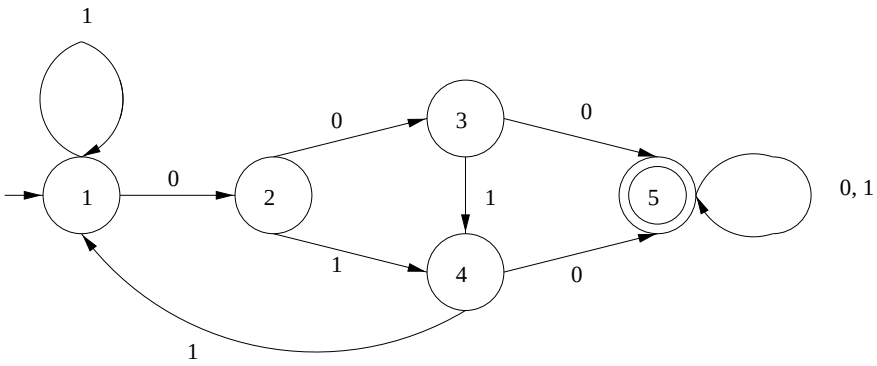


Figure 1: *A DFA recognizing* L_1

Correctness of Construction: We need to give a *formal* argument proving that the DFA constructed for L_1 accepts *all* and *only* those strings that are in L_1. The proof, thus, has two subproblems :

1. Prove that for any string x in L_1, x is accepted by the constructed DFA
2. Prove that for any string x not in L_1, x is not accepted by the constructed DFA

To get an idea for how to solve subproblem 1, recognize that we are trying to prove that a specific property (acceptance by the constructed DFA) holds for all elements of an infinite set (L_1). This is exactly the type of problem for which proof by induction is designed (see page 23 of the text). Subproblem 2 fits the same model as subproblem 1, since the complement of L_1 is also an infinite set. Hence, we can prove the construction correct by giving inductive proofs for subproblems 1 and 2.

Subproblem 1 - Proof by Induction In order to construct an inductive proof, we need to choose an appropriate variable to perform induction on. That is, we must find some function defined for all x in L_1 that maps L_1 to the set of integers. As every string in L_1 has a length which is an integer value, the *length* function is suitable. Hence, let l_x be the length of a string x. We shall perform induction on the variable l_x.

Basis *The constucted DFA accepts all x in L_1 such that $l_x <= 3$.*

When $l_x <= 2$, the basis claim is vacuously true since the definition of L_1 states that all elements contain either 000 or 010 as a substring, both of which have a length of $3 > 2$.

When l_x = 3, x must be either 000 or 010, as no other string of length 3 can possibly be in L_1. We need to show that the constructed DFA accepts both 000 and 010. From the start state, q_1, the string 000 moves the DFA along the following path of states : q_1 — q_2 — q_3 — q_5. Since q_5 is an accept state, the DFA accepts the string 000. Similarly, the string 010 causes transitions along the path : q_1 — q_2 — q_4 — q_5. Again, since q_5 is an accept state, the DFA accepts the string 010.

Having shown that the constructed DFA accepts all x in L_1 such that $l_x <= 3$, we may conclude that the basis of the induction is true.

Induction Step *Assume that the constucted DFA accepts all x in L_1 such that $l_x = k >= 3$. Then, the constructed DFA accepts all strings x in L_1 such that $l_x = k + 1$.*

Let x be any string in L_1 such that $l_x = k + 1$. Since $l_x > 2$, there exists some string y in $\sum^*$ such that either $x = y0$ or $x = y1$. The string y is either in L_1 or not in L_1.

If y is in L_1 then, by the inductive hypothesis, it is accepted by the constructed DFA. The strings $y0$ and $y1$ must also be accepted by the same DFA, since the DFA will be in accept state q_5 after processing y and since there are no transitions from q_5 to another state. Hence, x is accepted by the DFA whenever y is accepted by it.

If, on the other hand, y is not in L_1 then, since we assumed x is in L_1, there exists some string w in $\sum^*$ such that $x = y0$ and either $y0 = w000$ or $y0 = w010$. Processing the prefix string, w, may leave the DFA in any state (remember, we have not made any claims about the behavior of the DFA on strings that are not in L_1). Thus, we need to show that the accept state is reachable from any other state on input 000 or 010. There are 10 cases (5 DFA states X 2 choices for w), each of which the reader may easily verify leave the DFA in the accept state. It follows that x is accepted by the DFA whether or not y is in L_1.

Since, given x in L_1 such $l_x = k + 1$, we have shown that the inductive hypothesis implies that x is accepted by the constructed DFA, we may conclude that the constructed DFA accepts all x in L_1.

Subproblem 2 - Proof by Induction The inductive proof for this subproblem is very similar to the one given for Subproblem 1. It is left as an exercise.

Problem 2 - Designing DFAs

Give the state diagram of a DFA that recognizes the language L_2 where L_2 is given by

$$L_2 = \{w : w \text{ has } 3k + 1 \text{ 1's for some } k \in \mathcal{N}\}.$$

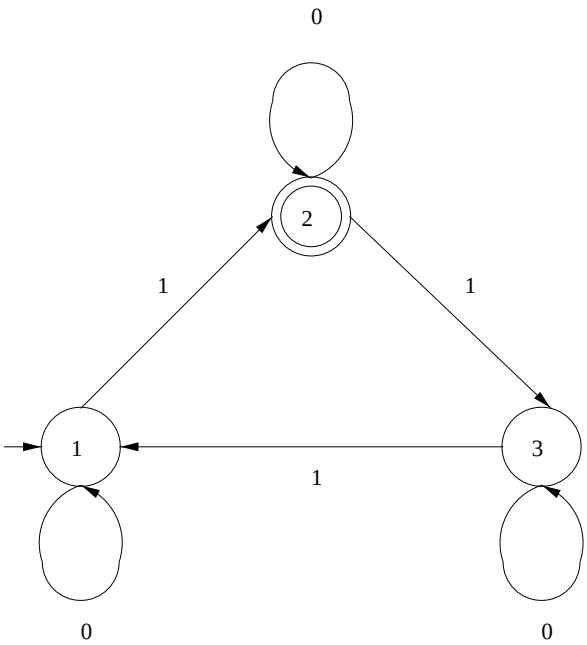


Figure 2: *A DFA recognizing L_2*

Problem 3 - Closure Properties of Regular Languages : Closure under Reversal

(Problem 1.24, page 88 of the text) For any string $x = x_1 x_2 \dots x_n$, the *reverse* of x, written $x^{\mathcal{R}}$, is the string x in reverse order, $x_n \dots x_2 x_1$. For any language A, let $A^{\mathcal{R}} = \{x^{\mathcal{R}} : x \in A\}$. Show that if A is regular, so is $A^{\mathcal{R}}$.

Given: A is regular. So there is a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that accepts A.

Want: To show that $A^{\mathcal{R}}$ is regular. We do this by constructing an NFA $M^{\mathcal{R}}$ that accepts $A^{\mathcal{R}}$.

Proof idea: The basic idea is that we want to modify the DFA M to "run backwards" on an input string of $A^{\mathcal{R}}$. We can do this by "reversing the transition arrows" in the original DFA. The original start state becomes the new final state. Similarly, we would like the original final states to become the new start states. However, since we are allowed only one start state, we need to add a new state which will also be our new start state with ϵ transitions to all the original final states.

Construction: Construct a machine $M' = (Q \cup s, \Sigma, \delta^{\mathcal{R}}, s, q_0)$, where $s \notin Q$ is a new start state we introduce. Define the new transition function such that for every $q \in Q$ and $\sigma \in \Sigma$,

$$\delta^{\mathcal{R}}(q, \sigma) = \{r \in Q : \delta(r, \sigma) = q\}$$

Also, let $\delta^{\mathcal{R}}(s, \epsilon) = F$. We claim that $M^{\mathcal{R}}$ is an NFA that accepts $A^{\mathcal{R}}$.

Correctness of Construction: Consider some $x = x_1 \dots x_n \in A^{\mathcal{R}}$. We are interested in the behaviour of $M^{\mathcal{R}}$ on x. By definition of $A^{\mathcal{R}}$ we have $x^{\mathcal{R}} = x_n \dots x_1 \in A$. Assume that $q_0 \dots q_n$ are the states in Q that are followed on input $x^{\mathcal{R}}$ in M. Then by our construction we have that $s, q_n \dots q_0$ is a path on input x in $M^{\mathcal{R}}$. Since this is a valid path ending in an accept state, we have that $M^{\mathcal{R}}$ accepts x.

Now, lets look at the behaviour of $M^{\mathcal{R}}$ on some $x = x_1 \dots x_n \notin A^{\mathcal{R}}$. We must show that there is no accepting path in $M^{\mathcal{R}}$ on input x. Again, by definition of $A^{\mathcal{R}}$, we have that $x^{\mathcal{R}} = x_n \dots x_1 \notin A$. We argue by noting that if there was an accepting path in $M^{\mathcal{R}}$ on input x then reversing it would yield an accepting path in M on input $x^{\mathcal{R}}$. However, since M does not accept $x^{\mathcal{R}}$, we know that there can be no accepting path in $M^{\mathcal{R}}$ on input x.

Problem 4 - Closure Properties of Regular Languages : Closure under $\oplus$

If A, B are languages, we define a new language

$$A \oplus B = \{x : x \in A \text{ or } x \in B \textit{ but not both}\}$$

Show that the class of regular languages is closed under this operation.

Given: A and B are regular.

Want: To show that $A \oplus B$ is regular.

Proof idea: While we can solve this via DFA or NFA constructions, it is considerably easier to do this by invoking only known closure properties of regular languages. The main trick here is to observe that

$$A \oplus B = (A \cup B) - (A \cap B) = (A \cup B) \cap (\overline{A \cap B})$$

Now we can use the fact that regular languages are closed under union, complementation and intersection to argue that $A \oplus B$ is regular.

Construction: Consider the following languages: $L_1 = (A \cup B)$, $L_2 = (A \cap B)$, $L_3 = \overline{L_2}$ and $L = L_1 \cap L_3$. We have $A \oplus B = L$. We claim that L is regular.

Correctness of Construction: Since A and B are regular and regular languages are closed under union, L_1 is regular. Since A and B are regular and regular languages are closed under intersection, L_2 is regular. Since L_2 is regular and regular languages are closed under complement, L_3 is regular. Since L_1 and L_3 are regular and regular languages are closed under intersection, L is regular. Since $L = A \oplus B$ we have that the class of regular languages is closed under the $\oplus$ operation.

Problem 5 - Closure Properties of Regular Languages: Closure under Hamming Distance

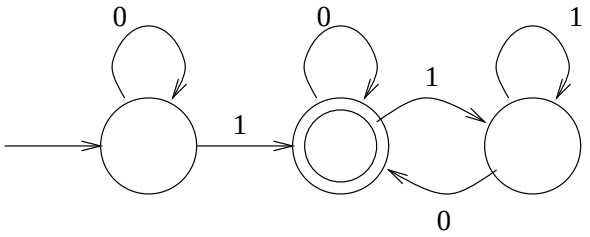
For any set $A \subseteq \{0,1\}^*$ and $k \leq 0$, define

$$N_k(A) = \{x | H(x, A) \leq k\},$$

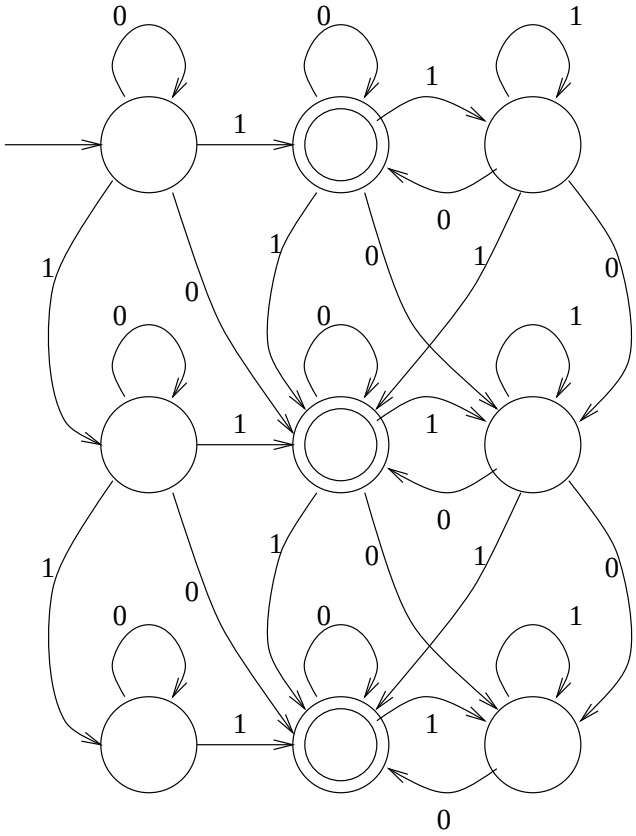
the set of strings of Hamming distance at most k from A. Prove that if $A \subseteq \{0,1\}^*$ is regular, then so is $N_2(A)$.

Main idea: If A is regular, then there is a deterministic finite automaton M that recognizes A. To show that $N_2(A)$ is regular, then, all we have to do is show that we can use M to build a new machine M' that recognizes $N_2(A)$. The general idea is to take the DFA M and transform it into an NFA that *allows the option of flipping up to two input bits* but otherwise processes an input string in exactly the same way as M does. This can be done by having M' contain three copies of M with the addition of transitions from copy 1 to copy 2 allowing the flipping of one bit and the addition of transitions from copy 2 to copy 3 allowing the flipping of one other bit. Then, in addition to accepting all strings accepted by M, the new machine will also accept all strings in A with either 1 or 2 bits flipped.

Simple example: Suppose M is the following machine:



Let the language recognized by this machine be called A. Then the following machine will accept $N_2(A)$:



This machine accepts the same strings as the previous machine, but it allows up to two bits in the input string to be flipped.

The formal construction: Given $M = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{s_1, ..., s_n\}$,

- Make three copies of M, named M_1, M_2, and M_3, where $M_i = (Q_i, \Sigma, \delta_i, q_{i,0}, F_i)$ is created by replacing each state s_j of M with $s_{i,j}$.
- Create the new machine M' that combines them so that $M' = (Q', \Sigma, \delta', q_0', F')$, where $Q = Q_1 \cup Q_2 \cup Q_3, q_0' = q_{1,0}$, and $F' = F_1 \cup F_2 \cup F_3$. First just copy the transition function: $\delta'(s_{i,j}, \sigma) = \delta_i(s_{i,j}, \sigma)$ (e.g. if M_1 contains the transition $\delta_1(q_{1,5}, 0) = q_{1,10}$, add the transition $\delta'(q_{1,5}, 0) = q_{1,10}$ to M').
- Add the transitions necessary to allow bit flipping as follows. For each transition $\delta'(s_{1,j}, 0) = s_{1,k}$, add a new transition $\delta'(s_{1,j}, 1) = s_{2,k}$. Likewise, for each transition $\delta'(s_{1,j}, 1) = s_{1,k}$, add a new transition $\delta'(s_{1,j}, 0) = s_{2,k}$.
- Do the same addition of transitions from machine 2 to machine 3, i.e. for each transition $\delta'(s_{2,j}, 0) = s_{2,k}$, add a new transition $\delta'(s_{2,j}, 1) = s_{3,k}$; for each transition $\delta'(s_{2,j}, 1) = s_{2,k}$, add a new transition $\delta'(s_{2,j}, 0) = s_{3,k}$.

Proof of correctness: To show that for a DFA M that recognizes A, the new NFA M' recognizes $N_2(A)$, we must argue that if $x \in N_2(A)$, M' accepts x, and if $x \notin N_2(A)$, M' does not accept x.

- If $x \in N_2(A)$, then $\exists y \in A$ such that $H(x, y) \leq 2$. If $H(x, y) = 0$ then $x \in A$, and the M_1 part of M' will accept x since M_1 contains all the transitions and accept states of M. If $H(x, y) = 1$, then x differs from $y \in A$ in only one position, so M_1 can process x up until it reaches the symbol at which x differs from y, switch to M_2, and continue processing the rest of x as if it was y, finally accepting it. The same argument holds if $H(x, y) = 2$, except the machine can move into an M_2 state at the first point at which x differs from y, then into an M_3 state at the other point at which x differs from y.
- If $x \notin N_2(A)$, then $\forall y \in A, H(x, y) > 2$. That is, more than two bits in x *must* be flipped to get a string A. If this is the case, M' cannot possibly accept x, since it allows at most two deviations from how the original machine M processes it.

2 Nondeterministic Finite Automata (NFA)

Problem 6 - Designing NFAs

Give the state diagram of an NFA with *at most four states* that recognizes the language L_1, where

$$L_1 = \{w : w \text{ contains either } 000 \text{ or } 010 \text{ as a consecutive substring}\}$$

Strategy: In this problem we show how one can use nondeterminism to save states. We need to make sure that there is at least one accepting path for all the strings we intend to accept. And that there is no accepting path for a string which is not in the language. Since we are interested in the presence of a 000 or 010 as a substring, we can basically ignore a preamble of 0's and 1's that does not contain sequences we are interested in. This is shown by the self loop in the first state. We need a state to maintain information that we have just seen at least one 0. Another is needed to record that we have just seen at least one occurence of a 00 or a 01. In some sense it was possible here to we collapse into one the two separate states used in the corresponding DFA to record this information. Finally we need an accepting state that records that we have found at least one occurence of a 000 or a 010.

Construction: The following NFA accepts L_1.

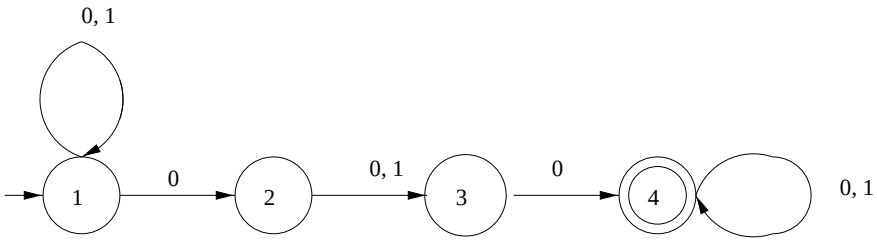


Figure 3: *An NFA recognizing L_1*

Correctness of Construction: We can informally prove the correctness of our construction by observing that sandwiched between the start and final states that allow any combinations of 0's and 1's is a path that can be traversed on processing either a 000 or a 010. Hence there is clearly at least one path from the start to the final state if the input string contains the desired substrings. To show that no string other than the ones that have the desired property are accepted, we argue that the only way to get from the start state to the final state is a path that can be traversed if only 000 or 010 appear as a substring.

Problem 7 - Converting an NFA to an equivalent DFA

Problem 1.12b, page 85 of the text.

Refer to Theorem 1.19, page 55 of text, and Example 1.21, page 57 of text.

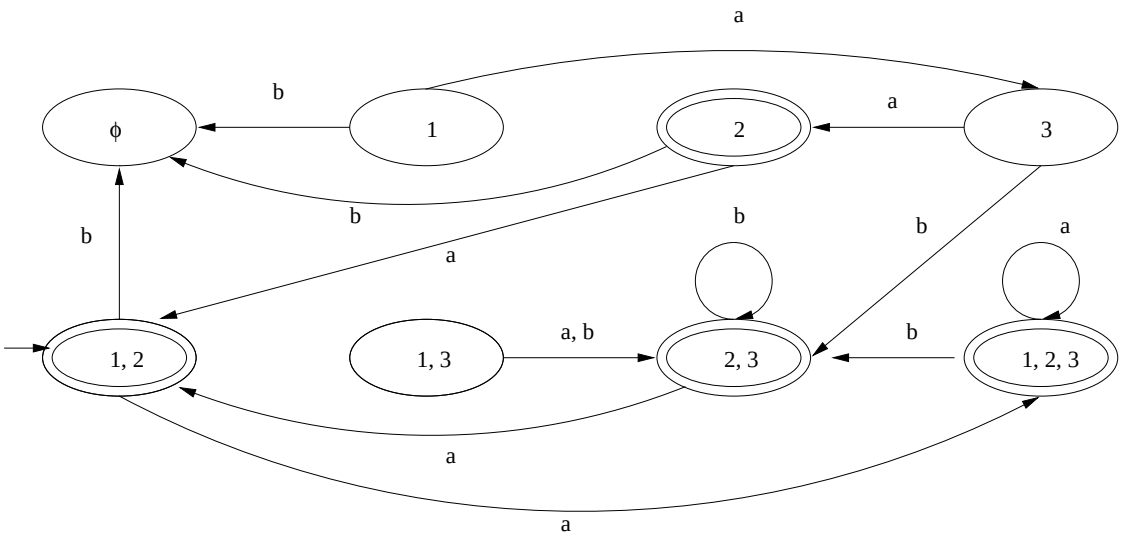


Figure 4: *A DFA that is equivalent to the given NFA*

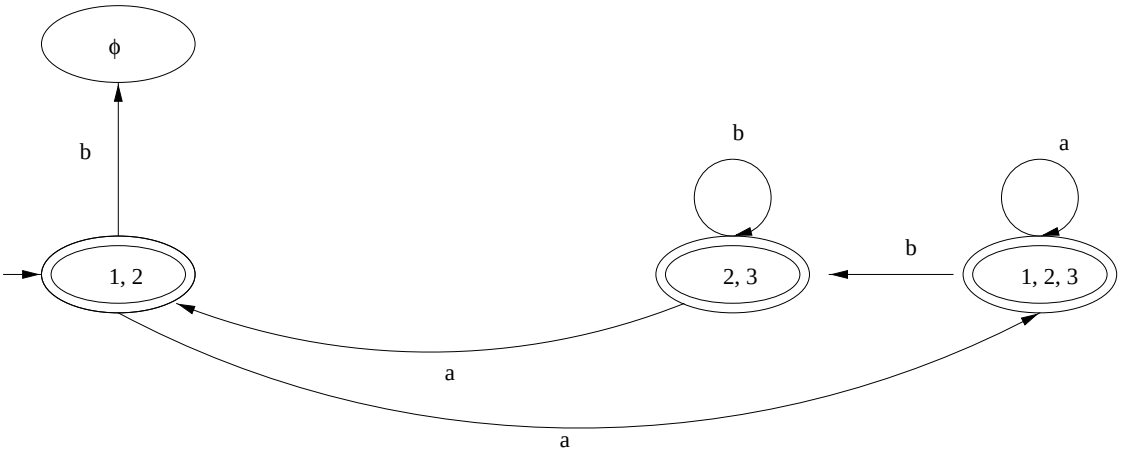


Figure 5: *The DFA above after removing unnecessary states*

3 Regular Expressions (RE)

Problem 8 - Designing Regular Expressions

Give the regular expression representing the language L_1 in Problem 1.

It is easy to see what the expected RE is by looking at the NFA we designed earlier. The regular expression is $(0 \cup 1)^*(000 \cup 010)(0 \cup 1)^*$.

Problem 9 - Designing Regular Expressions

Give the regular expression representing the language L_2 in Problem 2.

We first characterize the strings that are in L_2. These strings have one more 1 than some multiple of 3. That is all strings with just one 1, with four 1s, ..., and so on, are in L_2.

The regular expression is $0^*1(0^*10^*10^*1)^*0^*$.

Note: There are several possible correct solutions to such problems. For example another possible solution would be $0^*10^* \cup (0^*10^*1)(0 \cup \epsilon \cup 10^*10^*1)^*(10^*10^*)$.

Problem 10 - Converting NFAs to REs

Problem 1.16a, page 86 of the text.

Refer to Lemma 1.29, page 66 of text, and Example 1.30, page 68 of text.

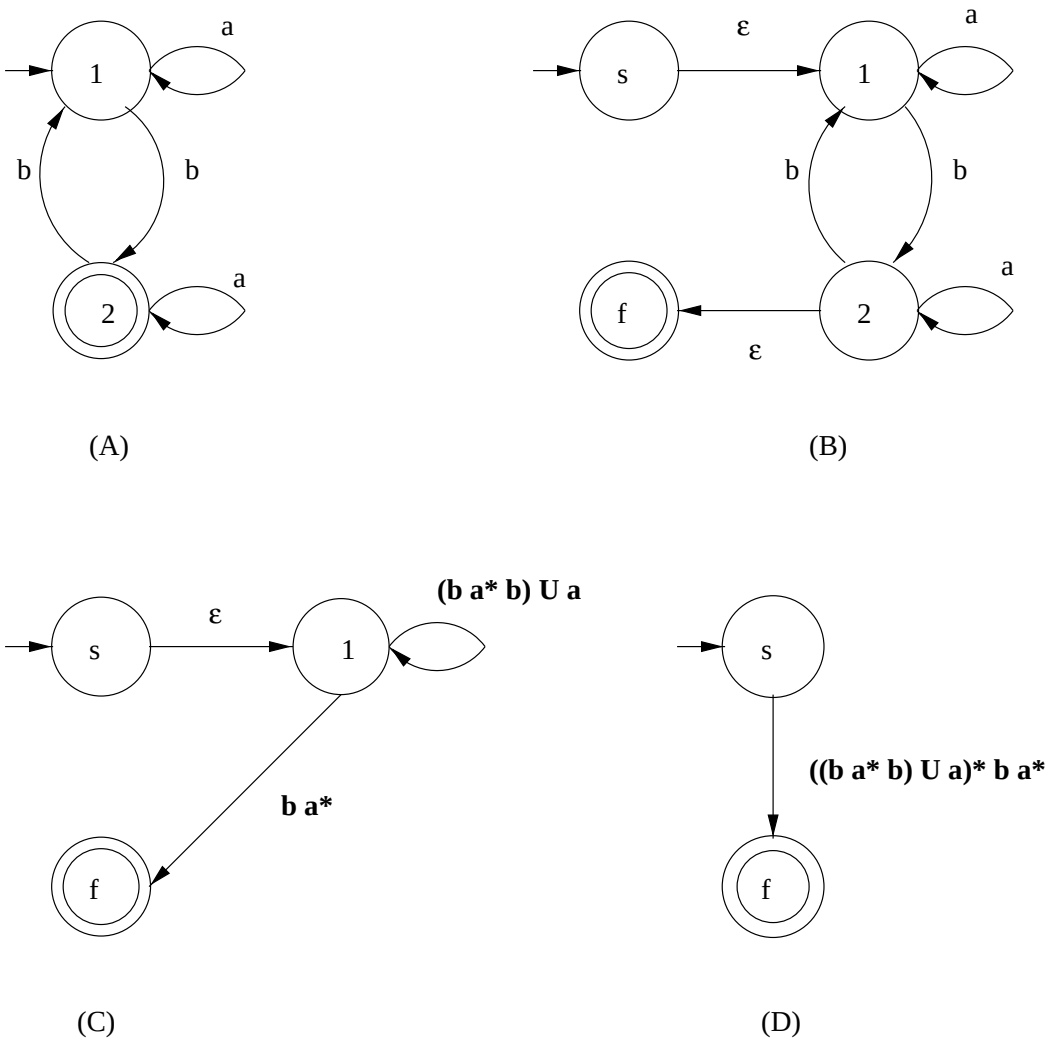


Figure 6: *The steps in converting the NFA given in (A) to a RE (transition label of (D))*

Problem 11 - Converting REs to NFAs

Problem 1.14c, page 86 of the text.

Refer to Lemma 1.32, page 69 of text, and Example 1.35, page 74 of text.

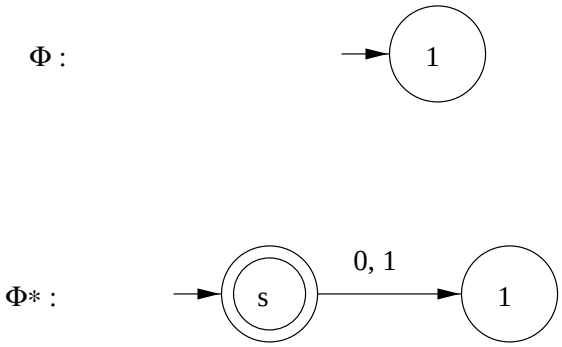


Figure 7: *Building an NFA from the RE* Φ^*

4 Non-regular Languages

Problem 12 - Application of Pumping Lemma

Problem 1.16a, page 86 of the text. Prove that the following language is not regular:

$$L_3 = \{w : w \in \{0,1\} \text{ is not a palindrome}\}$$

Proof: Consider the complement of the language L_3.

$$\overline{L_3} = \{w : w \in \{0,1\} \text{ is a palindrome}\}$$

We will prove that the language $\overline{L_3}$ is not regular. We can then use the fact that regular languages are closed under complementation to argue that L_3 cannot be regular either.
Let $A = \overline{L_3}$ and assume towards a contradiction that A is regular. We will use the pumping lemma to derive a contradiction.

Refer to Theorem 1.37, page 78 of the text. Let p be the "critical length" given by the theorem. Let us choose the string $s = 0^p1^p1^p0^p$. This string is, by definition of the language, contained in the language. Now since $s \in A$ and clearly $|s| > p$, the theorem says that it may be written as $s = xyz$ such that the three conditions of the theorem hold. Condition 3 says that $|xy| \leq p$. Since $s = 0^p1^p1^p0^p = xyz$, it must be that $xy = 0^k$ and $z = 0^{p-k}1^p1^p0^p$ for some $k \leq p$. Say $x = 0^{k-m}$ and $y = 0^m$. Condition 1 of the theorem holds for any $i \geq 0$. We choose $i = 2$. Now consider the string $s' = xy^2z$. This is $s' = 0^{k-m}0^{2m}z = 0^{k+m}0^{p-k}1^p1^p0^p$, and condition 1 says it is in A. But condition 2 of the theorem tells us $|y| > 0$, meaning $m > 0$. But then it is clear that s' is in fact not in A (i.e. s' is not a palindrome), because $p + m > p$ and s' will have more leading 0's than trailing 0's. This is a contradiction. Hence it must be that $\overline{L_3}$ is not regular.

Now consider the language L_3. Assume towards a contradiction that L_3 is regular. Since regular languages are closed under complementation, this would mean that $\overline{L_3}$ is also regular. In view of the result established earlier, this again is a contradiction. Hence it must be that L_3 is not regular.

Note: There are several ways of using the pumping lemma in proving the above. The trick is in choosing an initial string $s = xyz$ which makes our job easier.

Problem 13

Problem 1.28, page 89 of the text.

Proof: In order to show that language E is not regular, we're going to assume E is regular and then use the pumping lemma to show a contradiction. To do so, we first have to pick some string $s \in E$ whose length is at least p, the pumping length. This string should not be trivial, however. For example, if we pick $s = [0/0]^p$, we won't find any contradiction since the pumping lemma is going to work. So, let s be $[0/1]^p[1/0]^p$. Since $s \in E$ and $|s| \geq p$, s can be broken into three parts, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0, xy^iz \in E$;
2. $|y| > 0$; and
3. $|xy| \leq p$.

Because $|xy| \leq p$ and $|y| > 0$, $y = [0/1]^j$ for some $0 < j \leq p$. Then, when we pump up once ($i = 2$), we have $xy^2z = [0/1]^{p+j}[1/0]^p$ which is clearly not in E. Therefore, we have a contradiction and our initial assumption that E was regular was wrong.

6 Miscellaneous Problems

Problem 14 - All paths NFA

Problem 1.31, page 89 of the text.

We first characterize what it means for an all-paths-NFA to accept and reject. If the input is in the language, we require that *all* paths that may be followed end in accepting states. On the other hand, if the input is not in the language, we require that there exist *at least one* path that leads to a rejecting state.

There are two directions to this proof. We start with the easier one of proving that if L is a regular language then there exists an all-paths-NFA that accepts it.

Given: L is regular. So there is a DFA M that accepts L.

Want: To construct an all-paths-NFA M' that accepts L.

Construction: We claim that $M' = M$.

Correctness of Construction: Here we make the observation that the DFA conforms to the defintion of the all-paths-NFA. That is when the input is in L, there is only one possible path that accepts in a DFA for L, and hence we can say that all possible paths are accepting. And when the input is not in L, there is only one possible path that rejects, and hence we can say that there exists at least one path that rejects.

Now for the other direction,

Given: L is a language accepted by some all-paths-NFA.

Want: To show that L is regular.

Proof idea: We will construct an NFA that accepts $\overline{L}$. This would mean that $\overline{L}$ is regular. We know that the class of regular languages is closed under complementation. Thus we will have proved that L is regular.

Construction: Construct the machine M' which is the same as M except that the roles of accepting and rejecting states have been reversed in M'. That is all accepting states in M are rejecting states in M' and vice-versa. We claim that M' is an NFA that accepts the language $\overline{L}$.

Correctness of construction: Consider some $x \in \overline{L}$. We want M' to accept x. We have $x \notin L$. This means that on input x there is some path in M which ends in a reject state. This in turn implies that there exists some path in M' for input x which ends in an accept state and hence M' accepts x.
Now let us look at some $x \notin \overline{L}$. We want M' to reject x. Again we have $x \in L$. This means that on input x all paths in M end in accept states. This implies that on input x all paths in M' would end in reject states and thus M' rejects x.