University of Padua
Department of Information Engineering
Master Degree in Computer Engineering

Automata, Languages and Computation

SELECTED EXERCISES WITH SOLUTIONS

prof. Giorgio Satta

Acknowledgments

The exercises and the solutions presented in this collection are based on the material listed below.

- Collection of exercises with solutions from the class “Informatica Teorica”, within the Bachelor Degree Program in Computer Engineering, at University of Padua. The exercises were developed during academic years 1993/94, 1994/95, 1995/96 by students Paolo Chioetto, Carlo Fantozzi and Ferruccio Fantozzi.

- Collection of exercises with solutions from the class “Informatica Teorica”, within the the First Degree Program in Information Engineering, at University of Padua. The exercises were developed during academic years from 2005/06 to 2017/18 by professor Maria Silvia Pini and professor Cinzia Pizzi.

- Collection of exercises with solutions from the class “Automi, Linguaggi Formali e Computazione”, within the Second Degree Program in Computer Engineering, at University of Padua. The exercises were developed during academic year 2018/19 by students Francesco Cazzaro, Samuele Papa and Federico Soldà.

Notation

What follows are some notational conventions related to automata and formal languages, which have not been introduce by the class textbook but which are quite common in the literature and will be used throughout the following chapters.

- Symbol $\mathbb{N}$ denotes the set of natural numbers, including zero.
- Given two sets S_1 and S_2, we write $S_1 \smallsetminus S_2$ to denote the set difference between S_1 and S_2.
- Let Σ be a finite alphabet and let L be some language over Σ. We denote as $\overline{L}$ the complement language of L, defined as $\overline{L} = \Sigma^* \smallsetminus L$.
- Let w be a string over an alphabet Σ and let $a \in \Sigma$. We write $\#_a(w)$ to denote the number of occurrences of a in w.
- For a generic string w and an integer i with $1 \leq i \leq |w|$, we write $w[i]$ to denote the i-th symbol in w, from left to right.

Contents

Chapter 1

Regular Languages

Exercise 1.1 Consider the deterministic finite state automaton $A = (Q, \Sigma, \delta, q_0, F)$ with input alphabet $\Sigma = \{0,1\}$, set of states $Q = \{q_0, q_1\}$, initial state q_0, and final state set $F = \{q_1\}$. The transition function δ is graphically represented in Figure 1.1.

Prove that A recognizes the language

$$L \quad = \quad \{w \mid w \in \{0,1\}^*,\ \#_1(w) \text{ odd}\}.$$

Solution The textbook explains that to prove the equivalence $L = L(A)$, where A is a deterministic finite state automaton, we need to define a family of properties $\mathcal{P}_q$, one for each state q of A. In our case, properties $\mathcal{P}_q$ are defined over the set $\{0,1\}^*$ of binary strings. Next, we must prove that, for each property $\mathcal{P}_q$ and for each string $x \in \{0,1\}^*$, we have that $\mathcal{P}_q(x)$ holds true if and only if $\hat{\delta}(q_0, x) = q$, where q_0 indicates the initial state of A. Informally, this means that $\mathcal{P}_q(x)$ is true if and only if, starting from the initial state and reading x, the automaton reaches state q. The proof must be developed using the mutual induction technique; see at the end of the exercise solution for further discussion of this issue.

We proceed by defining the two properties $\mathcal{P}_{q_0}$ and $\mathcal{P}_{q_1}$.

- For every $x \in \{0,1\}^*$, $\mathcal{P}_{q_0}(x)$ holds if and only if $\#_1(x)$ is even.
- For every $x \in \{0,1\}^*$, $\mathcal{P}_{q_1}(x)$ holds if and only if $\#_1(x)$ is odd.

We must now prove the following two statements.

(i) For every $x \in \{0,1\}^*$, $\mathcal{P}_{q_0}(x)$ holds if and only if $\hat{\delta}(q_0, x) = q_0$.

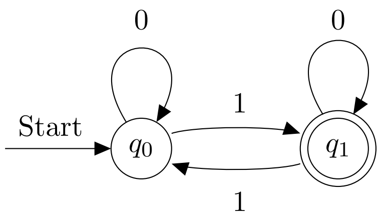

Figure 1.1: Graphical representation of the transition function of DFA A from Exercise 1.1.

(ii) For every $x \in \{0,1\}^*$, $\mathcal{P}_{q_1}(x)$ holds if and only if $\hat{\delta}(q_0, x) = q_1$.

Informally, (i) states that string x has an even number of occurrences of 1 if and only if, starting from the initial state and reading x, A reaches state q_0. Similarly, (ii) states that string x has an odd number of occurrences of 1 if and only if, starting from the initial state and reading x, A reaches state q_1. Intuitively, for a state q of A the property $\mathcal{P}_q$ indicates what language A would accept if A had q as the only one final state.

At this point we observe that statement (ii) can be used to prove the statement in the exercise. In fact we have the following chain of equivalences

$$\begin{array}{llll} x \in L(A) & \text{iff} & \hat{\delta}(q_0, x) = q_1 & (L(A)\text{'s definition}) \\ & \text{iff} & \mathcal{P}_{q_1}(x) & (\text{statement (ii)}) \\ & \text{iff} & \#_1(x) \text{ is odd} & (\mathcal{P}_{q_1}\text{'s definition}) \\ & \text{iff} & x \in L & (L\text{'s definition}) \end{array}$$

which leads us to the conclusion $L(A) = L$.

To complete the exercise, we must therefore demonstrate the statements (i) and (ii). Since (i) and (ii) are of the form “if and only if”, we need to prove four statements, and for each statement we need to prove the base case and the inductive case. As already suggested, we use mutual induction.

(Part (i), only if) We have to prove that, for every $x \in \{0,1\}^*$, if $\mathcal{P}_{q_0}(x)$ then $\hat{\delta}(q_0, x) = q_0$. We proceed by induction on the length of x.

Base. We have $|x| = 0$, which implies $x = \varepsilon$. Since $\#_1(\varepsilon)$ is even, $\mathcal{P}_{q_0}(\varepsilon)$ holds. Also, by definition of $\hat{\delta}$ we can write $\hat{\delta}(q_0, \varepsilon) = q_0$. We conclude then that the implication holds true.

Induction. Let $|x| = n > 0$. If $\mathcal{P}_{q_0}(x)$ is false, then the implication is always true. Let's consider instead the case in which $\mathcal{P}_{q_0}(x)$ is true, that is, $\#_1(x)$ is even. Let us write $x = ya$ with $a \in \{0, 1\}$ and $y \in \{0, 1\}^*$, $|y| = n - 1$. We need to distinguish two cases, depending on whether $a = 0$ or $a = 1$.

- If $a = 0$, then $\#_1(y) = \#_1(x)$ and therefore $\mathcal{P}_{q_0}(y)$ holds true. We then apply the inductive hypotheses for the only if part of statement (i), and conclude that $\hat{\delta}(q_0, y) = q_0$. From the definition of the transition function of A, we have that from state q_0 and by reading symbol 0 we can reach state q_0. So we can write
$$\hat{\delta}(q_0, x) = \delta(\hat{\delta}(q_0, y), 0) = \delta(q_0, 0) = q_0.$$
- If $a = 1$, then $\#_1(y) = \#_1(x) - 1$, that is, $\#_1(y)$ is odd, and hence $\mathcal{P}_{q_1}(y)$ holds. We can then apply the inductive hypothesis for part only if of statement (ii), and conclude that $\hat{\delta}(q_0, y) = q_1$. From the definition of the transition function of A, we have that from state q_1 and reading symbol 1 we can reach state q_0. Therefore we can write
$$\hat{\delta}(q_0, x) = \delta(\hat{\delta}(q_0, y), 1) = \delta(q_1, 1) = q_0.$$

This concludes the inductive part of the proof showing the only if part of the statement (i).

(Part (i), if) We have to prove that, for every $x \in \{0, 1\}^*$, if $\hat{\delta}(q_0, x) = q_0$ then $\mathcal{P}_{q_0}(x)$ holds. Again, we proceed by induction on the length of string x.

Base. We have $|x| = 0$, that is, $x = \varepsilon$. The relation $\hat{\delta}(q_0, \varepsilon) = q_0$ is always true, by definition of $\hat{\delta}$. Furthermore, $\mathcal{P}_{q_0}(\varepsilon)$ is also true, since $\#_1(\varepsilon)$ is even. We therefore conclude that the implication is true.

Induction. Let $|x| = n > 0$. If $\hat{\delta}(q_0, x) = q_0$ is false, then the implication we want to prove is always true. Let's consider now the case where $\hat{\delta}(q_0, x) = q_0$ is true. As before, let us write $x = ya$ with $a \in \{0, 1\}$ and $y \in \{0, 1\}^*$, $|y| = n - 1$. We distinguish two cases, depending on whether $a = 0$ or $a = 1$.

- Let $a = 0$. Since $\hat{\delta}(q_0, x) = q_0$, after reading x the automaton must be in state q_0. The last symbol of x is 0, and from the definition of A we see that by reading 0 we can reach state q_0 only if the automaton is already in q_0 after reading y. Therefore we conclude that $\hat{\delta}(q_0, y) = q_0$. We can now apply the inductive hypothesis for part if of statement (i), and conclude that $\mathcal{P}_{q_0}(y)$ is true, that is, y has an even number of occurrences of 1. Then x also has an even number of occurrences of 1, and we can assert that $\mathcal{P}_{q_0}(x)$ is also true.

- Let $a = 1$. Since $\hat{\delta}(q_0, x) = q_0$, the automaton is in state q_0 after reading x. The last symbol of x is 1, and from the definition of A we see that, by reading 1, state q_0 can be reached only if the automaton is in state q_1 after reading y. We then conclude that $\hat{\delta}(q_0, y) = q_1$. We now apply the inductive hypothesis for part if of statement (ii), and conclude that $\mathcal{P}_{q_1}(y)$ holds true, that is, y has an odd number of occurrences of 1. Since $\#_1(y) = \#_1(x) - 1$, we conclude that x has an even number of occurrences of 1, and we can assert that $\mathcal{P}_{q_0}(x)$ is true.

This concludes the inductive part of the proof, and we have then completed the if part of statement (i).

The if and the only if parts of statement (ii) can be proved in a very similar way to the proof of the same parts for statement (i) reported above. These two proofs are therefore omitted.

Note In this end note to the above exercise, we discuss the need for the use of mutual induction in the proof of statements (i) and (ii). Let us consider the inductive case of the only if part of statement (i). We have discussed two different scenarios, depending on whether string x ends with 0 or 1. In the latter case, we have $x = y1$, so if x has an even number of occurrences of 1, then y has an odd number of occurrences of the same symbol. For this reason, we can't invoke on y the inductive hypothesis of the same statement (i) we are proving, which concerns strings with an even number of occurrences of 1: we must instead invoke on y the inductive hypothesis of statement (ii), which concerns strings with an odd number of occurrences of 1. This requires the technique of mutual induction.

Exercise 1.2 Let $N = (Q, \Sigma, \delta_N, q_0, F)$ be an NFA with $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, $F = \{q_2\}$ and let δ_N be the transition function represented in tabular form in Figure 1.2. Construct a DFA D equivalent to N.

Solution In order to construct a DFA D equivalent to N, we exploit the *lazy evaluation* construction in which D's states are produced only when needed. We start with the initial state q_0 of N, which corresponds to the state $\{q_0\}$ of D, and we compute all the states that can be reached in the NFA by reading one occurrence of the two terminal symbols 0 and 1. The set of states obtained in this way is a single state in the DFA, for which the transition function will be computed. This DFA state represents the fact that, in the NFA, we can be in any of the states belonging to the set.

δ_N	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
q_1	$\{q_1, q_2\}$	$\emptyset$
$*q_2$	$\emptyset$	$\{q_2\}$

Figure 1.2: Transition function in tabular form for the NFA of Exercise 1.2.

We recall the reader that the formula to get the next state from a state $\{q_1, q_2, \ldots, q_r\}$ of the DFA, by reading symbol $a \in \Sigma$, is as follows:

$$\delta_D(\{q_1, q_2, \ldots, q_r\}, a) \quad = \quad \bigcup_{i=1}^{r} \delta_N(q_i, a).$$

We report below some among the first states of D, in order of their reachability from the initial state $\{q_0\}$:

$$\begin{aligned}
\delta_D(\{q_0\}, 0) &= \delta_N(q_0, 0) = \{q_0, q_1\} \\
\delta_D(\{q_0\}, 1) &= \delta_N(q_0, 1) = \{q_1\} \\
\delta_D(\{q_0, q_1\}, 0) &= \delta_N(q_0, 0) \cup \delta_N(q_1, 0) = \{q_0, q_1\} \cup \{q_1, q_2\} = \{q_0, q_1, q_2\} \\
\delta_D(\{q_0, q_1\}, 1) &= \delta_N(q_0, 1) \cup \delta_N(q_1, 1) = \{q_1\} \cup \emptyset = \{q_1\} \\
\delta_D(\{q_1\}, 0) &= \delta_N(q_1, 0) = \{q_1, q_2\} \\
\delta_D(\{q_1\}, 1) &= \delta_N(q_1, 1) = \emptyset
\end{aligned}$$

In a similar way, we can obtain the remaining states (the intermediate steps are omitted):

$$\begin{aligned}
\delta_D(\{q_0, q_1, q_2\}, 0) &= \{q_0, q_1, q_2\} \\
\delta_D(\{q_0, q_1, q_2\}, 1) &= \{q_1, q_2\} \\
\delta_D(\{q_1, q_2\}, 0) &= \{q_1, q_2\} \\
\delta_D(\{q_1, q_2\}, 1) &= \{q_2\} \\
\delta_D(\{q_2\}, 0) &= \emptyset \\
\delta_D(\{q_2\}, 1) &= \{q_2\}
\end{aligned}$$

The resulting DFA is graphically represented in Figure 1.3.

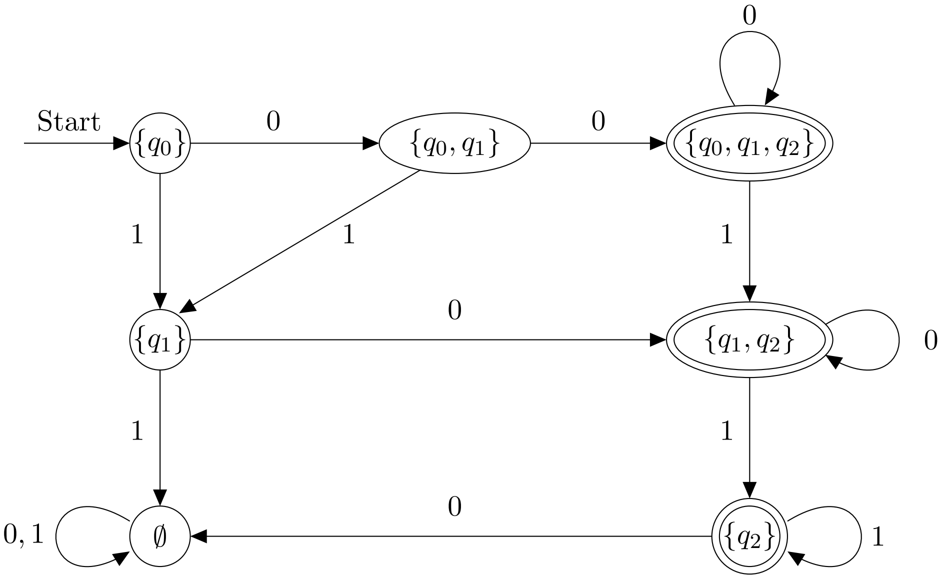

Figure 1.3: DFA D equivalent to the NFA with transition table reported in Figure 1.2.

Exercise 1.3 Construct a DFA recognizing the language of all strings w over $\{a, b\}$ such that the third to last symbol in w is b.

Solution The required language can be formally specified as follows

$$L = \{xby \mid x, y \in \{a, b\}^*, |y| = 2\}.$$

It's not difficult to convince oneself that L contains all of the strings whose third to last symbol is b. Even if the exercise requires to exhibit a deterministic automaton that recognizes L, it is convenient to build a nondeterministic automaton first. Afterwards, it will be possible to derive a deterministic version of the automaton by applying the construction already exploited by the solution to Exercise 1.2, that is, the construction used in the proof of the equivalence of languages recognized by the DFA and the NFA classes. The choice of this procedure stems from the fact that, as often happens, the nondeterministic automaton has a reduced number of states: this simplifies the specification of the automaton itself, as well as the proof that the recognised language is the required one.

We therefore specify the NFA $M = (Q, \{a, b\}, \delta, q_0, F)$, with $Q = \{q_0, q_1, q_2, q_3\}$ and $F = \{q_3\}$. The transition function $\delta : Q \times \{a, b\} \mapsto 2^Q$ is defined in Figura 1.4, both in the table and in the graphical form.

stato	q_0	q_1	q_2	q_3
a	$\{q_0\}$	$\{q_2\}$	$\{q_3\}$	$\emptyset$
b	$\{q_0, q_1\}$	$\{q_2\}$	$\{q_3\}$	$\emptyset$

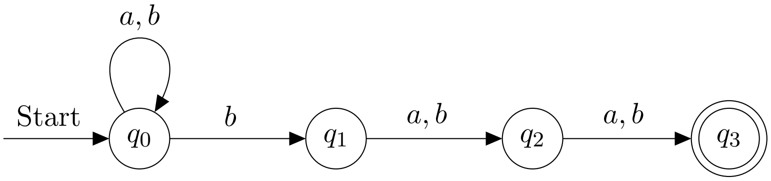

Figure 1.4: Transition table and graphical representation of the thransition function of the nondeterministic automaton M in the solution of Exercise 1.3.

We will now show that the language recognized by M is exactly L, that is, $L(M) = L$. Rather than producing a rigorous proof using the mutual induction technique, as previously done for Exercise 1.1, we propose here a shorter but more intuitive argument, although a less rigorous one. The proof is divided into two parts.

(Part $L \subseteq L(M)$) Given a generic string $z \in L$, we have to show that $z \in L(M)$. This result can be obtained by induction on the length of z.

Base. The shortest string in L can be obtained by letting $x = \varepsilon$ in the definition of the language: in this case, we have $z = by$ with $|y| = 2$, so that $|z| = 3$. Starting from the initial state, M can read symbol b, that is, the first symbol of z, and then move to state q_1, since $q_1 \in \delta(q_0, b)$. Afterwards, independently from the two symbols composing string y, M necessarily moves to state q_2 and then to the final state q_3, since $\delta(q_1, a) = \delta(q_1, b) = \{q_2\}$ and $\delta(q_2, a) = \delta(q_2, b) = \{q_3\}$. We have then found a path labeled $z = by$ from the initial state to a final state, and we can conclude that $zL(M)$.

Induction. Assume $|z| > 3$; by definition of L, we can write $z = xby$, with $|x| \geq 1$. Since x contains at least one symbol, we can also write $z = Xx'by$, where $X \in \{a, b\}$ represents the first symbol of x and x' belongs to $\{a, b\}^*$. Starting from the initial state, and independently of the specific symbol X, M can stay in state q_0, since $q_0 \in \delta(q_0, a)$ e $q_0 \in \delta(q_0, b)$. Let us now observe that the string $x'by$ also belongs to L, and we have $|x'by| < |z|$. By the inductive hypothesis, then, we derive that $x'by$ belongs to $L(M)$, and therefore there must be some path from q_0 to q_3 labeled by $x'by$. Overall, we've thus shown the existence of a path labeled $Xx'by = z$ from the initial state to some final state, and we can conclude that $z \in L(M)$.

(Part $L(M) \subseteq L$) For a generic string $z \in L(M)$, we define $n = |z|$ and we show that $z \in L(M)$ implies $z \in L$. We proceeds by induction on n.

Base. According to the definition of the transition function δ, in order to move from the initial state to the final state of M we need to visit in sequence all of the intermediate states of the automaton. Since M has a total of four states, it takes at least three steps to accept any string. It is easy to see that the paths of length 3 from q_0 to q_3 are labeled *baa*, *bab*, *bba*, *bbb*. In all cases, b is the third to last symbol, and we can conclude that all strings accepted by the automaton in three steps belong to L.

Induction. Assume now that $n > 3$. We observe that, in the first computation step, M must necessarily stay in state q_0: if it does not, the automaton would reach in two steps the final state, from which no more moves are possible. We can then write $z = Xx$, where $X \in \{a, b\}$ represents the first symbol of z. String x is accepted by the automaton in less than n steps. Using the inductive hypothesis, we derive that x belongs to L, and therefore its third to last symbol is a b. But the third to last symbol of x is also the third to last symbol of $Xx = z$. We can then conclude that $z \in L$.

We have concluded our proof that the nondeterministic automaton M exactly recognizes the language L. To complete the exercise, we now build an equivalent deterministic automaton $M' = (Q', A', B', F')$, using the same technique already exploited in Exercise 1.2. The states in Q' are subsets of Q and represent the sets of states in which "different copies" of M can be at the same time, after reading the same input string. To compute the elements of Q' we use the following definition:

$$\delta'(\{q_1, q_2, \ldots, q_r\}, X) \quad = \quad \bigcup_{i=1}^{r} \delta(q_i, X), \qquad X \in \{a, b\}.$$

With this definition, we start with the initial state $\{q_0\}$ in M', and and compute all reachable states.

$$\begin{aligned}
\delta'(\{q_0\},a) &= \delta(q_0,a) = \{q_0\}\\
\delta'(\{q_0\},b) &= \delta(q_0,b) = \{q_0,q_1\}\\
\delta'(\{q_0,q_1\},a) &= \delta(q_0,a)\cup\delta(q_1,a) = \{q_0\}\cup\{q_2\} = \{q_0,q_2\}\\
\delta'(\{q_0,q_1\},b) &= \delta(q_0,b)\cup\delta(q_1,b) = \{q_0,q_1\}\cup\{q_2\} = \{q_0,q_1,q_2\}\\
\delta'(\{q_0,q_2\},a) &= \delta(q_0,a)\cup\delta(q_2,a) = \{q_0\}\cup\{q_3\} = \{q_0,q_3\}\\
\delta'(\{q_0,q_2\},b) &= \delta(q_0,b)\cup\delta(q_2,b) = \{q_0,q_1\}\cup\{q_3\} = \{q_0,q_1,q_3\}\\
\delta'(\{q_0,q_1,q_2\},a) &= \delta(q_0,a)\cup\delta(q_1,a)\cup\delta(q_2,a)\\
&= \{q_0\}\cup\{q_2\}\cup\{q_3\} = \{q_0,q_2,q_3\}\\
\delta'(\{q_0,q_1,q_2\},b) &= \delta(q_0,b)\cup\delta(q_1,b)\cup\delta(q_2,b)\\
&= \{q_0,q_1\}\cup\{q_2\}\cup\{q_3\} = \{q_0,q_1,q_2,q_3\}\\
\delta'(\{q_0,q_3\},a) &= \delta(q_0,a)\cup\delta(q_3,a) = \{q_0\}\cup\emptyset = \{q_0\}\\
\delta'(\{q_0,q_3\},b) &= \delta(q_0,b)\cup\delta(q_3,b) = \{q_0,q_1\}\cup\emptyset = \{q_0,q_1\}\\
\delta'(\{q_0,q_1,q_3\},a) &= \delta(q_0,a)\cup\delta(q_1,a)\cup\delta(q_3,a)\\
&= \{q_0\}\cup\{q_2\}\cup\emptyset = \{q_0,q_2\}\\
\delta'(\{q_0,q_1,q_3\},b) &= \delta(q_0,b)\cup\delta(q_1,b)\cup\delta(q_3,b)\\
&= \{q_0,q_1\}\cup\{q_2\}\cup\emptyset = \{q_0,q_1,q_2\}\\
\delta'(\{q_0,q_2,q_3\},a) &= \delta(q_0,a)\cup\delta(q_2,a)\cup\delta(q_3,a)\\
&= \{q_0\}\cup\{q_3\}\cup\emptyset = \{q_0,q_3\}\\
\delta'(\{q_0,q_2,q_3\},b) &= \delta(q_0,b)\cup\delta(q_2,b)\cup\delta(q_3,b)\\
&= \{q_0,q_1\}\cup\{q_3\}\cup\emptyset = \{q_0,q_1,q_3\}\\
\delta'(\{q_0,q_1,q_2,q_3\},a) &= \delta(q_0,a)\cup\delta(q_1,a)\cup\delta(q_2,a)\cup\delta(q_3,a)\\
&= \{q_0\}\cup\{q_2\}\cup\{q_3\}\cup\emptyset = \{q_0,q_2,q_3\}\\
\delta'(\{q_0,q_1,q_2,q_3\},b) &= \delta(q_0,b)\cup\delta(q_1,b)\cup\delta(q_2,b)\cup\delta(q_3,b)\\
&= \{q_0,q_1\}\cup\{q_2\}\cup\{q_3\}\cup\emptyset = \{q_0,q_1,q_2,q_3\}
\end{aligned}$$

The set of states of M' is

$$\begin{aligned}
Q' \quad = \quad & \{\{q_0\},\{q_0,q_1\},\{q_0,q_2\},\{q_0,q_1,q_2\},\\
& \{q_0,q_3\},\{q_0,q_1,q_3\},\{q_0,q_2,q_3\},\{q_0,q_1,q_2,q_3\}\}.
\end{aligned}$$

The automaton M' has eight states, as opposed to the automaton M which has only four states. From the definition of the subset construction we know that the number of states of the equivalent DFA can grow exponentially: in our case, we were expecting a maximum of 16 states. The set F' consists of all states in Q' that contain at least one state in F; in our case,

$$F' \quad = \quad \{\{q_0, q_3\}, \{q_0, q_1, q_3\}, \{q_0, q_2, q_3\}, \{q_0, q_1, q_2, q_3\}\}.$$

Finally, the transition funcrion $\delta' : Q' \times \{a, b\} \mapsto Q'$ can be easily produced from the previous construction exploring the reachable subsets of Q. We have therefore completed the definition of the deterministic automaton M'. Since M and M' are equivalent, and since we have already shown that $L(M) = L$, we conclude that M' recognizes L, as required.

Exercise 1.4 State whether the following languages are regular or not, and justify your answer:

(i) $L_1 = \{w \mid w \in \{a, b\}^+,\ \#_a(w) \neq \#_b(w)\}$;

(ii) $L_2 = \{a, b\}^* \cdot L_1$.

Solution L_1 is not a regular language. To prove this fact, we introduce the auxiliary language

$$L_0 \quad = \quad \{w \mid w \in \{a, b\}^*,\ \#_a(w) = \#_b(w)\}$$

and prove that L_0 is not regular using the pumping lemma.

Let us assume that L_0 is regular. There exists a constant n such that each string $z \in L_0$, with $|z| \geq n$, can be written as $z = uvw$, with $|uv| \leq n$, $|v| \geq 1$ and $uv^iw \in L_0$ for every $i \geq 0$. Let us consider the string $z = a^n b^n$, which belongs to to the language and has a length strictly greater than n. We must factorize the string into three parts u, v, w, with $|uv| \leq n$ and $|v| \geq 1$. Since the first n characters of the string are all a, we must have $u = a^j$ and $v = a^k$, with $j + k \leq n$ and $k \geq 1$. According to the pumping lemma, also the string $z' = uv^0w = uw$ belongs to L_0: but this statement is false, since

$$\#_a(z') = \#_a(u) + \#_a(w) = \#_a(z) - \#_a(v) = n - k;$$
$$\#_b(z') = \#_b(u) + \#_b(w) = \#_b(w) = \#_b(z) = n,$$

with $n - k$ strictly less than n since $k \geq 1$. We have therefore established a contradiction, and we must conclude that L_0 is not a regular language.

Let us now observe that $L_1 = \{a, b\}^* \setminus L_0 = \overline{L_0}$. Since the class of regular languages is closed under complementation, if L_1 were regular then

L_0 would also be regular, against what we just have shown. Therefore we must conclude that L_1 is not a regular language.

The language L_2 is a regular language. In fact, we will show that $L_2 = \{a, b\}^+$, which is a well-known regular language. The proof of this fact is split into two parts.

(Part $L_2 \subseteq \{a, b\}^+$) Let z be a generic string in L_2. According to the definition of L_2, we can write $z = u \cdot v$, with $u \in \{a, b\}^*$ and $v \in L_1$. Since L_1 does not contain the null string, we have $v \neq \varepsilon$, and therefore $z \neq \varepsilon$. The statement of this part now follows immediately, since $\{a, b\}^+$ contains by definition each string that is composed of occurrences of a and b, excluding the null string.

(Part $\{a, b\}^+ \subseteq L_2$) Let z be a generic string in $\{a, b\}^+$. According to the definition of L_2, to show $z \in L_2$ we have to find strings u and v such that $z = u \cdot v$, with $u \in \{a, b\}^*$ and $v \in L_1$. Since the null string is not included in $\{a, b\}^+$, z contains at least one occurrence of a symbol. Therefore it is legitimate to place in v the last symbol occurrence in z, and to place in u all of the remaining symbol occurrences in z. Let us observe that we can have $v = a$ or $v = b$, but in either case $\#_a(v) \neq \#_b(v)$, and therefore $v \in L_1$ according to the definition of this language. Since $u \in \ a, b*$, this concludes our proof.

Exercise 1.5 Use the closure properties of regular languages to assess whether the following claims are true or false.

(i) If $L = L_1 \cup L_2$ is regular, then both languages L_1 and L_2 are regular.

(ii) If L_1 is not regular and L_2 is regular, then $L_1 \cap L_2$ is always regular.

Solution In order to simplify the presentation, let us assume that both L_1 and L_2 are defined over the same alphabet Σ.

(i) If $L = L_1 \cup L_2$ is regular, then both languages L_1 and L_2 are regular: this statement is false. To show this fact, we provide a counterexample. Consider any language L_1 which is not regular, and take the union with $L_2 = \Sigma^*$. The result is again Σ^*, which is a regular language, contrary to the claim above.

(ii) If L_1 is not regular and L_2 is regular, then $L_1 \cap L_2$ is always regular: this statement is false. Again, we provide a simple counterexample. Consider any non-regular language L_1 and perform the intersection with

$L_2 = \Sigma^*$, which is regular. The result is again L_1, which we have assumed to be a non-regular language.

Exercise 1.6 State whether the following languages, defined over the alphabet $\Sigma = \{a, b\}$, are regular or not, and motivate your answer:

(i) $L_= = \{a^n b^m \mid n, m \geq 0,\ n = m\}$;

(ii) $L_< = \{a^n b^m \mid n, m \geq 0,\ n < m\}$;

(iii) $L_> = \{a^n b^m \mid n, m \geq 0,\ n > m\}$;

(iv) $L_{\neq} = \ a^n b^m \mid n, m \geq 0,\ n \neq m\}$;

Solution (i) The $L_=$ language is not a regular language. To prove this statement, let us assume by now that $L_=$ is indeed regular. Then the pumping lemma should hold. Let n be the constant of the lemma, and let us choose the string $w \in L_=$ with $w = a^n b^n$, which has length $|w| \geq n$. We can write $w = xyz$, where $|xy| \leq n$ and $|y| = m \geq 1$. The pumping lemma states, among other things, that the string $w' = xz$ obtained by eliminating y from w must be a string of $L_=$. But this is not possible, since $w' = a^{n-m} b^n$ with $n - m \neq n$, and we have then violated $L_=$'s definition. We have therefore obtained a contradiction, and we must conclude that $L_=$ is not a regular language.

(ii) The language $L_<$ is not regular. We again proceed as above, assuming that $L_<$ is regular. We apply the pumping lemma and choose the string $w \in L_<$ with $w = a^n b^{n+1}$, which has length $|w| \geq n$. Then we can write $w = xyz$, where $|xy| \leq n$ and $|y| = m \geq 1$, and we must have that string $w_k = xy^k z$ is still a string of $L_<$, for any value of k with $k \geq 0$. This statement is not verified. In fact, taking for example $k = 2$, we get the string $w_2 = a^{n+m} b^{n+1}$. Since $m \geq 1$, we have that in w_2 the number of occurrences of symbol a is not smaller than the number of occurrences of symbol b, as required by $L_<$'s definition. So we have that $w_2 \notin L_<$, against our initial assumptions. We then have to conclude that $L_<$ is not regular.

Note We observe that, even for any other value of k with $k \geq 3$ we have that string w_k does not belong to $L_<$, since $w_k = a^{n-m} a^{km} b^{n+1}$ and again the number of occurrences of symbol a in w_k is not smaller than the number of occurrences of symbol b in the same string. However, this last observation is not necessary to our proof: in fact, to obtain a contradiction to our original assumption about $L_<$ being regular, it sufficis to prove that the pumping lemma is violated for at least one value of k.

(iii) The language $L_>$ is not regular. As in previous cases, let us assume that $L_>$ is regular and let n be the pumping lemma constant for the language. We now choose the string $w = a^n b^{n-1}$, which has length $|w| \geq n$. Then we can write $w = xyz$, where $|xy| \leq n$ and $|y| = m \geq 1$ and we have that, for each k value with $k \geq 0$, the string $w_k = xy^k z = a^{n-m} a^{km} b^{n-1}$ must still be a string of $L_>$. Choosing $k = 0$, we get the string $w_0 = a^{n-m} b^{n-1}$ which does not belong to $L_>$, since we have $m \geq 1$, and therefore in w_0 the number of occurrences of symbol a is not greater than the number of occurrences of symbol b, as required by $L_>$'s definition. This allows us to conclude that $L_>$ is not regular.

(iv) Finally, the language $L_{\neq}$ is not regular. However, in this case a direct application of the pumping lemma would be problematic; the reader is encouraged to make one attempt in this direction to realize this fact. We then take a different route with respect to cases (i), (ii) and (iii) above, More specifically, assuming that $L_{\neq}$ is regular, we will apply operators defined on regular languages that guarantee that the result is still a regular language, to arrive at a new language that we will show not to be regular using the pumping lemma. This allows us to conclude that even $L_{\neq}$ is not a regular language.

Complementing $L_{\neq}$ with respect to the alphabet $\Sigma = \{a, b\}$, we obtain the language $\overline{L_{\neq}}$ formed by all strings that have a certain number of a followed by the same number of b, as well as all strings that have a and b in various other positions that do not respect the $a^i b^j$ pattern. Since we have assumed that $L_{\neq}$ is regular, $\overline{L_{\neq}}$ should also be regular. This follows from the closure property of the regular languages under the complementation operator.

To make the next part of the proof easier, it is very useful to "filter" the language $\overline{L_{\neq}}$ using the auxiliary language

$$L_{ab} \quad = \quad \{a^i b^j \mid i, j \geq 0\} \; .$$

Such language is regular, since it is generated by the regular expression $\mathbf{a}^*\mathbf{b}^*$. Then we have the equality

$$\overline{L_{\neq}} \cap L_{ab} \quad = \quad L_= \; .$$

This last equality follows from our observation above that the strings in $\overline{L_{\neq}}$ are all strings formed by a certain number of a followed by the same number of b, but also all strings that have occurrences of symbols a and b mixed together, in a way that violates the pattern $a^i b^j$ imposed by L_{ab}. Since we are assuming that $\overline{L_{\neq}}$ is regular, and since we know that L_{ab} is also regular, language $L_{=}$ must also be regular: this follows from the closure property of the regular languages under the intersection operator. However, we have already shown in part (i) of this same exercise that $L_{=}$ is not regular, so we have obtained a contradiction. We must therefore conclude that $L_{\neq}$ is not a regular language.

Chapter 2

Context-Free Languages

Exercise 2.1 Specify a context-free grammar that generates the language

$$L \quad = \quad \{0^n 1^m \mid 0 \leq n < m\}.$$

Solution The grammar works by initially generating an equal number of 0 and 1 (zero or more) placed at the sides of the start symbol. To conclude the generation of a string, the grammar goes through an additional variable A that generates only symbols 1 (one or more). The required context-free grammar is therefore

$$G \quad = \quad (\{S, A\}, \{0, 1\}, P, S),$$

where P contains all and only the rules

$$\begin{array}{rcl} S & \rightarrow & 0S1 \mid A \\ A & \rightarrow & 1A \mid 1 \end{array}$$

In the general case, proving that $L(G) = L$ for a CFG G and a language L is very complex, the textbook mentions this problem in a special frame in Chapter 5. The technique used for such proofs is mutual induction, and the basic idea is the same we have studied for the case of DFAs; see for instance Exercise 1.1 in Chapter 1. In case of a DFA M, it is necessary to define properties $\mathcal{P}_q$ for each state q of M, and relate each property $\mathcal{P}_q$ with computations of M starting from the initial state and reaching q. In a similar way, in the case of a CFG G it is necessary to define a property $\mathcal{P}_A$ for each variable A of G and relate each property $\mathcal{P}_A$ to the derivations of G that start with A and generate a string entirely composed of terminal symbols. In case of the present exercise, G is very simple and uses only two variables, and therefore the proof of property $L(G) = L$ is relatively simple. We carry out the proof in the remaining part of this exercise.

Let us start with the definition of the properties associated with the two variables of G.

- For every $x \in \{0,1\}^*$, $\mathcal{P}_A(x)$ holds true if and only if x is a sequence composed by $n \geq 1$ occurrences of symbol 1.
- For every $x \in \{0,1\}^*$, $\mathcal{P}_S(x)$ holds true if and only if x is a sequence composed by $n \geq 0$ occurrences of symbol 0, followed by $m > n$ occurrences of symbol 1.

We now need to prove the following two statements.

(i) For every $x \in \{0,1\}^*$, $\mathcal{P}_A(x)$ holds true if and only if $A \overset{+}{\Rightarrow} x$.

(ii) For every $x \in \{0,1\}^*$, $\mathcal{P}_S(x)$ holds true if and only if $S \overset{+}{\Rightarrow} x$.

Finally, we observe that $\mathcal{P}_S$ matches the definition of language L. More precisely, for each string $x \in \{0,1\}^*$ we have that $\mathcal{P}_S(x)$ holds true if and only if $x \in L$. Thus statement (ii) above can be used to prove the property $L(G) = L$.

To complete the exercise, we must now prove statements (i) and (ii). Since the statements (i) and (ii) are of the form "if and only if", we must prove four statements, and for each statement we must provide the base case and the inductive case. The technique of mutual induction will be used in the proof of statement (ii), since the derivations that start with S and produce strings of terminal symbols use subderivations starting with A. Conversely, for the proof of statement (i) we only use simple induction, since the derivations starting with A and producing strings of terminal symbols do not depend on subderivations starting with S.

(Part (i), if) We must prove that if $A \overset{+}{\Rightarrow} x$ then $\mathcal{P}_A(x)$ holds true, for every $x \in \{0,1\}^*$. We proceed by induction over the length of the derivation.

Base. The shortest derivation of the form $A \overset{+}{\Rightarrow} x$ is $A \overset{1}{\Rightarrow} 1$ and has length of one. We have that x is a sequence composed by $n \geq 1$ symbols 1, and therefore $\mathcal{P}_A(1)$ holds true.

Induction. Let us now consider a derivation $A \overset{+}{\Rightarrow} x$ with length longer than one. This derivation must start with the production $A \to 1A$. Then we can write $A \overset{1}{\Rightarrow} 1A \overset{+}{\Rightarrow} 1y$, where $A \overset{+}{\Rightarrow} y$. By applying the inductive hypothesis to

$A \overset{+}{\Rightarrow} y$ we have $\mathcal{P}_A(y)$, that is, y is a sequence composed by $n \geq 1$ occurrences of symbol 1. It follows that $x = 1y$ is a sequence composed by $n \geq 2$ occurrences of symbols 1, and therefore $\mathcal{P}_A(x)$ holds true.

(Part (i), only if) We must prove that if $\mathcal{P}_A(x)$ holds true then $A \overset{+}{\Rightarrow} x$, for every $x \in \{0,1\}^*$. We proceed by induction on the length of x.

Base. Let $|x| = 1$. If $x = 1$, then $\mathcal{P}_A(1)$ holds true. The required derivation for x is $A \overset{1}{\Rightarrow} 1$. If $x = 0$, then $\mathcal{P}_A(0)$ is false. It follows that the statement "if $\mathcal{P}_A(0)$ is true then $A \overset{+}{\Rightarrow} 0$" holds true, since the antecedent is false.

Induction. Assuming $|x| > 1$, we consider the hypothesis that $\mathcal{P}_A(x)$ holds true and we prove $A \overset{+}{\Rightarrow} x$. We have that x is a sequence composed by $n > 1$ occurrences of symbols 1. We write $x = 1y$, with $y \neq \varepsilon$. Since y contains at least one occurrence of 1, $\mathcal{P}_A(y)$ holds true. Since $|y| \leq |x|$, we can apply the inductive hypothesis and conclude that $A \overset{+}{\Rightarrow} y$. Using the production $A \to 1A$, we can finally write $A \overset{1}{\Rightarrow} 1A \overset{+}{\Rightarrow} 1y = x$.

(Part (ii), if) We must show that if $S \overset{+}{\Rightarrow} x$ then $\mathcal{P}_S(x)$ holds true, for $x \in \{0,1\}^*$. We proceed by mutual induction on the length of the derivation.

Base. The shortest derivation of the form $S \overset{+}{\Rightarrow} x$ is $S \overset{1}{\Rightarrow} A \overset{1}{\Rightarrow} 1$ and has length of two. We have $x = 1 \in L$, and therefore $\mathcal{P}_S(1)$ holds true.

Induction. Let $S \overset{+}{\Rightarrow} x$ be a derivation of length greater than two. We distinguish two cases, based on the first production in $S \overset{+}{\Rightarrow} x$

- If the derivation $S \overset{+}{\Rightarrow} x$ starts with the production $S \to 0S1$, then we can write $S \overset{1}{\Rightarrow} 0S1 \overset{+}{\Rightarrow} 0y1 = x$. We thus have $S \overset{+}{\Rightarrow} y$. Applying the inductive hypothesis to $S \overset{+}{\Rightarrow} y$ we obtain that $\mathcal{P}_S(y)$ holds true, that is, y is a sequence composed by $n \geq 0$ occurrences of symbol 0, followed by $m > n$ occurrences of symbol 1. It follows that $x = 0y1$ is a sequence composed by $n \geq 1$ occurrences of symbol 0, followed by $m > n$ occurrences of symbol 1, and therefore $\mathcal{P}_S(x)$ holds true.

- If the derivation $S \overset{+}{\Rightarrow} x$ starts with the production $S \to A$ then we can write $S \overset{1}{\Rightarrow} A \overset{+}{\Rightarrow} x$. Since the derivation $A \overset{+}{\Rightarrow} x$ has one step less than the derivation $S \overset{+}{\Rightarrow} x$, we can apply the mutual induction to $A \overset{+}{\Rightarrow} x$, deriving that $\mathcal{P}_A(x)$ holds true, that is, x is a sequence composed by $n \geq 1$ occurrences of symbol 1. This also means that x is a sequence composed by $n \geq 0$ occurrences of symbol 0, followed by $m > n$ occurrences of symbol 1, and therefore we have that $\mathcal{P}_S(x)$ holds true.

(Part (ii), only if) We must prove that if $\mathcal{P}_S(x)$ holds true then $S \overset{+}{\Rightarrow} x$, for $x \in \{0,1\}^*$. We proceed by mutual induction on the length of x.

Base. Let $|x| = 1$. If $x = 1$, then $\mathcal{P}_S(x)$ holds true. The required derivation for x is $S \overset{1}{\Rightarrow} A \overset{1}{\Rightarrow} 1$. If instead $x = 0$, we have $\#_0(0) > \#_1(0) = 0$. Since the property $\mathcal{P}_S(x)$ requires $\#_1(x) > \#_0(x)$, we conclude that $\mathcal{P}_S(0)$ is false. It follows that the statement "if $\mathcal{P}_S(0)$ holds true then $S \overset{+}{\Rightarrow} 0$" is true, since the antecedent is false.

Induction. Let $|x| > 1$. We assume the hypothesis that $\mathcal{P}_S(x)$ holds true, and we prove $S \overset{+}{\Rightarrow} x$. We know that x is a sequence composed by $n \geq 0$ occurrences of symbol 0, followed by $m > n$ occurrences of symbol 1. we distinguish two cases, based on the first symbol in string x

- If x starts with 0, then the last symbol of x must be 1, otherwise we would have a violation of $\mathcal{P}_S(x)$. Thus we can write $x = 0y1$ for some string $y \in \{0,1\}^*$. Since $\#_0(y) = \#_0(x)-1$ and $\#_1(y) = \#_1(x)-1$, the string y is still a sequence composed by $n \geq 0$ occurrences of symbol 0, followed by $m > n$ occurrences of symbol 1, and then $\mathcal{P}_S(y)$ holds true. Since $|y| < |x|$, we can apply the inductive hypothesis to y and obtain the relation $S \overset{+}{\Rightarrow} y$. Using the production $S \to 0S1$, we can write $S \overset{1}{\Rightarrow} 0S1 \overset{+}{\Rightarrow} 0y1 = x$.

- If x starts with 1, we write $x = 1y$. We know that $y \neq \varepsilon$. We also know that there cannot be occurrences of 0 in y, otherwise we would have a violation of $\mathcal{P}_S(x)$. Then y is a sequence composed by $n \geq 1$ occurrences of symbols 1, and therefore $\mathcal{P}_A(y)$ holds true. Since $|y| < |x|$, we can apply the mutual induction on y and conclude that $A \overset{+}{\Rightarrow} y$. Using the productions $S \to A$ and $A \to 1A$, we can then write $S \overset{1}{\Rightarrow} A \overset{1}{\Rightarrow} 1A \overset{+}{\Rightarrow} 1y = x$, which provides the desired derivation for x.

Exercise 2.2 Consider the language

$$L = \{a^i b^j c^k \mid i,j,k \geq 1\}.$$

Specify a context-free grammar G that generates L, and use mutual induction to show the relation $L(G) = L$.

Solution The basic idea underlying our context-free grammar is to initially generate a run of a's through variable S, then switch to variable B to generate a run of b's, and finally switch to variable C to generate a run of c's. The required context-free grammar is therefore

$$G = (\{S, B, C\}, \{a, b, c\}, P, S),$$

where P consists of the following rules

$$\begin{array}{rcl} S & \rightarrow & aS \mid aB \\ B & \rightarrow & bB \mid bC \\ C & \rightarrow & cC \mid c \end{array}$$

To prove $L(G) = L$ we need to use mutual induction, as in Exercise 2.1. Let us start with the definition of the properties associated with the three variables of G. For every $w \in \{a, b, c\}^*$

- $\mathcal{P}_C(w)$ holds true if and only if w is a string of $k \geq 1$ occurrences of symbol c;
- $\mathcal{P}_B(w)$ holds true if and only if w is a string of $j \geq 1$ occurrences of symbol b, followed by $k \geq 1$ occurrences of symbol c;
- $\mathcal{P}_S(w)$ holds true if and only if w is a string of $i \geq 1$ occurrences of symbol a, followed by $j \geq 1$ occurrences of symbol b, followed by $k \geq 1$ occurrences of symbol c.

We now need to prove the following statements. For every $w \in \{a, b, c\}^*$

(i) $\mathcal{P}_C(w)$ if and only if $C \overset{+}{\Rightarrow} w$;

(ii) $\mathcal{P}_B(w)$ if and only if $B \overset{+}{\Rightarrow} w$;

(iii) $\mathcal{P}_S(w)$ if and only if $S \overset{+}{\Rightarrow} w$.

Since each of the above statements has the form "if and only if", we must prove six statements overall, and for each statement we must provide the base case and the inductive case. Finally, we observe that $\mathcal{P}_S$ matches the definition of language L, thus statement (iii) can be used to prove the desired property $L(G) = L$.

(Part (i), if) We must prove that if $C \overset{+}{\Rightarrow} w$ then $\mathcal{P}_C(w)$ holds true, for every $w \in \{a, b, c\}^*$. We proceed by induction over the length of the derivation.

Base. The shortest derivation of the form $C \overset{+}{\Rightarrow} w$ in our grammar G is $C \overset{1}{\Rightarrow} c$ and has length one. We have $w = c$, and it is immediate to see that $\mathcal{P}_C(w)$ holds true.

Induction. Let us now consider a derivation $C \overset{+}{\Rightarrow} w$ with length two or more. This derivation must start with the production $C \rightarrow cC$, and we can write $C \overset{1}{\Rightarrow} cC \overset{+}{\Rightarrow} cw'$. We can now use factorization and obtain the shorter derivation $C \overset{+}{\Rightarrow} w'$. By applying the inductive hypothesis to $C \overset{+}{\Rightarrow} w'$ we have $\mathcal{P}_C(w')$, that is, w' is a sequence composed by $k \geq 1$ occurrences of symbol c. It follows that $w = cw'$ is a sequence composed by $k \geq 2$ occurrences of symbol c, and therefore $\mathcal{P}_C(w)$ holds true.

(Part (i), only if) We must prove that if $\mathcal{P}_C(w)$ holds true then $C \overset{+}{\Rightarrow} w$, for every $w \in \{a, b, c\}^*$. We proceed by induction on the length of w.

Base. Let $|w| = 1$. If $w = c$, then $\mathcal{P}_C(w)$ holds true. The required derivation for w is then $C \overset{1}{\Rightarrow} c$. If $w = a$ or $w = b$, then $\mathcal{P}_C(w)$ is false and the implication is always true.

Induction. Assuming $|w| > 1$, we consider the hypothesis that $\mathcal{P}_C(w)$ holds true and we prove $C \overset{+}{\Rightarrow} w$. Under the above hypothesis, w is a sequence composed by $k > 1$ occurrences of symbol c. We can write $w = cw'$, with $|w'| > 0$. Since w' contains at least one occurrence of c, we have that $\mathcal{P}_C(w')$ holds true. Since $|w'| \leq |w|$, we can apply the inductive hypothesis and conclude that $C \overset{+}{\Rightarrow} w'$. Using the derivation $C \overset{1}{\Rightarrow} cC$ and composition, we can finally write $C \overset{1}{\Rightarrow} cC \overset{+}{\Rightarrow} cw' = w$.

(Part (ii), if) We must show that if $B \overset{+}{\Rightarrow} w$ then $\mathcal{P}_B(w)$ holds true, for any $w \in \{a, b, c\}^*$. We proceed by induction on the length of the derivation.

Base. The shortest derivation of the form $B \overset{+}{\Rightarrow} w$ in G is $B \overset{1}{\Rightarrow} bC \overset{1}{\Rightarrow} bc$ and has length of two. We can easily see that $w = bc$ and $\mathcal{P}_B(w)$ holds true.

Induction. Let $B \overset{+}{\Rightarrow} w$ be a derivation of length greater than two. We distinguish two cases, based on the choice of the first rule in the derivation.

- If the derivation $B \overset{+}{\Rightarrow} w$ starts with the rule $B \rightarrow bB$, then we can write $B \overset{1}{\Rightarrow} bB \overset{+}{\Rightarrow} bw' = w$. Using factorization we conclude that $B \overset{+}{\Rightarrow} w'$. Since the latter derivation is one unit shorter than our initial derivation, we can apply to $B \overset{+}{\Rightarrow} w'$ the inductive hypothesis for statement "(ii), if", and derive that $\mathcal{P}_B(w')$ holds true, that is, w' is a sequence composed by $j \geq 1$ occurrences of symbol b followed by $k \geq 1$ occurrences of symbol c. Therefore we have that $w = bw'$ has the same structure and $\mathcal{P}_B(w)$ holds true as well.

- If the derivation $B \overset{+}{\Rightarrow} w$ starts with the rule $B \rightarrow bC$, then we can write

$B \overset{1}{\Rightarrow} bC \overset{+}{\Rightarrow} bw' = w$. Again, we can use factorization and conclude that $C \overset{+}{\Rightarrow} w'$. Since the latter derivation is one unit shorter than our initial derivation, we can apply to $C \overset{+}{\Rightarrow} w'$ mutual induction for statement "(i), if", and derive that $\mathcal{P}_C(w')$ holds true, meaning that w' is a sequence composed by $k \geq 1$ occurrences of symbol c. This in turn means that $w = bw'$ is a sequence composed by $j \geq 1$ occurrences of symbol b followed by $k \geq 1$ occurrences of symbol c, and thus $\mathcal{P}_B(w)$ holds true.

(Part (ii), only if) We must prove that if $\mathcal{P}_B(w)$ holds true then $B \overset{+}{\Rightarrow} w$, for any $x \in \{a, b, c\}^*$. We proceed by mutual induction on the length of w.

Base. The shortest string w such that $\mathcal{P}_B(w)$ holds true is $w = bc$. The required derivation in G for w is $B \overset{1}{\Rightarrow} bC \overset{1}{\Rightarrow} bc$. It is easy to see that no other string of length 2 satisfies property $\mathcal{P}_B()$.

Induction. Let $|w| > 2$. We work under the hypothesis that $\mathcal{P}_B(w)$ holds true, and we prove $B \overset{+}{\Rightarrow} w$. From our hypothesis we have that w is a sequence composed by $j \geq 1$ occurrences of symbol b followed by $k \geq 1$ occurrences of symbol c. We distinguish two cases, based on whether the second symbol in string w is b or c.

- If w has a prefix bb, then we can write $w = bw'$ and claim that w' is a sequence composed by $j \geq 1$ occurrences of symbol b followed by $k \geq 1$ occurrences of symbol c. This means that $\mathcal{P}_B(w')$ holds true. Since $|w'| < |w|$, we apply on w' induction for statement "(ii), only if", and we conclude that $B \overset{+}{\Rightarrow} w'$. Then we can compose the desired derivation $B \overset{1}{\Rightarrow} bB \overset{+}{\Rightarrow} bw' = w$.

- If w has a prefix bc, then we can write $w = bw'$ and claim that w' is a sequence composed by $k \geq 1$ occurrences of symbol c. This means that $\mathcal{P}_C(w')$ holds true. Since $|w'| < |w|$, we apply on w' mutual induction for statement "(i), only if", and we conclude that $C \overset{+}{\Rightarrow} w'$. Then we can compose the desired derivation $B \overset{1}{\Rightarrow} bC \overset{+}{\Rightarrow} bw' = w$.

(Part (iii), if) We must show that if $S \overset{+}{\Rightarrow} w$ then $\mathcal{P}_S(w)$ holds true, for any $w \in \{a, b, c\}^*$. We use mutual induction on the length of the derivation.

Base. The shortest derivation of the form $S \overset{+}{\Rightarrow} w$ is $S \overset{1}{\Rightarrow} aB \overset{1}{\Rightarrow} abC \overset{1}{\Rightarrow} abc$, with length three. We can easily see that $\mathcal{P}_S(w)$ holds true for $w = abc$.

Induction. Let $S \overset{+}{\Rightarrow} w$ be a derivation of length greater than three. We distinguish two cases, based on the first rule in the derivation.

- If the derivation $S \overset{+}{\Rightarrow} w$ starts with rule $S \to aS$, then we can write $S \overset{1}{\Rightarrow} aS \overset{+}{\Rightarrow} aw' = w$. Factorizing the derivation we obtain a shorter derivation $S \overset{+}{\Rightarrow} w'$. We can thus apply the inductive hypothesis, and conclude that $\mathcal{P}_S(w')$ holds true, that is, w' is a sequence composed by $i \geq 1$ occurrences of symbol a, followed by $j \geq 1$ occurrences of symbol b, and followed by $k \geq 1$ occurrences of symbol c. From $w = aw'$, we can easily see that w has the same property, and then $\mathcal{P}_S(w)$ holds true.
- If the derivation $S \overset{+}{\Rightarrow} w$ starts with the production $S \to aB$, then we can write $S \overset{1}{\Rightarrow} aB \overset{+}{\Rightarrow} aw' = w$. Again, we factorize the derivation, and obtain a shorter derivation $B \overset{+}{\Rightarrow} w'$. We can then apply to derivation $B \overset{+}{\Rightarrow} w'$ the mutual induction hypothesis for statement "(ii), if", deriving that $\mathcal{P}_B(w')$ holds true. This means that w is a sequence composed by $j \geq 1$ occurrences of symbol b followed by $k \geq 1$ occurrences of symbol c. Then $w = aw'$ is a sequence composed by one occurrence of symbol a, followed by $j \geq 1$ occurrences of symbol b, and followed by $k \geq 1$ occurrences of symbol c. This in turn means that $\mathcal{P}_S(w)$ holds true.

(Part (iii), only if) We now prove that if $\mathcal{P}_S(w)$ holds true then $S \overset{+}{\Rightarrow} w$, for $w \in \{a, b, c\}^*$. We proceed by mutual induction on the length of w.

Base. The shortest string in $\{a, b, c\}^*$ such that $\mathcal{P}_S(w)$ holds true is $w = abc$. The required derivation for w is $S \overset{1}{\Rightarrow} aB \overset{1}{\Rightarrow} abC \overset{1}{\Rightarrow} abc$. For all other strings $w \in \{a, b, c\}^*$ with length 3 we have that $\mathcal{P}_S(w)$ is false, and then the desired implication is true.

Induction. We now consider strings in $\{a, b, c\}^*$ with $|w| > 3$ such that $\mathcal{P}_S(w)$ holds true, and we prove $S \overset{+}{\Rightarrow} w$. From our hypothesis we have that w is a sequence composed by $i \geq 1$ occurrences of symbol a, followed by $j \geq 1$ occurrences of symbol b, followed by $k \geq 1$ occurrences of symbol c. We distinguish two cases, based on whether the second symbol in string w is a or b.

- If w has a prefix aa, then we can write $w = aw'$ and claim that w' is a sequence composed by $i \geq 1$ occurrences of symbol a, followed by $j \geq 1$ occurrences of symbol b, followed by $k \geq 1$ occurrences of symbol c. This means that $\mathcal{P}_S(w')$ holds true. Since $|w'| < |w|$, we apply induction on w', and conclude that $S \overset{+}{\Rightarrow} w'$. Then we can compose the desired derivation $S \overset{1}{\Rightarrow} aS \overset{+}{\Rightarrow} aw' = w$.

- If w has a prefix ab, then we can write $w = aw'$ and claim that w' is a sequence composed by $j \geq 1$ occurrences of symbol b, followed by $k \geq 1$ occurrences of symbol c. This means that $\mathcal{P}_B(w')$ holds true. Since $|w'| < |w|$, we apply on w' mutual induction for statement "(ii), only if", and we conclude that $B \overset{+}{\Rightarrow} w'$. Then we can compose the desired derivation $S \overset{1}{\Rightarrow} aB \overset{+}{\Rightarrow} aw' = w$.

This concludes the proof that $L(G) = L$.

For the purpose of gaining a better understanding of the mutual induction technique, as applied to this typology of problems, we now explore an alternative solution for this exercise. More precisely, we develop a different context-free grammar G' for generating our language L. The basic idea underlying G' is to generate the three runs of a's, b's and c's by means of three, independent variables A, B, and C, respectively. The new context-free grammar is specified as

$$G' \quad = \quad (\{S, A, B, C\}, \{a, b, c\}, P, S),$$

where P consists of the following rules

$$\begin{array}{rcl} S & \rightarrow & ABC \\ A & \rightarrow & aA \mid a \\ B & \rightarrow & bB \mid b \\ C & \rightarrow & cC \mid c \end{array}$$

Although G' has some more variables and rules than G, it is somehow simpler than G since the variables A, B, and C are all mutually independent, and the rules associated with these variables all work in the same way.

To prove the relation $L(G') = L$, we define the properties associated with the variables of G'. For every $w \in \{a, b, c\}^*$

- $\mathcal{P}_A(w)$ holds true if and only if w is a string of $i \geq 1$ occurrences of symbol a;

- $\mathcal{P}_B(w)$ holds true if and only if w is a string of $j \geq 1$ occurrences of symbol b;

- $\mathcal{P}_C(w)$ holds true if and only if w is a string of $k \geq 1$ occurrences of symbol c;
- $\mathcal{P}_S(w)$ holds true if and only if w is a string of $i \geq 1$ occurrences of symbol a, followed by $j \geq 1$ occurrences of symbol b, followed by $k \geq 1$ occurrences of symbol c.

We now need to prove the following statements. For every $w \in \{a, b, c\}^*$

(i) $\mathcal{P}_A(w)$ if and only if $A \overset{+}{\Rightarrow} w$;

(ii) $\mathcal{P}_B(w)$ if and only if $B \overset{+}{\Rightarrow} w$;

(iii) $\mathcal{P}_C(w)$ if and only if $C \overset{+}{\Rightarrow} w$;

(iv) $\mathcal{P}_S(w)$ if and only if $S \overset{+}{\Rightarrow} w$.

Again, we observe that $\mathcal{P}_S$ matches the definition of language L, thus statement (iv) can be used to prove the desired property $L(G) = L$.

We now observe that the rules for variable C in the two grammars G and G' are exactly the same. This means that the proof of statement (i) for G and the proof of statement (iii) for G' are the same. We therefore omit below the proof of statement (iii) for G'.

Furthermore, we observe that the proofs of statements (i), (ii) and (iii) for G' must be very similar, since the rules for the three variables A, B and C in G' have the same basic structure. This means that we can omit below the proofs of statements (i), (ii) as well. We are therefore left with the only proof of statement (iv).

(Part (iv), if) We must show that if $S \overset{+}{\Rightarrow} w$ then $\mathcal{P}_S(w)$ holds true, for any $w \in \{a, b, c\}^*$. We use mutual induction on the length of the derivation.

Base. The shortest derivations in G' having the form $S \overset{+}{\Rightarrow} w$ need to use four steps. There are six such derivations, reported below, and they all generate the same string $w = abc$:

$$
\begin{aligned}
&S \overset{1}{\Rightarrow} ABC \overset{1}{\Rightarrow} aBC \overset{1}{\Rightarrow} abC \overset{1}{\Rightarrow} abc \ , \\
&S \overset{1}{\Rightarrow} ABC \overset{1}{\Rightarrow} aBC \overset{1}{\Rightarrow} aBc \overset{1}{\Rightarrow} abc \ , \\
&S \overset{1}{\Rightarrow} ABC \overset{1}{\Rightarrow} AbC \overset{1}{\Rightarrow} abC \overset{1}{\Rightarrow} abc \ , \\
&S \overset{1}{\Rightarrow} ABC \overset{1}{\Rightarrow} AbC \overset{1}{\Rightarrow} Abc \overset{1}{\Rightarrow} abc \ , \\
&S \overset{1}{\Rightarrow} ABC \overset{1}{\Rightarrow} ABc \overset{1}{\Rightarrow} aBc \overset{1}{\Rightarrow} abc \ ,
\end{aligned}
$$

$$S \overset{1}{\Rightarrow} ABC \overset{1}{\Rightarrow} ABc \overset{1}{\Rightarrow} Abc \overset{1}{\Rightarrow} abc \; .$$

It is immediate to see that $\mathcal{P}_S(w)$ holds true for $w = abc$.

Induction. Let $S \overset{+}{\Rightarrow} w$ be a derivation of length greater than four. Since there is only one rule in G' for the variable S, the derivation must have the form $S \overset{1}{\Rightarrow} ABC \overset{+}{\Rightarrow} w$. We now factorize this derivation into three derivations, one for each of the three variables A, B and C:

$$A \overset{+}{\Rightarrow} w_1 \; ,$$
$$B \overset{+}{\Rightarrow} w_2 \; ,$$
$$C \overset{+}{\Rightarrow} w_3 \; ,$$

for strings w_1, w_2 and w_3 such that $w_1 w_2 w_3 = w$. We remark that at this point we *cannot say anything* about the terminal symbols appearing in w_1, w_2 and w_3.

Assume now that ℓ is the length of the original derivation $S \overset{1}{\Rightarrow} ABC \overset{+}{\Rightarrow} w$. We observe that the sum of the length of the three derivations above must be $\ell - 1$. To see this, consider that each derivation step in the original derivation appears in exactly one of the three derivations above, with the only exception of the derivation step $S \overset{1}{\Rightarrow} ABC$, which is omitted. Since all of the three derivations above have length shorter than ℓ, we can apply mutual induction to each of them.

Starting with derivation $A \overset{+}{\Rightarrow} w_1$, we use mutual induction for statement "(i), only if" and obtain that $\mathcal{P}_A(w_1)$ holds true. Similarly, from $B \overset{+}{\Rightarrow} w_2$ and statement "(ii), only if" we obtain $\mathcal{P}_B(w_2)$, and from $C \overset{+}{\Rightarrow} w_3$ and statement "(iii), only if" we obtain $\mathcal{P}_C(w_3)$. From these properties and from $w = w_1 w_2 w_3$, we can immediately derive that w is a string of $i \geq 1$ occurrences of symbol a, followed by $j \geq 1$ occurrences of symbol b, followed by $k \geq 1$ occurrences of symbol c. This in turns means that $\mathcal{P}_S(w)$ holds true.

(Part (iv), only if) We now prove that if $\mathcal{P}_S(w)$ holds true then $S \overset{+}{\Rightarrow} w$ in G', for every $w \in \{a, b, c\}^*$. We proceed by mutual induction on the length of w.

Base. The shortest string such that $\mathcal{P}_S(w)$ holds true is $w = abc$. One possible derivation for w, among several ones, is $S \overset{1}{\Rightarrow} ABC \overset{1}{\Rightarrow} aBC \overset{1}{\Rightarrow} abC \overset{1}{\Rightarrow} abc$. For all other strings $w \in \{a, b, c\}^*$ with length 3 we have that $\mathcal{P}_S(w)$ is false, and then the desired implication is true.

Induction. We now consider strings in $\{a, b, c\}^*$ with $|w| > 3$ such that $\mathcal{P}_S(w)$ holds true, and we show $S \overset{+}{\Rightarrow} w$. From the definition of $\mathcal{P}_S(w)$, we have that w is a sequence composed by $i \geq 1$ occurrences of symbol a, followed by $j \geq 1$ occurrences of symbol b, followed by $k \geq 1$ occurrences of symbol c. It follows that we can factorize w as $w = w_1w_2w_3$, such that w_1 is a sequence composed by $i \geq 1$ occurrences of symbol a, w_2 is a sequence composed by $j \geq 1$ occurrences of symbol b, and w_3 is a sequence composed by $k \geq 1$ occurrences of symbol c. By definition, we have that $\mathcal{P}_A(w_1)$, $\mathcal{P}_B(w_2)$, and $\mathcal{P}_C(w_3)$ are all true.

We observe that $|w_1| + |w_2| + |w_3| = |w|$ and $|w_i| \geq 1$, $1 \leq i \leq 3$. We can easily conclude that $|w_i| < |w|$, $1 \leq i \leq 3$. This means that we can apply mutual induction to each string w_i. Since $\mathcal{P}_A(w_1)$, we apply mutual induction for statement "(i), if" and obtain $A \overset{+}{\Rightarrow} w_1$. Similarly, from $\mathcal{P}_B(w_2)$ and statement "(ii), if" we get $B \overset{+}{\Rightarrow} w_2$, and from $\mathcal{P}_C(w_3)$ and statement "(ii), if" we get $C \overset{+}{\Rightarrow} w_3$. Composing these three derivations with derivation $S \overset{1}{\Rightarrow} ABC$, we obtain the derivation

$$S \overset{1}{\Rightarrow} ABC \overset{+}{\Rightarrow} w_1BC \overset{+}{\Rightarrow} w_1w_2C \overset{+}{\Rightarrow} w_1w_2w_3 = w$$

which has the desired form $S \overset{+}{\Rightarrow} w$. This concludes the proof that $L(G') = L$.

Exercise 2.3 Using the pumping lemma for CFL, prove that the language

$$L = \{0^i1^j \mid i, j \geq 1,\ j = i^2\}$$

is not a CFL.

Solution Let us suppose that L is a CFL. Then L must satisfy the pumping lemma for context-free languages, and there exits a number n such that, given a string $z \in L$ with $|z| \geq n$, we can write $z = uvwxy$ and assert that:

(i) $|v| + |x| \geq 1$;

(ii) $|vwx| \leq n$;

(iii) $uv^kwx^ky \in L$ for each natural number $k \geq 0$.

The string $z = 0^n1^{n^2}$ is in L and satisfies the hypotheses of the pumping lemma, since $|z| = n + n^2 \geq n$. However, as we will show below, there is no choice of vwx in z such that $uv^kwx^ky \in L$ for any choice of k. In the following we viwe the string z as composed by two blocks: a block with n occurrences of symbol 0, and a block with n^2 occurrences of symbol 1. We now analyze four possible cases, according to the position of vwx relative to these two blocks.

- vwx is entirely contained in the block composed by 0. For $k = 0$ the number of occurrences of 0 decreases to a value smaller than n, since $vx \neq \varepsilon$, while the number of occurrences of 1 remains unchanged. Therefore the resulting string uwy does not belong to L.

- vwx is entirely contained in the block consisting of 1. This case is solved in a similar way to the previous case, by taking k =0.

- If v or x contains occurrences of both 0 and 1, then for k =2 the resulting string is no longer formed by a block of 0 followed by a block of 1. Therefore the string uv^2wx^2y does not belong to L

- The last case takes into consideration the possibility that v consists only of occurrences of 0, and x consists only of occurrences of 1, and both of these strings are different from the empty string ε. For a generic $k > 1$, let us consider the string $z_k = uv^kwx^ky$. The number of 0 in z_k is $n + (k-1)|v|$ and the number of 1 in z_k is $n^2 + (k-1)|x|$. In order for z_k to belong to L we must have

  $$(n + (k-1)|v|)^2 = n^2 + (k-1)|x|,$$

  that is

  $$2n(k-1)|v| + (k-1)^2|v|^2 = (k-1)|x|.$$

  We observe now that on the left-hand side of the second equation we have a term that grows quadratically in k, while on the right we have a term that grows linearly in k. It follows that the equality cannot be verified for every value of k.

Having identified at least one value of k for which $uv^kwx^ky \notin L$, for every possible factorization of the string z, we conclude that L is not a context-free language.

Exercise 2.4 Consider the following language

$$L \;=\; \{ww^Rw \mid w \in \{0,1\}^*\}.$$

Produce a PDA that recognizes L or else rigorously prove that L is not a context-free language.

Solution Following the first suggestion of the exercise, the first thing we could try is the construction of a PDA that recognizes L. Considering such construction, however, we soon realize that a nondeterministic PDA could store w in the stack and then read w^R, matching the latter against the stack content. However, after this process the stack is emptied and the information about w is no longer available for the third part of the computation. In fact, the language L is not context-free, as shown below.

Let us suppose that L is a context-free language. Then the pumping lemma for context-free languages should be satisfied. This in turn means that there exists a positive integer n such that, given a string $z \in L$ with $|z| \geq n$, we can write $z = uvwxy$ such that:

(i) $|v| + |x| \geq 1$;

(ii) $|vwx| \leq n$;

(iii) $uv^iwx^iy \in L$ for every $i \geq 0$.

The string $z = 0^n1^{2n}0^{2n}1^n$ is in L, since $z = ss^Rs$, with $s = 0^n1^n$, and it satisfies the hypotheses of pumping lemma, since $|z| = 6n > n$. However, as we will show below, there is no choice of vwx within z such that $uv^iwx^iy \in L$ for any choice of i. First of all, let us view string z as composed of the blocks shown in Figure 2.1. In what follows, we will also call *block* of length k any substring of z composed of k equal characters.

$$z \;=\; \underbrace{\underbrace{0\cdots0}_{\text{I}}\underbrace{1\cdots1}_{\text{II}}}_{s}\;\underbrace{\underbrace{1\cdots1}_{\text{III}}\underbrace{0\cdots0}_{\text{IV}}}_{s^R}\;\underbrace{\underbrace{0\cdots0}_{\text{V}}\underbrace{1\cdots1}_{\text{VI}}}_{s}$$

Figure 2.1: Block composition for string z.

There are several places within string z in which vwx can be located. However, keeping in mind that z consists of 6 blocks of length n, and that the length of vwx cannot be larger than the length of anyone of these blocks, we need to consider only 2 possible scenarios, discussed below.

(i) vwx is located within a single block. If we choose $i = 0$, that is, if we delete the strings v and x from our factorization of z, then the resulting string z' differs from z by only a single block. We have the following possible configurations

$$\begin{aligned} z' &= 0^j 1^n 1^n 0^n 1^n, \\ z' &= 0^n 1^j 1^n 0^n 0^n 1^n = 0^n 1^j 0^n 0^n 1^n, \\ z' &= 0^n 1^n 1^n 0^j 0^n 1^n = 0^n 1^n 1^n 0^j 1^n, \\ z' &= 0^n 1^n 1^n 0^n 1^j, \end{aligned}$$

where $j < n$. For none of the above configurations it is possible to view z' as composed by three parts, necessarily of equal length, so that the definition of L is satisfied.

(ii) vwx is located between 2 blocks. In this case, since $|vwx| \leq n$, vwx can only span over two consecutive blocks. If we choose $i = 0$, we can have the following possible configurations for the resulting string z'', depending on the position of vwx:

$$\begin{aligned} z'' &= 0^j 1^k 1^n 0^n 1^n, \\ z'' &= 0^n 1^j 1^k 0^n 1^n, \\ z'' &= 0^n 1^j 0^k 0^n 1^n, \\ z'' &= 0^n 1^n 1^j 0^k 1^n, \\ z'' &= 0^n 1^n 1^n 0^j 1^k, \end{aligned}$$

where j and k are natural numbers such that $j + k < 2n$. Again, for any of the above choices, we have $z'' \notin L$.

To summarize the above discussion, for each possible factorization of z there is at least one value of i that violates the pumping lemma. We can therefore conclude that L is not context-free.

Exercise 2.5 Assess whether the following languages are context-free, and justify your answer:

(i) $L_1 = \{a^p b^q a^p b^q \mid p, q \geq 1\}$;

(ii) $L_2 = \{a^p b^q a^q b^p \mid p, q \geq 1\}$.

Solution The language L_1 is not context-free. An intuitive justification of this fact can be derived by considering how a PDA works. We observe that a PDA can store one or more non-negative integers by recording in its stack an appropriate number of symbols, in some order. Later on, the PDA can access the stored integers by extracting the associated representation from the stack, in the reverse order. In the case under consideration, p and q are stored in the stack, but p is inaccessible to the PDA because it is stored "below q".

A mathematical proof that L_1 is not context-free can be provided using the pumping lemma for context-free languages. As usual, we start by assuming that L_1 is a context-free language, and then we derive a contraddiction. If L_1 is a context-free language, there must be a number n, dependent on L_1, such that, for an arbitrary string $z \in L_1$ with $|z| \geq n$, we can write $z = uvwxy$ and assert that:

(i) $|v| + |x| \geq 1$;

(ii) $|vwx| \leq n$;

(iii) $uv^kwx^ky \in L_1$ for each natural number $k \geq 0$.

We then choose $z = a^nb^na^nb^n \in L_1$. To be used later, we name each of the blocks of the string z as shown in Figure 2.2.

$$z \;=\; \underbrace{a \cdots a}_{\text{I}}\underbrace{b \cdots b}_{\text{II}}\underbrace{a \cdots a}_{\text{III}}\underbrace{b \cdots b}_{\text{IV}}$$

Figure 2.2: Factorization of the string $z = a^nb^na^nb^n$ into named blocks.

The string vwx can be placed at several positions within z. However, we show below that none of these choices satisfies the pumping lemma. We distinguish the following possible cases.

- The string vwx consists only of occurrences of symbol a. Since $|vwx| \leq n$ and since the blocks shown in Figure 2.2 all have length n, we conclude that all of the a's in vwx must necessarily belong to a single block. We now choose $k = 0$, i.e., we delete v and x from z. The resulting string $z' = uwy$ has blocks I and III of different lengths, contrary to the definition of L_1.

- The string vwx consists only of occurrences of symbol b. This case is similar to the previous one: deleting v and x, the resulting string $z' = uwy$ has blocks II and IV of different lengths, and cannot belong to L_1.

- The string vwx contains occurrences of both a and b. In this case, since $|vwx| \leq n$, the a block and the b block that are involved must be consecutive in z. By choosing $k = 0$, these two blocks get shorter, while the other two blocks retain their original length. Also in this case, then, the resulting string cannot be divided into two equal parts, as require by the definition of L_1.

We therefore conclude that L_1 cannot be context-free.

The language L_2, on the other hand, is context-free. In order to prove this fact, we provide a CFG G and prove that $L(G) = L_2$. The grammar is specified as $G = (V, T, P, S)$, where $V = \{S, A\}$ is the set of variables and $T = \{a, b\}$ is the set of terminal symbols. The productions contained in P are as follows:

$$\begin{array}{lllll} 1. & S & \rightarrow & aSb \\ 2. & S & \rightarrow & aAb \\ 3. & A & \rightarrow & bAa \\ 4. & A & \rightarrow & ba \end{array}$$

To rigorously prove the equivalence $L(G) = L_2$, we should proceed using the mutual induction technique, as in Exercise 2.1. We have already observed that carrying out this type of demonstration is quite laborious, also for grammars with a small number of variables. We therefore omit this part, in order not to lengthen too much the presentation of the exercise.

Exercise 2.6 State whether the following languages are context-free, and motivate your answer:

(i) $L_1 = \{ww \mid w \in \{a, b\}^*\}$;

(ii) $L_2 = L_1 \cdot \{a, b\}^*$.

Solution The language L_1 is not context-free. In order to get an intuitive idea of this claim, try to think of a push-down automaton M that attempts to recognize L_1. This automaton will be nondeterministic, since there is no special symbol that indicates where to finish the first part of the input string and start the second one. Such automaton can certainly store in the stack the sequence of a's and b's representing w, but this sequence can then be retrieved from the stack only in the reverse order; in other words, you can write w on the stack, but you can only read w^R from the stack.

We now provide a rigorous proof of the fact that L_1 is not a context-free language. We initially assume that L_1 is a context-free language, and then apply the pumping lemma for this class of languages, showing that we reach a contradiction. To apply the pumping lemma, we could for instance choose the string $z = a^{2n}b^{2n}a^{2n}b^{2n}$, which belongs to L_1 since $z = ww$ with $w = a^{2n}b^{2n}$. Following this line, however, we can very soon realize that the factorization of z according to the pumping lemma requires the analysis of many possible cases, and for some cases the analysis is quite complex. It is also unclear which other strings of the language L_1 to choose, as an alternative, in order to simplify the overall analysis. The problem with the complexity of the analysis of L_1 according to the pumping lemma is due to the fact that the structure of the strings in L_1 is not strongly restricted, in the sense that for a string $ww \in L_1$ we do not have specific restrictions on the substring w, that may then be used to simplify the analysis. To prove that L_1 is not a context-free language, we then choose a different strategy, combining the application of the pumping lemma with some of the closure properties of context-free languages, in order to avoid the *direct* application of the pumping lemma to L_1.

Again, let us assume that L_1 is a context-free language. The language must therefore satisfy the closure property with the regular languages. In particular, let L' be the language generated by the regular expression $\mathbf{a}^*\mathbf{b}^*\mathbf{a}^*\mathbf{b}^*$. Then the language $L'_1 = L_1 \cap L'$ must still be a context-free language. It is not difficult to see that we can define the language L'_1 as

$$L'_1 \quad = \quad \{a^p b^q a^p b^q \mid p, q \geq 0\} \; .$$

Now the strings of L'_1 have a much more restricted form than the strings of L_1, and the application to L'_1 of the pumping lemma for context-free languages is simpler than in the case of L_1. Actually, we have already encountered language L'_1 at item (i) of Exercise 2.5, and we have already shown using the pumping lemma that such language is not context-free.

To summarize, we have assumed that language L_1 is a context-free language, we have applied the closure property of context-free languages with respect to the intersection with regular languages, and finally we have applied the pumping lemma for the class of context-free languages. All of this has led us to a contraddiction. We must therefore conclude that L_1 cannot be context-free.

The language $L_2 = L_1 \cdot \{a, b\}^*$ is instead a regular language, and therefore also a context-free language: this answers the question in item (ii) of the exercise. To show that L_2 is a regular language, we now provide a rigorous proof that $L_2 = \{a, b\}^*$.

First of all, we definitely have $L_2 \subseteq \{a, b\}^*$, since $\{a, b\}^*$ contains all possible strings over alphabet $\Sigma = \{a, b\}$ and, among them, all those in L_2.

Let us now show that $\{a, b\}^* \subseteq L_2$: we will then be able to conclude that $L_2 = \{a, b\}^*$. Let w be an arbitrary string in $\{a, b\}^*$. Since $\varepsilon \in L_1$, we can factorize w as $w = \varepsilon \cdot w,$, that is, w is the concatenation of a string in L_1 and a string in $\{a, b\}^*$. Then we have $w \in L_2$.

Exercise 2.7 Consider the language

$$L = \{w \mid w \in \{a, b, c\}^*,\ \#_a(w) = \#_b(w) = \#_c(w) \geq 1\} \ .$$

Prove that L is not a context-free language.

Solution We solve this exercise by applying to L the pumping for context-free languages. The pumping lemma asserts that if L is a context-free language then there exists a constant n, depending only on L, such that for every string $z \in L$ with length greater than or equal to n, z can always be factorized in the form $z = uvwxy$, such that $|vx| \geq 1$, $|vwx| \leq n$, and $uv^iwx^iy \in L$ for every $i \geq 0$.

Let us start by assuming that L is a context-free language, and let n be the constant of the pumping lemma. We choose the string $z = a^nb^nc^n$. This string certainly belongs to L, since it contains an equal number of a's, b's and c's, and in addition $|z| = 3n \geq n$. We now look for a way to write z as a concatenation of five strings u, v, w, x, y that satisfy the three requirements of the pumping lemma.

Our string z is formed by three consecutive blocks of length n, where each block is entirely composed by occurrences of only one symbol. Let us consider the string vwx: since its length must be at most n, there are only two possible scenarios, discussed below.

- The string vwx is placed entirely within one of the three blocks. We consider here only the case in which vwx is placed within the block of a's, as shown in Figure 2.3: the remaining two cases can be treated in a similar way. If we set $i = 0$, we have that the resulting string uwy will have fewer occurrences of a than occurrences of b and c, and therefore it will not belong to L.

$$\underbrace{aaaaa}_{u}\,\underbrace{aaaaaaaa}_{vwx}\,\underbrace{aabbbbbbbbbbbbbbbccccccccccccccc}_{y}$$

$$\underbrace{aaaaaaaaaaaaaaabb}_{u}\,\underbrace{bbbbbbbbbb}_{vwx}\,\underbrace{bbbccccccccccccccc}_{y}$$

$$\underbrace{aaaaaaaaaaaaaaabbbbbbbbbbbbbbbccc}_{u}\,\underbrace{cccccc}_{vwx}\,\underbrace{cccccc}_{y}$$

Figure 2.3: Placement of string vwx within the first block of $a^n b^n c^n$.

$$\underbrace{aaaaaaa}_{u}\,\underbrace{aaaaaaaabbbb}_{vwx}\,\underbrace{bbbbbbbbbbbccccccccccccccc}_{y}$$

$$\underbrace{aaaaaaaaaaaaaaabbbbbbbbbb}_{u}\,\underbrace{bbbbbcccccc}_{vwx}\,\underbrace{ccccccccc}_{y}$$

Figure 2.4: Placement of string vwx across the first two blocks of $a^n b^n c^n$.

- The string vwx is placed across two adjacent blocks. We observe that vwx cannot be placed across all of the three blocks, because otherwise vwx would have length greater than n. We consider here only the case in which vwx is placed across the block of a's and the block of b's, as shown in Figure 2.4: the remaining case can be treated in a similar way. Again, let us set $i = 0$. As we can see from Figure 2.4, even if our factorization preserves the balance between the occurrences of a and b in the resulting string uwy, there will be a mismatch with the the occurrences of c, and the string will not belong to L.

The fact that we have not been able to find a way to factorize the string z in the form $uvwxy$ in such a way that the conditions imposed by the pumping lemma are satisfied leads us to the conclusion that L cannot be a context-free language.

Exercise 2.8 State the truth or falsehood of the following statements, providing a rigorous proof.

(i) Given two non-regular languages L_1 and L_2, the language $L_1 \cup L_2$ cannot be regular.

(ii) Given a context-free language L_1, each subset of L_1 is still a context-free language.

(iii) Given a context-free language L_1 and given a regular language L_2, the language $L_1 \cap L_2$ could be a regular language.

(iv) Given a context-free language L_1 and given a regular language L_2, the language $L_1 \cap L_2$ could be a non-regular language.

Solution Statement (i) is false. Let us consider as a counterexample the two following languages, both defined over the alphabet $\Sigma = \{a, b\}$

$$\begin{array}{rcl} L_1 & = & \{w \mid w = a^n b^n,\ n \geq 0\} \\ L_2 & = & \{w \mid w = a^n b^m,\ n, m \geq 0,\ n \neq m\} \end{array}$$

L_1 is a well-known non-regular language. L_2 is also a non-regular language, as we have already seen in Exercise 1.6.

It is not difficult to see that we can define the union of L_1 and L_2 as

$$L_1 \cup L_2 \quad = \quad \{w \mid w = a^n b^m,\ n, m \geq 0\}$$

The language $L_1 \cup L_2$ can be generated by the regular expression $\mathbf{a}^*\mathbf{b}^*$ and is therefore a regular language. Thus L_1 and L_2 are a counterexample to statement (i).

Statement (ii) is false. Let us consider as a counterexample the alphabet $\Sigma = \{a, b, c\}$ and the language $L_1 = \Sigma^*$, which is a regular language and therefore also a context-free language. Let us now consider the language

$$L \quad = \quad \{w \mid a^n b^n c^n,\ n \geq 0\}$$

for which we have $L \subseteq L_1$. Language L is not a context-free language, as can be easily proved using the pumping lemma for context-free languages. L is therefore a counterexample to statement (ii).

Statement (iii) is true. To show this, we provide a simple example. Consider the alphabet $\Sigma = \{a, b\}$ and the following two languages

$$\begin{array}{rcl} L_1 & = & \{w \mid a^n b^m,\ n, m \geq 0\} \\ L_2 & = & \Sigma^* \end{array}$$

L_1 is a regular language, as already seen in point (ii), and therefore also a context-free language. L_2 is definitely a regular language. We also have the relationship $L_1 \cap L_2 = L_1$, which proves statement (iii).

Statement (iv) is true. To show this, we again provide a simple example. Consider the alphabet $\Sigma = \{a, b\}$ and the two languages

$$\begin{array}{rcl} L_1 & = & \{w \mid a^n b^n,\ n \geq 0\} \\ L_2 & = & \Sigma^* \end{array}$$

L_1 is a well-known context-free language but not a regular language, and L_2 is definitely a regular language. It is easy to see that we have $L_1 \cap L_2 = L_1$, which proves statement (iv).

Exercise 2.9 Consider the operator P defined as follows. For each language L

$$P(L) \quad = \quad \{x \mid x \in L, |x| = 2n,\ n \in \mathbb{N}\}\ .$$

Is the class of context-free languages closed with respect to operator P?

Solution To solve this exercise we have to show that, for each context-free language L, $P(L)$ is still a context-free language. To prove this property we can take advantage of the fact that the intersection of a context-free language with a regular language is still a context-free language. In our case, the language $P(L)$ consists of all the strings from language L having even length. We can then use as language L' in the intersection the regular language formed by all the strings of even length in Σ^*, where Σ is the alphabet of the language L.

The fact that L' is a regular language can easily be verified noting that $L' = L(M)$, where M is the DFA defined as

$$M \quad = \quad (\{q_0, q_1\}, \Sigma, \delta, q_0, \{q_0\})$$

where the transition function δ is specified as

$$\begin{array}{rcll} \delta(q_0, a) & = & q_1, & a \in \Sigma\ ; \\ \delta(q_1, a) & = & q_0, & a \in \Sigma\ . \end{array}$$

Since $L \cap L' = P(L)$, we can concluded that the class of context-free languages is closed under the operator P.

Chapter 3

Recursive and Recursively Enumerable Languages

Exercise 3.1 Consider the alphabet $\Sigma = \{a, b\}$ and the language

$$L \quad = \quad \{w \mid w \in \Sigma^*,\ w = a^n bba^n,\ n \geq 0\}.$$

Specify a Turing machine that accepts L and stops for every possible input in Σ^*.

Solution The required Turing machine is based on a cycle looking for the two occurrences of symbol a placed at the two ends of the input, to be replaced with the B (blank) symbol. At the end of this cycle, the machine accepts if the tape contains only two occurrences of symbol b, one next to the other.

More precisely, M uses the following strategy.

- In state q_0, M reads a a at the beginning of the input string, erases it by replacing it with the B, and moves to the right by entering state q_1.
- In state q_1, M moves to the right leaving the tape unchanged, until it reaches the first occurrence of B. At this point M goes back to the left by one tape cell, therefore positioning itself at the rightmost cell with a symbol different from B, entering state q_2.
- In state q_2, M pretends to read a. If there is no such symbol at the rightmost position of the working tape, M stops in a non-accepting state. If M reads instead an a, it deletes it replacing with B, and moves to the left of a position, entering state q_3.

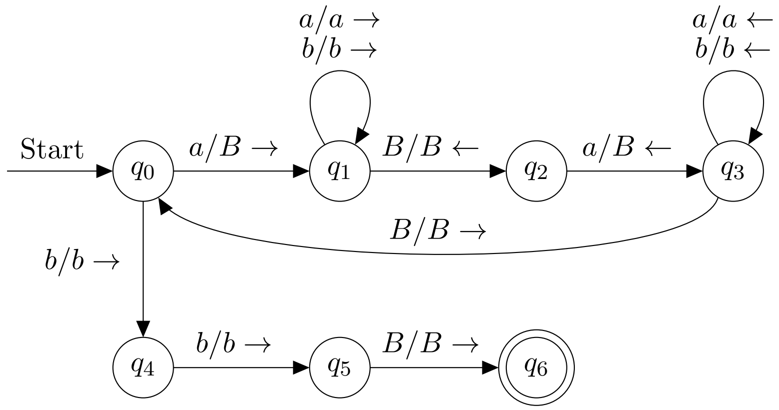

Figure 3.1: Graphical representation of the transition function of the TM from Exercise 3.1.

- In state q_3, M moves to the left leaving the tape unchanged, until it reaches the first B. At this point, M moves to the right of the current cell, therefore positioning itself at the beginning of working tape, and enters state q_0 restarting with the above cycle.
- If M is in state q_0 reading b, it moves to the right by one cell, leaving the tape unchanged and entering state q_4 and, if it can read a second occurrence of b, it still moves to the right by one cell, leaving the tape unchanged and entering the state q_5. Finally, if in state q_5 M reads B, then it enters the accepting state q_6 and stops.

Formally, the required TM is defined as $M = (Q, \Sigma, \Gamma, \delta, q_0, B, \{q_6\})$, where $Q = \{q_0, q_1, \ldots, q_6\}$, $\Gamma = \Sigma \cup \{B\}$, and the transition function δ is represented graphically in Figure 3.1.

Exercise 3.2 Let $\Sigma = \{0, 1\}$ and consider the language over Σ defined as

$$L \;=\; \{w \mid w \in \Sigma^*,\ \#_0(w) = \#_1(w)\}.$$

Specify a Turing machine that accepts L and halts for every possible input in Σ^*.

Solution The required TM M uses two different computation phases. In the first phase, M scans the tape from left to right, looking for the first occurrence of a 0, and then replaces this occurrence with the special symbol X. At this point M repositions its head at the beginning of the tape. The second phase is similar to the first: M scans the tape again from left to right, looking for the first occurrence of a 1, and replaces it with the special symbol Y. Then M repositions its head at the beginning of the tape. Altogether, a computation of M if formed by a cycle in which phases one and two above alternate. The cycle ends when there are no more occurrences of 0 or 1 in the tape. In such condition, M halts in a final state, therefore accepting the input string. In all other cases, M stops without accepting.

More precisely, M uses the following strategy.

- In state q_0, M starts from the beginning of the tape and proceeds to the right. When reading an occurrence of 0, M replaces it with the character X, and moves to the left by one tape cell, entering state q_1.

- In state q_1, M moves to the left leaving the tape unchanged. Note that the encountered symbols can only be $1, X, Y$, since the occurrences of 0 have all been replaced. When M reaches the first B (blank), it moves to the right by one tape cell, and positions itself at the beginning of the tape, entering state q_2.

- In state q_2, M starts from the beginning of the tape and proceeds to the right. When reading an occurrence of 1, M replaces it with the character Y, and moves to the left by one tape cell, entering state q_3.

- In state q_3, M moves to the left leaving the tape unchanged. Note that the encountered symbols can only be $0, X, Y$, since the occurrences of 1 have all been replaced. When M reaches the first B, it moves to the right by one tape cell, and positions itself at the beginning of the tape, entering state q_0 and restarting the previous cycle.

- If M is in state q_0, moving to the right in search of the leftmost occurrence of 0 in the tape, and reads a B character, then M has reached the end of the tape and must conclude that there are no more occurrences of 0 on the tape. In this case M leaves B unchanged, moves to the left by one tape cell, and enters state q_4.

- In state q_4, M runs the tape to the left to verify that there are no unprocessed occurrences of 1. If this is the case, M enters the final state q_5, indicating acceptance of the input string, and halts. In all

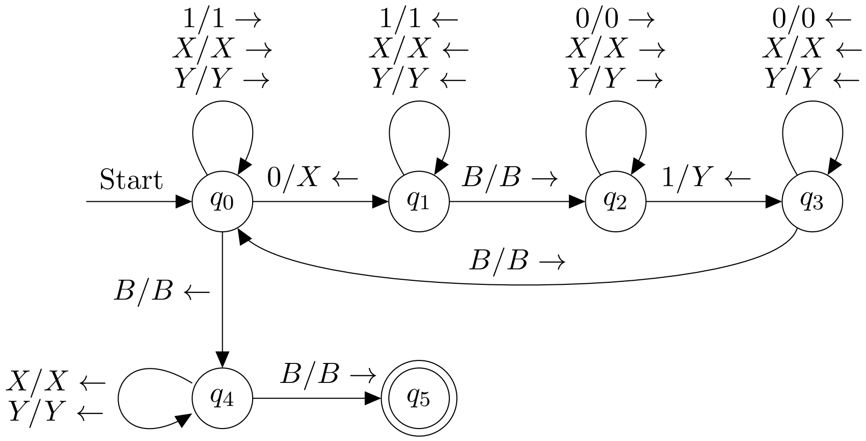

Figure 3.2: Graphical representation of the transition function of the TM from Exercise 3.2.

remaining cases, i.e. in cases not covered in the description reported above, M halts without accepting.

Formally, we define the TM M as follows

$$\begin{aligned} M &= (Q, \Sigma, \Gamma, \delta, q_0, B, \{q_5\}), \\ Q &= \{q_0, q_1, \ldots, q_5\}, \\ \Gamma &= \Sigma \cup \{X, Y, B\}. \end{aligned}$$

The transition function δ is graphically represented in Figure 3.2.

Exercise 3.3 Consider the alphabet $\Sigma = \{a, b\}$ and the symbol $c \notin \Sigma$. Specify a TM with output that, given any string $w \in \Sigma^*$, produces as output the string wcw and halts.

Solution The required TM M must write on its tape the string wcw. Initially M writes a c at the end of the input string, and then it positions its tape head at the beginning of the input string. At this point M enters a cycle in which it performs the following actions. M reads the first character

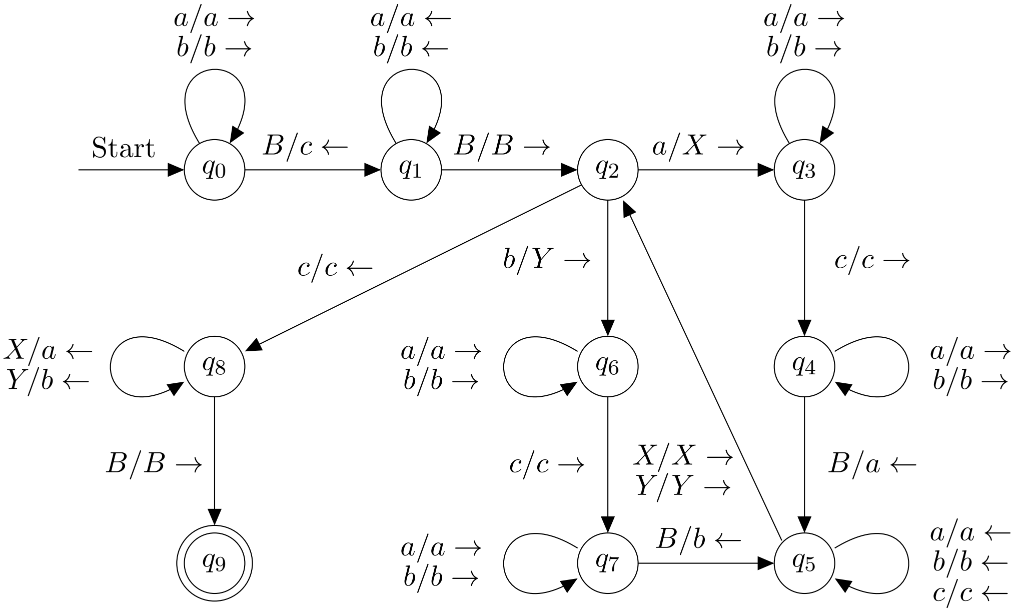

Figure 3.3: Graphical representation of the transition function of the TM from Exercise 3.3.

of w not yet copied, and replaces it with X or Y depending on whether this character is a or b, respectively. Recording the character in its internal state, M copies it at the end of a new string to the right of c. When all the characters of the string w to the left of c have been replaced by X or Y, M will have entirely copied w to the right of c. At this point, M replaces every X with a and every Y with b, restoring the initial string w to the left of c.

More precisely, M uses the following strategy.

- In state q_0, M starts from the beginning of the tape and proceeds to the right. When M reaches the first character B (blank), M rewrites it with symbol c, and moves to the left by a tape cell, entering state q_1.
- In state q_1, M proceeds to the left. When the first character B is reached, M leaves it unchanged and moves to the right by a tape cell,

entering state q_2. At this point M is placed at the beginning of the input string w.

- In state q_2, M starts a cycle. If the read symbol is a, M replaces it with X, moves all the way to the right, and copies a in place of the first B it encounters. To do this, M passes through states q_3, q_4 and q_5. Symmetrically, if the read symbol is b, M replaces it with Y and copies the b in place of the first B it encounters when moving to the right, passing through states q_6, q_7 and q_5.

- In state q_5, M proceeds to the left until it reaches an occurrence of either X or Y. At this point M moves to the right by one tape cell, entering state q_2. In this last state, if M reads a or b, it means that there are symbols of the input string that have not yet been copied, so M restarts its main cycle. If M reads the c character instead, then all symbols of the input string have been copied. In the latter case M moves to the left of a cell, and enters state q_8.

- In state q_8, M proceeds to the left, replacing each X with a and each Y with b. When M reaches the first symbol B, M leaves it unchanged and moves to the right by a tape cell, enters the final state q_8 and accepts.

Formally, we define the TM M as follows

$$\begin{aligned} M &= (Q, \Sigma, \Gamma, \delta, q_0, B, \{q_9\}), \\ Q &= \{q_0, q_1, \dots, q_9\}, \\ \Gamma &= \Sigma \cup \{c, X, Y, B\}. \end{aligned}$$

The transition function δ of M is graphically represented in Figure 3.3.

Exercise 3.4 Let L_1 be a recursive language which is not CFL. Consider the following property of the recursively enumerable (RE) languages:

$$\mathcal{P} = \{L \mid L \in \text{RE},\ L \subseteq L_1\}.$$

Let $L_{\mathcal{P}} = \{\mathsf{enc}(M) \mid L(M) \in \mathcal{P}\}$, where $\mathsf{enc}(M)$ represents an encoding of the Turing machine M. Assess whether $L_{\mathcal{P}}$ is a recursive language.

Solution By applying Rice's theorem we can very often solve exercises as this one, asking to assess whether $L_{\mathcal{P}}$ is a recursive language. Rice's theorem states that if the property $\mathcal{P}$ is non-trivial, that is if $\mathcal{P} \neq \text{RE}$ and $\mathcal{P} \neq \emptyset$, then $\mathcal{P}$ is not decidable. Recall that a property $\mathcal{P}$ of the RE languages is decidable if the language $L_{\mathcal{P}}$ is recursive, where $L_{\mathcal{P}}$ consists of strings that are codings of Turing machine that accept languages in $\mathcal{P}$.

We certainly have $\mathcal{P} \neq \emptyset$: in fact L_1 is recursive, and therefore it is also in RE, and it belongs to $\mathcal{P}$ since $L_1 \subseteq L_1$. We also have $\mathcal{P} \neq \text{RE}$, although this verification is slightly more complex. We observe that $L_1 \neq \Sigma^*$, because otherwise L_1 would be regular, and therefore also context-free, violating the hypothesis. Then there exists at least one string $w \in \Sigma^*$ that does not belong to L_1. We can then choose $L_2 = \{w\}$ as a RE language that is not a subset of L_1 and therefore does not belong to $\mathcal{P}$.

Applying now Rice's theorem, we conclude that $\mathcal{P}$ is not decidable, so $L_{\mathcal{P}}$ is not recursive.

Exercise 3.5 Let w be a string in Σ^*. Consider the following property of the recursively enumerable (RE) languages:

$$\mathcal{P} = \{L \mid L \in RE,\ w \in L\}.$$

Let also $L_{\mathcal{P}} = \{\mathsf{enc}(M) \mid L(M) \in \mathcal{P}\}$. Assess whether

(i) $L_{\mathcal{P}}$ is recursive;

(ii) $L_{\mathcal{P}}$ is recursively enumerable.

Solution As a first question, the exercise asks whether $L_{\mathcal{P}}$ is recursive. For this point we can apply Rice's theorem, and check whether $\mathcal{P}$ is a trivial property.

First, we have that $\mathcal{P} \neq \emptyset$. To see this, we observe that $L_1 = \{w\}$ is certainly a recursively enumerable language. This follows from the fact that L_1 is a finite language and thus a regular language, and the class of regular languages is included in the class RE. Furthermore, L_1 contains w and therefore $L_1 \in \mathcal{P}$. Second, we have $\mathcal{P} \neq \text{RE}$. To see this, we observe that the language $L_2 = \{z\}$ with z a string different from w, is certainly a recursively enumerable language (from the same argument as above) but does not belong to $\mathcal{P}$ since it does not contain the string w.

We have thus shown that the property $\mathcal{P}$ is not trivial. At this point we can apply Rice's theorem and conclude that $L_{\mathcal{P}}$ is not recursive.

We now move on with the second question of the exercise. It is quite easy to specify a TM $M_{\mathcal{P}}$ accepting $L_{\mathcal{P}}$.

- $M_{\mathcal{P}}$ receives as input a string z and checks if it is a valid encoding $\mathsf{enc}(M)$ of some TM M. If not, then $M_{\mathcal{P}}$ halts in a non-final state.

- $M_{\mathcal{P}}$ simulates M on input w. If M accepts, then $M_{\mathcal{P}}$ also accepts.

Note that, at the second item above, M could not halt on input w. In this case also $M_{\mathcal{P}}$ would not halt. Since $L_{\mathcal{P}}$ is not recursive, this scenario can not be avoided.

The following chain of logical equivalence relations formally proves that $L(M_{\mathcal{P}}) = L_{\mathcal{P}}$:

$$\begin{array}{rlll}
\mathsf{enc}(M) \in L(M_{\mathcal{P}}) & \text{iff} & w \in L(M) & (\text{definition of } M_{\mathcal{P}}) \\
& \text{iff} & L(M) \in \mathcal{P} & (\text{definition of } \mathcal{P}) \\
& \text{iff} & \mathsf{enc}(M) \in L_{\mathcal{P}} & (\text{definition of } L_{\mathcal{P}}).
\end{array}$$

Exercise 3.6 Let Σ be some alphabet and let $a \in \Sigma$. Consider the following property of recursively enumerable (RE) languages:

$$\mathcal{P} \;=\; \{L \mid L \in \text{RE}, \; \#_a(w) \geq 4 \text{ for each } w \in L\}.$$

Let $L_{\mathcal{P}} = \{\mathsf{enc}(M) \mid L(M) \in \mathcal{P}\}$. State whether $L_{\mathcal{P}}$ is a recursive language.

Solution To assess whether $L_{\mathcal{P}}$ is recursive, we check the hypotheses of Rice's theorem.

First, we show that $\mathcal{P} \neq \emptyset$. To see this, consider the language $L_1 = \{aaaaa\}$. L_1 is a finite language, and therefore also a recursively enumerable language. Furthermore, its only string contains more than four occurrences of a and therefore L_1 belongs to $\mathcal{P}$.

Second, $\mathcal{P} \neq$ RE. To see this, we need to construct a recursively enumerable language in which for at least one string w we have $\#_a(w) \leq 4$. A simple choice is for instance $L_2 = \{aa\}$.

Let us note that, in the construction of L_1 and L_2, we have only used the symbol a: this is the only symbol that belongs to Σ for sure; as far as we know, the alphabet might be composed by only one symbol!

Having verified that $\mathcal{P}$ is non-trivial, we now apply Rice's theorem and conclude that $L_{\mathcal{P}}$ is not recursive.

Exercise 3.7 Let L be a generic language defined over an alphabet Σ. Let us define the prefix operator

$$\mathsf{pref}(L) \quad = \quad \{w \mid wx \in L,\ w \in \Sigma^*,\ x \in \Sigma^+\}.$$

In words, $\mathsf{pref}(L)$ contains all *proper* prefixes of strings from L, that is, all strings that when prolonged by means of one or more occurrences of certain symbols from Σ can form a string in L.

Show that the class RE of all recursively enumerable languages is closed under the prefix operator.

Solution To solve the exercise we show that, for any language L in RE, we can construct a nondeterministic Turing machine N such that $L(N) = \mathsf{pref}(L)$. We know that we can always convert N into a deterministic Turing machine, and therefore we can conclude that $\mathsf{pref}(L)$ is a language in RE.

Let M be a deterministic Turing machine such that $L(M) = L$; M exists since we have assumed that L is an RE language. To accept all and only the strings in $\mathsf{pref}(L)$, the nondeterministic Turing machine N uses the following strategy.

- Using nondeterminism, N "guesses" a string $x \in \Sigma^+$.
- Next, N concatenates the input string w with x, and simulates M on wx. If M accepts, then N accepts as well; if instead M stops without accepting or else if it does not stop, then N does the same.

It is not difficult to see that $L(N) = \mathsf{pref}(L)$. In fact, if a string wx is accepted by N, then wx is also accepted by M and therefore $wx \in L$. Since we have chosen x other than the empty string ε, we can conclude that w is a proper prefix of some string in L, and therefore $w \in \mathsf{pref}(L)$. In the opposite direction, if $w \in \mathsf{pref}(L)$ by definition there exists some non empty string x such that $wx \in L$. Then, among the many nondeterministic calculations of N, there must be a computation that generates x and simulates M on wx. Since M accepts wx, N accepts w.

Alternative solution We can solve the exercise by directly building a deterministic Turing machine M_d such that $L(M_d) = \mathsf{pref}(L)$. This alternative proof gives us the opportunity of introducing a technique which is fairly widespread in standard exercises about computability theory.

As in our previous solution, le M be a deterministic Turing machine such that $L(M) = L$. We might then think about building M_d in such a way that it generates all strings $x \in \Sigma^+$, one at a time, and then dispatches string

wx to M as an input. However, this idea hides a severe mistake. Consider x_i, the i-th string in Σ^+ assuming the lexicographic order. Computation of M on wx_i might never stop, according to the definition of languages in RE. In this case, any strings $wx_j \in L$ with $j > i$ would never be examined, and therefore the machine M_d would not have the correct behavior.

The problem can be solved by inserting into M_d a *pair generator* G. The latter produces as output all possible pairs of natural numbers (i, j) with $i, j > 0$. G can easily be realized by increasing some counter k, starting from 2, and printing for each value of k all pairs of positive integers satisfying $i + j = k$, sorted by growing values of i. The integer pairs are therefore generated by G in the following order

$$(1,1), (1,2), (2,1), (1,3), (2,2), (3,1), \ldots$$

Every time G generates a pair (i, j), M_d performs a simulation of the TM M on wx_i for exactly j computation steps, after which we move on with the next pair generated by G. In this way, we can avoid infinite computations. So let us assume that there exists some string in L having the form wx_i. Then M accepts wx_i in some number j of computation steps. Furthermore, pair (i, j) must be generated by G after some finite number of steps. At that point, M_d successfully completes the simulation of M on wx_i, and therefore accepts w. On the other hand, if there is no string $x_i \in \Sigma^+$ such that $wx_i \in L$, then none of the simulations of M for a finite number of computation steps will ever be successful, and M_d will continue to cycle endlessly without ever accepting the input string w.

Exercise 3.8 Let L_1 be a finite language. Consider the following property of the recursively enumerable (RE) languages:

$$\mathcal{P} \quad = \quad \{L \mid L \in \text{RE},\ L_1 \cap L \neq \emptyset\}.$$

Let $L_{\mathcal{P}} = \{\mathsf{enc}(M) \mid L(M) \in \mathcal{P}\}$. Assess whether

(i) $L_{\mathcal{P}}$ is recursive;

(ii) $L_{\mathcal{P}}$ is recursively enumerable.

Solution We need to distinguish between two different cases, depending on whether L_1 is the empty language or not.

$L_1 = \emptyset$. Then the property $\mathcal{P}$ is trivial. To see this, observe that any language intersected with L_1 provides the empty language. Then no recursively enumerable language can satisfy the condition $L_1 \cap L \neq \emptyset$ in the definition of property $\mathcal{P}$, and therefore $\mathcal{P} = \emptyset$. In this case $L_{\mathcal{P}} = \emptyset$, and it is certainly a recursive language, since it is recognized by a TM that, regardless of the given input, always halts without accepting. Of course, $L_{\mathcal{P}}$ is also a recursively enumerable language.

$L_1 \neq \emptyset$. We start again from the verification of the hypotheses of Rice's theorem. First, we observe that $\mathcal{P} \neq \emptyset$. To see this observe that L_1 is a finite language, and therefore also a regular language. Since the class of regular languages is included in the class of recursively enumerable languages, we have that L_1 is a recursively enumerable language. Furthermore, we have $L_1 \in \mathcal{P}$, since $L_1 \cap L \neq \emptyset$ for $L = L_1$.

Second, $\mathcal{P}$ does not include all recursively enumerable languages. To see this, we use the fact that L_1 is a finite language. Therefore there exists a string $w \in \Sigma^*$ such that $w \notin L_1$. Then we can define $L_2 = \{w\}$, which is a recursively enumerable language. Furthermore, we have $L_2 \notin \mathcal{P}$, since $L_1 \cap L = \emptyset$ for $L = L_2$. Applying now Rice's theorem, we have that $L_{\mathcal{P}}$ is not recursive.

Under the assumption that $L_1 \neq \emptyset$, the language $L_{\mathcal{P}}$ is a recursively enumerable language. To see this, assume that the strings in the finite language L_1 are sorted according to the lexicographical ordering. For each j with $1 \leq j \leq |L_1|$, let w_j be the j-th string of L_1. Consider then the TM $M_{\mathcal{P}}$ specified by means of the following steps.

- $M_{\mathcal{P}}$ receives as input a string z and checks whether z is a valid encoding $\mathsf{enc}(M)$ of a TM M. If not, $M_{\mathcal{P}}$ halts in a non-final state.
- $M_{\mathcal{P}}$ sets $i = 1$.
- For each j, $1 \leq j \leq |L_1|$, $M_{\mathcal{P}}$ simulates M on w_j for exactly i steps. If M accepts at least one string w_j in i steps, then $M_{\mathcal{P}}$ halts and accepts.
- $M_{\mathcal{P}}$ increments by one unit the value of i.
- $M_{\mathcal{P}}$ iterates starting at third item above.

The simulation of M for a limited number of steps i is needed to avoid computations on some strings that may take forever. In fact, assume that M does not halt after any finite number of steps on some string $w_j \in L_1$. In this case, if $M_{\mathcal{P}}$ would not impose any time limit on its simulation of M on w_j, then $M_{\mathcal{P}}$ would never be able to examine the next string w_{j+1}. But w_{j+1} could perhaps be the string witnessing that the intersection between L_1 and $L(M)$ is not empty.

As already done for some previous exercises, we now show that $L(M_{\mathcal{P}}) = L_{\mathcal{P}}$ through a chain of equivalences:

$$\begin{array}{lll} \mathsf{enc}(M) \in L(M_{\mathcal{P}}) & \text{iff} & M \text{ accepts some string } w \in L_1 \\ & \text{iff} & L_1 \cap L(M) \neq \emptyset \\ & \text{iff} & L(M) \in \mathcal{P} \\ & \text{iff} & \mathsf{enc}(M) \in L_{\mathcal{P}}. \end{array}$$

Exercise 3.9 Consider the following language

$$L \;=\; \{\mathsf{enc}(M, M') \mid L(M) \cap L(M') \neq \emptyset\}$$

where M, M' are generic TMs and $\mathsf{enc}(M, M')$ is a string representing a fixed encoding for M, M'. Prove that L is not a recursive language. (Hint: reduce from L_u to L.)

Solution We must show that there is no algorithm that can recognize all and only the strings in L, i.e., that there is no TM that recognizes L and halts for every possible input string. At first sight, one might think that to solve this exercise, Rice's theorem should be exploited. However, a closer look reveals that this is not the right way to proceed. In fact, Rice's theorem only applies to languages of the form $\{\mathsf{enc}(M) \mid L(M)$ satisfies some property $\mathcal{P}$ of RE languages$\}$. This is not the case for the exercise at hand, where each instance has instead the form $\mathsf{enc}(M, M')$, that is, each instance represents the encoding of a pair of TMs, not just one.

To solve the exercise, we instead follow the hint provided in the text and prove that the universal language L_u defined in the textbook can be reduced to L, which we denote by writing $L_u \leq_m L$. This means that deciding whether a string belongs to L is at least as difficult as deciding whether a string belongs to L_u. Since the universal language L_u is undecidable, as shown in the textbook, we conclude that also L must be undecidable.

To prove $L_u \leq_m L$, we must provide a construction (basically, a TM) that transforms any string of the type $\mathsf{enc}(M, w)$, which represents an instance of the problem associated with L_u, into a string $\mathsf{enc}(M_1, M_2)$, which represents an instance of the problem associated with our target language L. In order for our construction to represent a valid reduction, the two TMs M_1 and M_2 must satisfy the condition

$$L(M_1) \cap L(M_2) \neq \emptyset \text{ if and only if } w \in L(M).$$

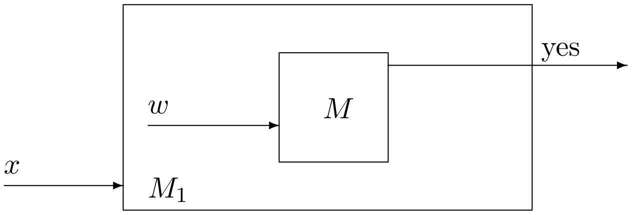

Figure 3.4: The TM M_1 in the solution of Exercise 3.9.

A possible pair M_1, M_2 satisfying the above condition is obtained by taking two identical copies of the TM reported in Figure 3.4. More precisely, M_1 and M_2 are identical and are constructed in such a way that

$$L(M_1) = L(M_2) = \begin{cases} \Sigma^*, & \text{if } w \in L; \\ \emptyset, & \text{if } w \notin L. \end{cases}$$

To see that M_1 and M_2 satisfy the required condition, we observe that if $w \in L(M)$, we have $L(M_1) \cap L(M_2) = \Sigma^*$ and therefore $L(M_1) \cap L(M_2) \neq \emptyset$. If instead $w \notin L(M)$, the language accepted by each of the two machines is $\emptyset$ and therefore $L(M_1) \cap L(M_2) = \emptyset$.

We have therefore shown that $L_u \leq_m L$. Since L_u is not a recursive language, our reduction completes the exercise.

An alternative way to prove the reduction $L_u \leq_m L$ is as follows. As above, let $\mathsf{enc}(M, w)$ be the input instance and let $\mathsf{enc}(M_1, M_2)$ be the output instance of the construction. We then let $M_1 = M$ and $M_2 = M_w$, where M_w is a special TM that recognizes the language $\{w\}$, that is, M_w accepts only the string w. We can easily see that

$$L(M) \cap L(M_w) = \begin{cases} \{w\}, & \text{if } w \in L(M) \\ \emptyset, & \text{if } w \notin L(M) \end{cases}$$

which is exactly the desired condition.

Alternative solution. An alternative solution for this exercise is to reduce from L_{ne} to L, where

$$L_{ne} \quad = \quad \{\mathsf{enc}(M) \mid L(M) \neq \emptyset\}$$

is a language introduced in the textbook, where it is also shown that L_{ne} is not recursive.

Let $\mathsf{enc}(M)$ be an instance of the decision proble associated with L_{ne}, that is, $\mathsf{enc}(M)$ is the encoding of a TM for which we have to test whether $L(M) \neq \emptyset$. The reduction produces an instance $\mathsf{enc}(M, M)$ of L, that is, a string representing the encoding of two copies of M. The following chain of logical equivalences shows that the proposed construction represents a valid reduction:

$$\begin{array}{rlll} \mathsf{enc}(M, M) \in L & \text{iff} & L(M) \cap L(M) \neq \emptyset & \text{(definition of } L) \\ & \text{iff} & L(M) \neq \emptyset & \text{(definition of } \cap) \\ & \text{iff} & \mathsf{enc}(M) \in L_{ne} & \text{(definition of } L_{ne}) \ . \end{array}$$

Since there is no TM that always halts and that recognizes L_{ne}, we can conclude that L is not a recursive language.

Exercise 3.10 Provide answers to the following questions, together with a mathematical proof for your statemets.

(i) Let L_1 and L_2 be two recursive languages. Is the language L_1L_2 a recursive language?

(ii) Let L_1 be a recursive language and let L_2 be a recursively enumerable language. Is $L_1 \cap L_2$ a recursive language?

(iii) Consider the language

$$L \quad = \quad \{\mathsf{enc}(M, M') \mid L(M) \subseteq L(M')\}$$

where M, M' are generic TMs and $\mathsf{enc}(M, M')$ is an encoding of M, M'. Is L a recursively enumerable language?

Solution Question 1: we show that the language L_1L_2 is recursive. Since L_1 and L_2 are recursive languages, there must exist two TM M_1 and M_2 that halt for every possible input, and such that $L(M_1) = L_1$ and $L(M_2) = L_2$. Using M_1 and M_2 we can build a nondeterministic TM N that recognizes the concatenation language L_1L_2. We will also show that N can be turned into a deterministic TM that halts for every possible input string.

The TM N adopts the following strategy.

- Given a string w as input, N "guesses" nondeterministically a factorization $w = xy$.

- Next, N simulates M_1 on the input string x, and simulates M_2 on the input string y.
- N halts and accepts if both M_1 and M_2 accept their respective inputs. Otherwise, N stops and does not accept the input string w.

Let us first show the equivalence relation $L_1L_2 = L(N)$, separately discussing the two parts $L_1L_2 \subseteq L(N)$ and $L(N) \subseteq L_1L_2$. By definition of the concatenation operation, we have that if $w \in L_1L_2$ then there exists at least one pair of strings x, y such that $xy = w$, $x \in L_1$ and $y \in L_2$. N will certainly guess the factorization $xy = w$. Furthermore, the simulation of both machines M_1 and M_2 on x and y, respectively, end up in a final state, and thus also N accepts. We have then $w \in L(N)$, and therefore $L_1L_2 \subseteq L(N)$.

Let us now assume $w \in L(N)$. There exists a computation of N that guesses a factorization $w = xy$, such that M_1 and M_2 halt on x and y, respectively, both in a final state. This implies that $x \in L_1$ and $y \in L_2$, and by definition of concatenation we have $w \in L_1L_2$. Therefore $L(N) \subseteq L_1L_2$.

Finally, we observe that all of the computations of N come to a halt. Indeed, both TM M_1 and M_2 halt for every possible input, since the languages associated with the two TMs are recursive. In addition, the possible ways to split a string w into two parts are $|w|+1$. In this case therefore we can transform N into a deterministic TM M with $L(N) = L(M)$, using the technique studied in the textbook, and we are guaranteed that even M halts for every possible input. We conclude therefore that the language L_1L_2 is recursive.

Question 2: we show that the language $L_1 \cap L_2$ may not be recursive. To prove this, it is sufficient to show a counterexample. Consider the language $L_1 = \Sigma^*$, which is a regular language and therefore also a recursive language. Consider also the language $L_2 = L_u$, which is a well-known recursively enumerable language but not recursive. We have $L_1 \cap L_2 = L_u$ which, as already stated, is not a recursive language.

Question 3: we show that the language L is not a recursively enumerable language (RE). To prove this, we provide a reduction from a language not in RE to L. As a language not in RE we choose the following language, which has been presented in the textbook

$$L_e \quad = \quad \{\mathsf{enc}(M'') \mid L(M'') = \emptyset\}.$$

The reduction takes as input an instance of L_e, $\mathsf{enc}(M'')$, and builds an instance of L, $\mathsf{enc}(M, M')$, such that $\mathsf{enc}(M'') \in L_e$ if and only if $\mathsf{enc}(M, M') \in L$. The construction of $\mathsf{enc}(M, M')$ is defined below.

- We set $M = M''$
- We set $M' = M_e$, where M_e is any TM that accepts the empty language.

We now prove that the proposed construction is a valid reduction, that is, $\mathsf{enc}(M') \in L_e$ if and only if $\mathsf{enc}(M, M') \in L$.

(Only if part) If $\mathsf{enc}(M'') \in L_e$, then $L(M'') = \emptyset$. Consequently, we have $L(M) = L(M') = \emptyset$, and therefore $L(M) \subseteq L(M')$. This allows us to conclude that $\mathsf{enc}(M, M') \in L$.

(If part) If $\mathsf{enc}(M'') \notin L_e$, then $L(M'') \neq \emptyset$. Therefore we have $L(M) \not\subseteq L(M')$, and then $\mathsf{enc}(M, M') \notin L$.

The reduction allows us to conclude that the recognition of the language L is at least as complex as the recognition of the language L_e. Since we know that L_e does not belong to RE, we must conclude that L is not in RE as well.

Exercise 3.11 Consider the language

$$L = \{\mathsf{enc}(M, w) \mid w \notin L(M)\}$$

where $\mathsf{enc}(M, w)$ is a binary string encoding the TM M and the binary string w. Prove that L is not a recursively enumerable language.

Solution We observe upfront that the language defined in the exercise is the complement of the universal language L_u defined in the textbook, that is, $L = \overline{L_u}$. Since we have seen that L_u is recursively enumerable but not recursive, from a well-known theorem characterizing the complement of recursively enumerable but not recursive languages, we have that $\overline{L_u}$ is not recursively enumerable.

We can also directly prove that $\overline{L_u}$ is not recursively enumerable by providing a reduction from L_d, the diagonal language defined in the textbook as:

$$L_d = \{w_i \mid w_i \notin M_i\}$$

where w_i is the i-th string in the language $\{0, 1\}^*$, according to the indexing studied in the textbook, and M_i is the Turing machine whose encoding $\mathsf{enc}(M_i)$ is w_i.

To define our reduction, let us consider a string w which represents an instance of the decision problem associated with L_d. We specify a Turing machine that, on input w, finds the index i such that $w_i = w$, and returns as output the string $\mathsf{enc}(M_i, w_i)$ where, as already mentioned, M_i is the TM whose binary code is w_i.

To conclude our proof we must show that the construction is a valid reduction, that is, we must show that $w \in L_d$ implies $\mathsf{enc}(M_i, w_i) \in \overline{L_u}$, and $\mathsf{enc}(M_i, w_i) \in \overline{L_u}$ implies $w \in L_d$. We have the following chain of logical equivalences

$$\begin{array}{lll} w \in L_d & \text{iff} \quad w = w_i \notin L(M_i) & \text{(definition of } L_d) \\ & \text{iff} \quad \mathsf{enc}(M_i, w_i) \notin L_u & \text{(definition of } L_u) \\ & \text{iff} \quad \mathsf{enc}(M_i, w_i) \in \overline{L_u} & \text{(definition of complement operator).} \end{array}$$

Since there is no TM that can recognize L_d, we conclude that there is no Turing machine for $\overline{L_u}$ either, and then $\overline{L_u}$ is not a recursively enumerable language.

Exercise 3.12 Consider the language

$$L \;=\; \{\mathsf{enc}(M_1, M_2, M_3) \mid L(M_1) \cap L(M_2) \neq L(M_3),\ \}$$

where M_1, M_2 and M_3 are generic TMs, and where $\mathsf{enc}(M_1, M_2, M_3)$ represents an appropriate encoding of these machines. Prove that L is not a recursive language.

Solution We have already proved in Exercise 3.9 that the language $L' = \{\mathsf{enc}(M, M') \mid L(M) \cap L(M') \neq \emptyset\}$ is not a recursive language. To solve the present exercise, then, we can provide a reduction from L' to L, proving therefore that if there were a TM that always halts and recognizes L, then there would be a TM that always halts and recognizes L', which contradicts the findings of Exercise 3.9.

To construct our reduction we operate as follows. Let $\mathsf{enc}(M, M')$ be an input instance for L', that is, a string for which we must verify membership in L'. We need to provide an instance $\mathsf{enc}(M_1, M_2, M_3)$ and show that the construction is indeed a valid reduction. We set $M_1 = M$, $M_2 = M'$. Furthermore for M_3 we take and TM that accepts the empty set. At this point we have

$$\begin{array}{llll} \mathsf{enc}(M, M') \in L' & \text{iff} & L(M) \cap L(M') \neq \emptyset & \text{(definition of } L') \\ & \text{iff} & L(M_1) \cap L(M_2) \neq L(M_3) & \text{(definition of } M_3) \\ & \text{iff} & \mathsf{enc}(M_1, M_2, M_3) \in L & \text{(definition of } L). \end{array}$$

This shows that our construction is a valid reduction, and concludes the solution of the exercise.

As an alternatively solution, we can reduce L_u to L as follows. Let $\mathsf{enc}(M, w)$ be an instance of the decision problem associated with L_u. Let also $M_\emptyset$ be a TM that recognizes the empty language, and let M_w be a TM that recognizes the language $\{w\}$, i.e. a language consisting only of the string w. We can map the instance $\mathsf{enc}(M, w)$ into the instance $\mathsf{enc}(M, M_w, M_\emptyset)$. It is immediately evident that

$$L(M) \cap L(M_w) = \begin{cases} \{w\}, & \text{if } w \in L(M) \\ \emptyset, & \text{if } w \notin L(M) \end{cases}$$

Since $\{w\} \neq \emptyset$ is always true, we can conclude that $\mathsf{enc}(M, w) \in L_u$ if and only if $\mathsf{enc}(M, M_w, M_\emptyset) \in L$, which proves that the presented construction is indeed a valid reduction. Finally, since we know that L_u is not a recursive language, we obtain that even L is not a recursive language.