Master Degree in Computer Engineering

Final Exam for
Automata, Languages and Computation

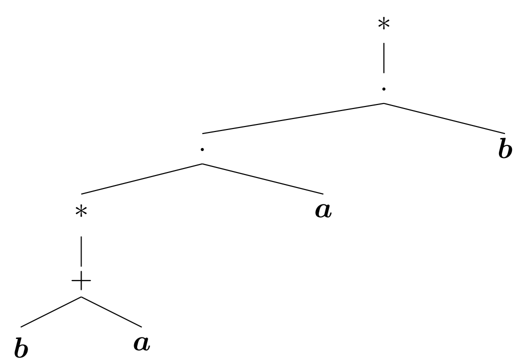
January 19th, 2026

1. **[4 points]** Consider the regular expression $R = ((\boldsymbol{a} + \boldsymbol{b})^*\boldsymbol{ab})^*$. Convert R into an equivalent ϵ-NFA using the construction provided in the textbook, and report all the **intermediate steps**.
Important: do not use any other construction different from the one presented in the textbook, and do not simplify the regular expression R before applying the construction.

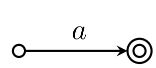
Solution

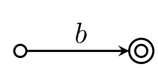
The construction to convert a regular expression into an equivalent ϵ-NFA is presented in Theorem 3.7 from Chapter 3 of the textbook. The construction must be applied using structural induction, that is, it must be applied to all the subexpressions of the input regular expression. For a subexpression S of R, we write $\gamma(S)$ to represent its conversion into an equivalent ϵ-NFA.

We first need to parse R into a tree representing its internal structure and all of its subexpressions. According to the recursive definition of regular expression, R can be associated with the following tree (we use the left-associative property of the concatenation operator, and we ignore the round brackets):

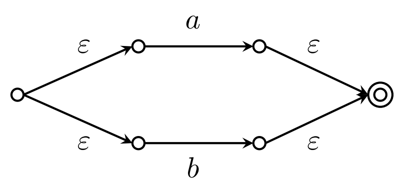


For the base case, we convert the regular expressions $\boldsymbol{a}$ and $\boldsymbol{b}$, resulting in the following automata (we do not annotate the start states, since these are always the leftmost states in the graph representation):

$\gamma(\boldsymbol{a})$: 

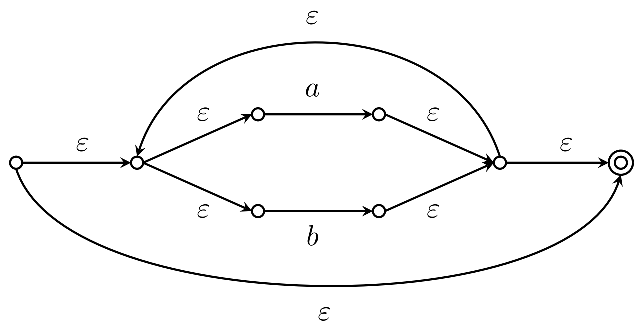
$\gamma(\boldsymbol{b})$: 

Next, we use $\gamma(\boldsymbol{a})$ and $\gamma(\boldsymbol{b})$ to convert the regular expression $\boldsymbol{a}+\boldsymbol{b}$, resulting in the following automaton:

$\gamma(\boldsymbol{a}+\boldsymbol{b})$: 

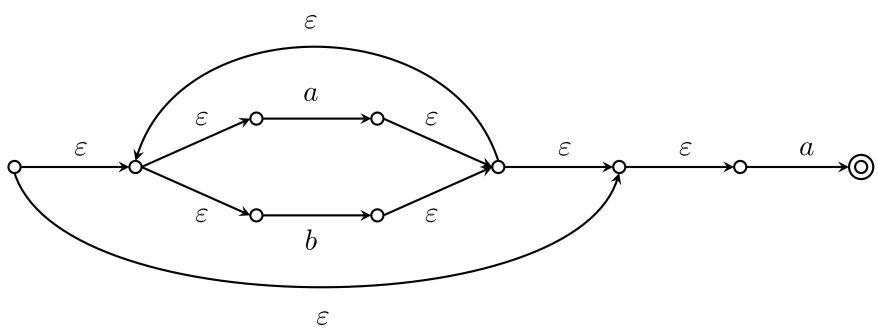
We can now process the innermost Kleen-star operator and convert the regular expression $(\boldsymbol{a}+\boldsymbol{b})^*$. We use the automaton $\gamma(\boldsymbol{a}+\boldsymbol{b})$, resulting in the following automaton:

$\gamma((\boldsymbol{a}+\boldsymbol{b})^*)$:



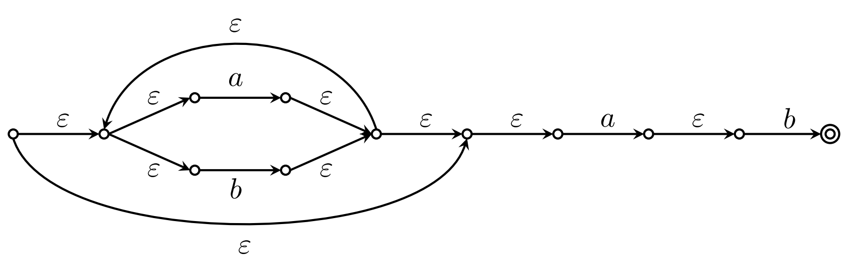
To convert the regular expression $(\boldsymbol{a}+\boldsymbol{b})^*\boldsymbol{a}$ we use the automata $\gamma((\boldsymbol{a}+\boldsymbol{b})^*)$ and $\gamma(\boldsymbol{a})$, resulting in the following automaton:

$\gamma((\boldsymbol{a}+\boldsymbol{b})^*\boldsymbol{a})$:

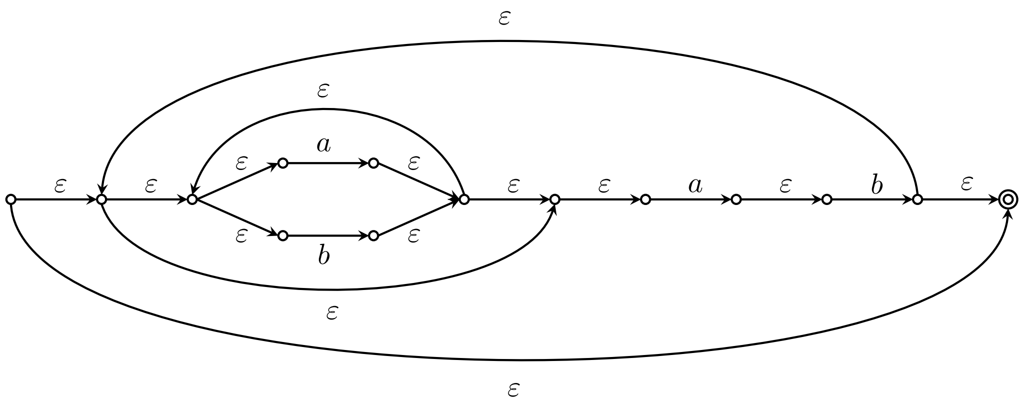


Similary, to convert the regular expression $(\boldsymbol{a}+\boldsymbol{b})^*\boldsymbol{ab}$ we use the automata $\gamma((\boldsymbol{a}+\boldsymbol{b})^*\boldsymbol{a})$ and $\gamma(\boldsymbol{b})$, resulting in the following automaton:

$\gamma((\boldsymbol{a}+\boldsymbol{b})^*\boldsymbol{ab})$:



The last step processes the outermost Kleen-star operator, using the automaton $\gamma((\boldsymbol{a}+\boldsymbol{b})^*\boldsymbol{ab})$ and producing the desired ϵ-NFA for our input regular expression R:



2. **[9 points]** Consider the following languages, defined over the alphabet $\Sigma = \{a, b\}$:

$$\begin{aligned} L_1 &= \{a^n b a^n \mid n \geq 1\} \\ L_2 &= \{a^n (ba)^n \mid n \geq 1\} \\ L_3 &= \{(aba)^n \mid n \geq 1\} \end{aligned}$$

For each of the above languages, state whether it belongs to REG, to CFL$\setminus$REG, or else whether it is outside of CFL. Provide a mathematical proof for all of your answers.

Solution

(a) L_1 belongs to the class CFL$\setminus$REG.

We first show that L_1 is not a regular language, by applying the pumping lemma for this class. In what follows, we view each string in L_1 as composed by two blocks of occurrences of symbol a, one at the left of the occurrence of b and the other to its right. These two blocks must have the same length.

Let N be the pumping lemma constant for L_1. We choose the string $w = a^N b a^N \in L_1$ with $|w| \geq N$. We now consider all possible factorizations of the form $w = xyz$ satisfying the conditions $|y| \geq 1$ and $|xy| \leq N$ of the pumping lemma. Since $|xy| \leq N$, string y can only span over the block of a's placed at the left of b. Therefore we need to consider only one case in our discussion.

We choose $k = 0$ in the pumping lemma, and obtain the new string $w_{k=0} = xy^0z = xz$, which has the form $a^{N-|y|} b a^N$. Since $|y| \geq 1$, the two blocks of a's do not have the same length, and thus $w_{k=0} \notin L_1$. We conclude that L_1 does not satisfy the pumping lemma, and therefore cannot be a regular language.

As a second part of the answer, we need to show that L_1 belongs to the class CFL. Consider the CFG G_1 with productions:

$$\begin{aligned} S &\to aSa \mid aBa \\ B &\to b \end{aligned}$$

It is not too difficult to see that $L(G_1) = L_1$.

(b) L_2 belongs to the class CFL$\setminus$REG.

We first show that L_2 is not a regular language, by applying the pumping lemma for this class. To this end, it is very useful to observe that every string in L_2 has the property that the number of occurrences of symbol a is twice the number of occurrences of symbol b.

Let N be the pumping lemma constant for L_2. We choose the string $w = a^N(ba)^N \in L_2$ with $|w| \geq N$. We now consider all possible factorizations of the form $w = xyz$ satisfying the conditions $|y| \geq 1$ and $|xy| \leq N$ of the pumping lemma. Since $|xy| \leq N$, string y can only span over the first (left-to-right) N occurrences of symbol a in w, that is, string w cannot include any occurrence of symbol b from w.

We then choose $k = 0$ in the pumping lemma, and obtain the new string $w_{k=0} = xy^0z = xz$, which has the form $a^{N-|y|}(ba)^N$. Since $|y| \geq 1$, it is immediate to see that $w_{k=0}$ violates the condition that the number of occurrences of symbol a is twice the number of occurrences of symbol b, and thus $w_{k=0} \notin L_2$. This is a violation of the pumping lemma, and we conclude that L_2 cannot be a regular language.

As a second part of the answer, we need to show that L_2 belongs to the class CFL. Consider the CFG G_2 with productions:

$$\begin{aligned} S &\to aSB \mid aB \\ B &\to ba \end{aligned}$$

It is not too difficult to see that $L(G_2) = L_2$.

(c) L_3 belongs to the class REG.

It is very easy to see that L_3 is generated by the regular expression $R = \boldsymbol{aba}(\boldsymbol{aba})^*$.

3. **[5 points]** Consider the CFG G implicitly defined by the following productions:

$$\begin{aligned} S &\to ABA \mid BAB \mid BBB \\ A &\to aAB \mid bBB \\ B &\to b \mid \varepsilon \end{aligned}$$

Apply the methods specified in the textbook, in the proper order, to transform G into a new CFG G' in Chomsky normal form such that $L(G') = L(G) \setminus \{\varepsilon\}$. Report the CFGs obtained at each of the **intermediate steps**.

Important: do not use any other construction different from the one presented in the textbook.

Solution

The algorithms that need to be applied to the grammar G are specified in the following list, in the required order, and are all reported in Chapter 7 of the textbook

- elimination of ε-productions
- elimination of unary productions
- elimination of useless symbols
- construction of a CFG in Chomsky normal form

(a) The set of nullable variables of G is $n(G) = \{B\}$. After elimination of the ε-productions we obtain the intermediate CFG G_1

$$\begin{aligned} S &\rightarrow ABA \mid BAB \mid BBB \mid AA \mid AB \mid BA \mid BB \mid A \mid B \\ A &\rightarrow aAB \mid bBB \mid aA \mid bB \mid b \\ B &\rightarrow b \end{aligned}$$

(b) There are two unary productions in G_1: $S \rightarrow A$ and $S \rightarrow B$. Thus the set of unary pairs of G_1 is

$$u(G_1) \quad = \quad \{(S, A), (S, B)\} \cup \{(X, X) \mid X \in \{S, A, B\}\}.$$

After elimination of the unary productions we obtain the intermediate CFG G_2

$$\begin{aligned} S &\rightarrow ABA \mid BAB \mid BBB \mid AA \mid AB \mid BA \mid BB \\ S &\rightarrow aAB \mid bBB \mid aA \mid bB \mid b \\ A &\rightarrow aAB \mid bBB \mid aA \mid bB \mid b \\ B &\rightarrow b \end{aligned}$$

(c) All nonterminals in G_2 are reachable and generating, that is, there are no useless nonterminals in G_2. Therefore this step does not change the intermediate CFG obtained at the previous step.

(d) The construction of a CFG in Chomsky normal form from G_2 proceeds in two steps. The first step eliminates terminal symbols in the right-hand side of the productions of G_2, in case they appear along with some other symbols. To do this we introduce new nonterminal symbols C_a, C_b and produce the intermediate CFG G_3

$$\begin{aligned} S &\rightarrow ABA \mid BAB \mid BBB \mid AA \mid AB \mid BA \mid BB \\ S &\rightarrow C_aAB \mid C_bBB \mid C_aA \mid C_bB \mid b \\ A &\rightarrow C_aAB \mid C_bBB \mid C_aA \mid C_bB \mid b \\ B &\rightarrow b \\ C_a &\rightarrow a \\ C_b &\rightarrow b \end{aligned}$$

The second step factorizes productions of G_3 having right-hand side of length larger than two. To do this we introduce new nonterminal symbols D, E, F and produce CFG G_4

$$\begin{aligned} S &\rightarrow AD \mid BE \mid BF \mid AA \mid AB \mid BA \mid BB \\ S &\rightarrow C_aE \mid C_bF \mid C_aA \mid C_bB \mid b \\ A &\rightarrow C_aE \mid C_bF \mid C_aA \mid C_bB \mid b \\ B &\rightarrow b \\ D &\rightarrow BA \\ E &\rightarrow AB \\ F &\rightarrow BB \\ C_a &\rightarrow a \\ C_b &\rightarrow b \end{aligned}$$

CFG G_4 is in Chomsky normal form, and we have $L(G_4) = L(G) \smallsetminus \{\varepsilon\}$. The desired CFG G' is then G_4.

4. **[6 points]** Assess whether the following statements are true or false, providing motivations for all of your answers.

   (a) There exists languages L_1, L_2 in REG, defined over alphabet $\Sigma = \{a, b\}$, such that $L_1 \cap L_2 = \emptyset$ and $L_1 \cup L_2 = \Sigma^*$.

   (b) There exists languages L_1, L_2 in CFL$\smallsetminus$REG, defined over alphabet $\Sigma = \{a, b\}$, such that $L_1 \cap L_2 = \emptyset$ and $L_1 \cup L_2 = \Sigma^*$.

   (c) For every language L in CFL and for every string $w \in L$, we have that $L \smallsetminus \{w\}$ is in CFL.

   (d) The class $\mathcal{P}$ of languages that can be recognized in polynomial time by a TM is closed under union.

Solution

(a) True. We can satisfy the conditions in the question by taking $L_1 = \emptyset$ and $L_2 = \Sigma^*$.
More generally, observe that for every language L_1 in REG and $L_2 = \overline{L_1}$, that is, L_2 is the complement of L_1, we have that L_1 and L_2 satisfy the conditions in the question.

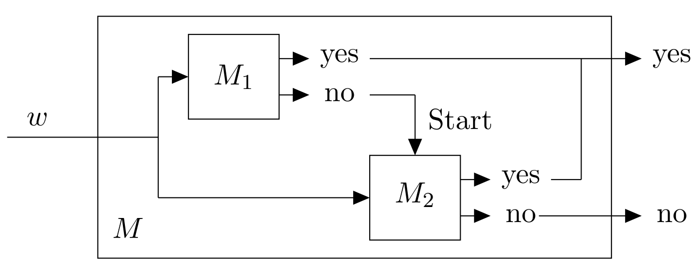
(b) True. For a string w defined over Σ and for any symbol $X \in \Sigma$, let us write $\#_X(w)$ to denote the number of occurrences of X in w. We then define

$$\begin{aligned} L_1 &= \{w \mid w \in \Sigma^*, \#_a(w) = \#_b(w)\} \\ L_2 &= \{w \mid w \in \Sigma^*, \#_a(w) \neq \#_b(w)\} \end{aligned}$$

We know from the textbook that both L_1 and L_2 are in CFL$\smallsetminus$REG. It is also easy to see that $L_1 \cap L_2 = \emptyset$ and $L_1 \cup L_2 = \Sigma^*$.

(c) True. We can express set difference through intersection and complementation, and write $L \smallsetminus \{w\} = L \cap \overline{\{w\}}$. We also observe that $\{w\}$ is a finite language, and hence a regular language. Since regular languages are closed under complementation, $\overline{\{w\}}$ is also a regular language. Finally, the class of context-free languages is closed under the intersection with regular languages. Therefore we have that $L \cap \overline{\{w\}}$ is still in CFL.

(d) True. Let L_1 and L_2 be two arbitrary languages in $\mathcal{P}$. From the definition of $\mathcal{P}$, there exist TMs M_1 and M_2, both running in polynomial time, such that $L(M_1) = L_1$, and $L(M_2) = L_2$.
Consider the Turing machine M defined in the following block diagram.



It is immediate to see that $L(M) = L_1 \cup L_2$. Furthermore, since both M_1 and M_2 run in polynomial time and are simulated only once, M also runs in overall polynomial time.

5. **[9 points]** In relation to the theory of Turing machines (TMs), answer the following questions. All the TMs introduced below are defined over the input alphabet $\Sigma = \{0, 1\}$.

For a string $w \in \Sigma^*$ we write $\overline{w}$ to represent the string obtained from w by changing all occurrences of 0 into 1 and all occurrences of 1 into 0. As an example, we have $\overline{011001} = 100110$. Consider the following property of the RE languages

$$\mathcal{P} \quad = \quad \{L \mid L \in \text{RE}, \ \text{ for every string } w \in L \text{ we have } \overline{w} \notin L\}$$

and define $L_{\mathcal{P}} = \{\mathsf{enc}(M) \mid L(M) \in \mathcal{P}\}$.

(a) Use Rice's theorem to prove that $L_{\mathcal{P}}$ is not in REC.

(b) Prove that $L_{\mathcal{P}}$ is not in RE.

(c) For TMs M_1, M_2 we write $\mathsf{enc}(M_1, M_2)$ to represent some fixed binary encoding of these machines. Consider the language

$$L \quad = \quad \{\mathsf{enc}(M_1, M_2) \mid \ \text{ for every string } w \in L(M_1) \text{ we have } \overline{w} \notin L(M_2)\} \ .$$

Show that L is not in RE by establishing a reduction $L_{\mathcal{P}} \leq_m L$.

Solution

(a) We show that the property $\mathcal{P}$ is nontrivial, that is, $\mathcal{P}$ is neither empty nor equal to RE.

- $\mathcal{P} \neq \emptyset$. The language $L_1 = \{011001\}$ is in RE, since it is finite. We now have to check that for every string $w \in L_1$ we have $\overline{w} \notin L_1$. There is only one string in L_1, namely 011001, and $\overline{011001} = 100110$ is not in L_1. We conclude that $L_1 \in \mathcal{P}$ and thus $\mathcal{P} \neq \emptyset$.
- $\mathcal{P} \neq$ RE. The language $L_2 = \{011001, 100110\}$ is in RE, since it is finite. For string $011001 \in L_2$ we have that $\overline{011001} = 100110 \in L_2$. This means that $L_2 \notin \mathcal{P}$ and then $\mathcal{P} \neq$ RE.

We can now apply Rice's theorem and conclude that, since $\mathcal{P}$ is nontrivial, $L_{\mathcal{P}}$ is not in REC.

(b) We now show that $L_{\mathcal{P}}$ is not in RE. The most convenient way to do this is to consider the complement language $\overline{L_{\mathcal{P}}} = L_{\overline{\mathcal{P}}}$, where $\overline{\mathcal{P}}$ is the complement of the class $\mathcal{P}$ with respect to RE, and can be specified as

$$\overline{\mathcal{P}} \quad = \quad \{L \mid L \in \text{RE}, \ \text{there exists a string } w \in L \text{ such that } \overline{w} \in L\} \ .$$

We now define a nondeterministic TM N such that $L(N) = L_{\overline{\mathcal{P}}}$. Since every nondeterministic TM can be converted into a standard TM, this shows that $L_{\overline{\mathcal{P}}}$ is in RE. Our nondeterministic TM N takes as input the encoding $\mathsf{enc}(M)$ of a TM M and performs the following steps.

- N nondeterministically guesses a string $w \in \Sigma^*$.

- N simulates M on w. If this computation terminates with a positive answer, then N moves on with the next step. If the computation terminates with a negative answer, then N does not accept and halts. If the simulation of M on w does not halt, then N runs for ever and therefore does not accept its input.
- N simulates M on $\overline{w}$. If this computation terminates with a positive answer, then N accepts and halts. If the computation terminates with a negative answer, then N does not accept and halts. Finally, if the simulation of M on $\overline{w}$ does not halt, then N runs for ever and therefore does not accept its input.

It is not difficult to see that $L(N) = L_{\overline{\mathcal{P}}}$.

Since $L_{\overline{\mathcal{P}}}$ is in RE, if its complement language $L_{\mathcal{P}}$ were in RE as well, then we would conclude that both languages are in REC, from a theorem in Chapter 9 of the textbook. But we have already shown in (a) that $L_{\mathcal{P}}$ is not in REC. We must therefore conclude that $L_{\mathcal{P}}$ is not in RE.

(c) Recall from Chapter 9 that, in order to provide a reduction $L_{\mathcal{P}} \leq_m L$, we need to establish a mapping m from input instances of $L_{\mathcal{P}}$ to output instances of L such that positive instances are mapped to positive instances and negative instances are mapped to negative instances. From a known theorem about reductions, since $L_{\mathcal{P}}$ is not in RE then L cannot be in RE as well.

We need to map strings of the form $\mathsf{enc}(M)$ into strings of the form $\mathsf{enc}(M_1, M_2)$. Our reduction m does this by setting $M_1 = M_2 = M$. To conclude the proof, we now show the desired relation between the mapped instances, by means of the following chain of logical equivalences:

$$\begin{array}{llll} \mathsf{enc}(M) \in L_{\mathcal{P}} & \text{iff} & L(M) \in \mathcal{P} & \text{(definition of } L_{\mathcal{P}}) \\ & \text{iff} & \text{for every string } w \in L(M), \overline{w} \notin L(M) & \text{(definition of } \mathcal{P}) \\ & \text{iff} & \text{for every string } w \in L(M_1), \overline{w} \notin L(M_2) & \text{(definition of reduction } m) \\ & \text{iff} & \mathsf{enc}(M_1, M_2) \in L & \text{(definition of } L) \ . \end{array}$$