Chapter 4

Turing Machines and Computable Functions

We have seen that PDAs, containing a finite control and equipped with a stack-like memory, can accept a wide range of languages. The class of context-free languages can be characterized as the class of languages accepted by such a PDA. However, we have also seen some languages that are not context-free. The language $\{ a^n b^n c^n \mid n \geqslant 0 \}$ of Example **??** and the language $\{ ww \mid w \in \{a, b\}^* \}$ of Example **??** cannot be accepted by a PDA.

In this chapter, we consider reactive Turing machines like the machine depicted in Figure 4.1. Reactive Turing machines accept a class of languages called the recursively enumerable languages. This class is wider than the class of context-free languages accepted by PDAs. In particular, there are Turing machines that accept the languages of the examples mentioned above.

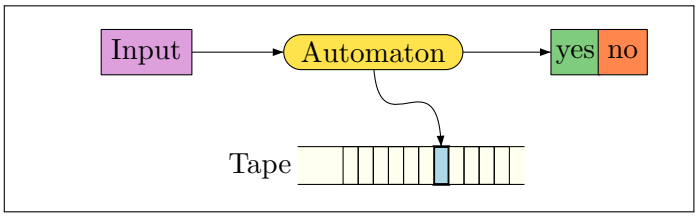


Figure 4.1: Architecture of the Reactive Turing machine.

The push-down automata of the previous chapter have a memory containing a stack of symbols which can only be accessed at the top. The Turing machine has a tape as memory, which also symbols as contents, but can be accessed at any place. It is said that a Turing machine has random access. This entails that the so-called tape head can move around the string of symbols on the tape. The Turing machine seems only like a small advancement over the possibilities of a push-down automaton. Nevertheless, Turing machines are much more powerful than PDAs. Even, it can be argued that any computation that can be done by any computer can also be done by a Turing machine.

We will generalize the case of output of only 'yes' or 'no' for the reactive Turing machine to the case where the output is an arbitrary string left at the tape at the end of a computation. This yields the notion of a classical Turing machine. If a function can be implemented on a classical Turing machine, we speak of a *computable* function.

4.1 The reactive Turing machine

We will first introduce some notation that is useful to describe Turing machines. The memory tape, or just tape for short, of a Turing machine consists of an infinite supply of sequentially ordered non-numbered cells, and each cell will contain a data element $d \in \Delta$ or is empty. We use the special symbol #, the *blank*, to denote an empty cell ($\# \notin \Delta$). The extended tape alphabet $\Delta_{\#}$, that we use besides the input alphabet Σ, consists of all symbols in Δ together with the blank, i.e. $\Delta_{\#} = \Delta \cup \{\#\}$. So, one can also say that a tape cell always contains a symbol, a non-blank $d \in \Delta$ or the blank symbol #.

At each point in an execution, the tape head of the Turing machine is positioned at exactly one particular cell, the cell in the eye of the tape head. From this cell the Turing machine can read a data element or a blank and can write a data element or erase the content (i.e. replace it with a blank). Then the tape head will move one cell to the right or one cell to the left.

We use the expression $x\langle e\rangle y$, with $x, y \in \Delta_{\#}^*$, $e \in \Delta_{\#}$, to denote a tape containing the string xey: the tape head is at a cell holding e, the cells left of the tape head together contain the string x, the cells right of the tape head together contain the string y. To define a unique expression $x\langle e\rangle y$ we require for $x \neq \varepsilon$ that the leftmost symbol of x is not a blank. Likewise, if $y \neq \varepsilon$ then its rightmost symbol is assumed not to be a blank. Thus, on the left, leading blanks are ignored, and on the right, trailing blanks are ignored. We write $\langle e\rangle$ as shorthand for $\varepsilon\langle e\rangle\varepsilon$, and $\langle e\rangle y$ and $x\langle e\rangle$ for $\varepsilon\langle e\rangle y$ and $x\langle e\rangle\varepsilon$, respectively. Note, $\langle\#\rangle$ denotes the empty tape.

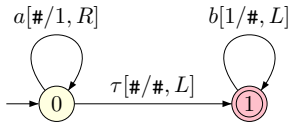


Figure 4.2: A simple reactive Turing machine.

Example 4.1. Consider Figure 4.2, depicting a reactive Turing machine with input alphabet $\Sigma = \{a, b\}$. The tape alphabet $\Delta = \{1\}$, hence $\Delta_{\#} = \{1, \#\}$. We start out in state q_0 with $\langle\#\rangle$ in memory. Thus control is in the initial state and the tape head is reading a blank, also all other cells are empty. We can take the loop of q_0 if the symbol a is on input. Then, a 1 is written in the cell that is currently pointed at and the tape head moves to the right. Thus we obtain $1\langle\#\rangle$ as tape content. Alternatively, we can take silently the edge labeled τ to the final state q_1, keeping $\langle\#\rangle$ as it appears. (In fact, the tape head has moved to the left, but our notation doesn't show.) Taking the loop three times on input aaa gives $111\langle\#\rangle$ with control in state q_0, then taking the edge

to q_1, consuming no input, gives $11\langle 1\rangle$, at which point, in q_1, we can either terminate, or execute the loop on input b to obtain $1\langle 1\rangle$. Maximally three b's can be read at the expense of erasing a 1 for each b.

Definition 4.2 (Reactive Turing machine)**.** A *reactive Turing machine*, or Turing machine for short, is a septuple $M = (Q, \Sigma, \Delta, \#, \rightarrow, q_0, F)$ where Q is a finite set of states, Σ is a finite input alphabet with $\tau \notin \Sigma$, Δ is a finite data or tape alphabet, $\# \notin \Delta$ a special symbol called *blank*, $\rightarrow \subseteq Q \times \Sigma_\tau \times \Delta_\# \times \Delta_\# \times \{L, R\} \times Q$, where $\Sigma_\tau = \Sigma \cup \{\tau\}$ and $\Delta_\# = \Delta \cup \{\#\}$, is a finite set of *transitions* or *steps*, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states.

We use the symbol $\mu \in \{L, R\}$ to denote a move of the tape head, either left or right. For $a \in \Sigma$, if $(q, a, e, e', \mu, q') \in \rightarrow$, we write $q \xrightarrow{a[e/e',\mu]}_M q'$, and this means that the Turing machine M, when it is in state q, reading the symbol a on input, and reading the symbol e on tape, it can consume the input, change the symbol on the tape to e', move one cell left if $\mu = L$ or one cell right if $\mu = R$ and thereby change control to state q'. It is also possible that e and/or e' is $\#$: if e is $\#$, we are looking at an empty cell on the tape; if $e \in \Delta$ is a non-blank and e' is $\#$, then we say that the symbol e is erased.

Similarly, if $(q, \tau, e, e', \mu, q') \in \rightarrow$, we write $q \xrightarrow{\tau[e/e',\mu]}_M q'$. Now it means that the Turing machine M, when it is in state q and reading the symbol e on tape, can (without a change on input) write the symbol e' under the tape head, move left or right if $\mu = L$ or $\mu = R$, respectively, and change control to state q'. As a combined notation we may write $q \xrightarrow{\alpha[e/e',\mu]}_M q'$ with α ranging over Σ_τ.

To record the steps taking by a reactive Turing machine, we need to have a means to update the tape. Suppose the tape contains $x\langle e\rangle y$, and we can execute the transition $q \xrightarrow{\alpha[e/e',\mu]}_M q'$. Note, the data symbol e is both occurring in $x\langle e\rangle y$ as well is in the label $\alpha[e/e', \mu]$ of the transition. In this setting, we define the update $x\langle e\rangle y\,[e/e', \mu]$ of tape content $x\langle e\rangle y$ by tape head action $[e/e', \mu]$ as follows.

- $x\langle e\rangle\varepsilon\,[e/e', R] = xe'\langle\#\rangle\varepsilon$ for $x \in \Delta_\#^*$,
- $x\langle e\rangle dy[e/e', R] = xe'\langle d\rangle y$ for $d \in \Delta_\#$, $x, y \in \Delta_\#^*$,
- $\varepsilon\langle e\rangle y[e/e', L] = \varepsilon\langle\#\rangle e'y$ for $y \in \Delta_\#^*$,
- $xd\langle e\rangle y[e/e', L] = x\langle d\rangle e'y$ for $d \in \Delta_\#$, $x, y \in \Delta_\#^*$.

Here, in case the left string x or the right string y is empty, we use the full notation $\varepsilon\langle e\rangle y$ and $x\langle e\rangle\varepsilon$, respectively. When using the shorthand, the empty string cases read $x\langle e\rangle\,[e/e', R] = xe'\langle\#\rangle$ and $\langle e\rangle y\,[e/e', L] = \langle\#\rangle e'y$.

We see that the content of the cell that is currently scanned is filled with e'. If the tape head moves right, the first symbol d of dy comes under the tape head or, in case of ε, a blank comes under the tape head. If the tape head moves left, two similar cases apply.

Expressions of the form $x\langle e\rangle y \in \Delta_{\#}^* \times \Delta_{\#} \times \Delta_{\#}^*$ occur frequently in the remainder of the chapter. We introduce the class of tape content $\mathcal{Z}$ by putting

$$\mathcal{Z} = \{\, x\langle e\rangle y \mid x \in \Delta_{\#}^* \colon x = \varepsilon \vee \mathit{first}(x) \neq \#,\ e \in \Delta_{\#},\ y \in \Delta_{\#}^* \colon y = \varepsilon \vee \mathit{last}(y) \neq \# \,\} \tag{4.1}$$

and have z range over $\mathcal{Z}$. We want the first symbol of x and the last symbol of y to be a symbol in Δ, if possible. This way, given the position of the tape head, the tape content is uniquely defined.

With the notion of an update available, we can define a configuration or instantaneous description of a Turing machine $M = (\, Q,\, \Sigma,\, \Delta,\, \#,\, \rightarrow,\, q_0,\, F\,)$, ID for short. A configuration of the Turing machine M is a triple (q, w, z) of a state q, an input string w and tape content z. Thus $(q, w, z) \in Q \times \Sigma^* \times \mathcal{Z}$. The q and z, respectively, are the current state and current content of the tape, w represents the input that is not read so far.

We write $(q, w, z) \vdash_M (q', w', z')$ iff either (i) for some $a \in \Sigma$, $e, e' \in \Delta_{\#}$, and $\mu \in \{L, R\}$, we have

$$q \xrightarrow{a[e/e',\mu]}_M q' \ \wedge\ w = aw' \ \wedge\ z[e/e', \mu] = z'$$

or, (ii) for some $e, e' \in \Delta_{\#}$, and $\mu \in \{L, R\}$, we have

$$q \xrightarrow{\tau[e/e',\mu]}_M q' \ \wedge\ w = w' \ \wedge\ z[e/e', \mu] = z'$$

Thus, we have $(q, w, z) \vdash_M (q', w', z')$ if Turing machine M can move from configuration (q, w, z) in one step to configuration (q', w', z'). Note, this does not exclude that $(q, w, z) \vdash_M (q'', w'', z'')$ for a configuration (q'', w'', z'') different from (q', w', z'). We write $(q, w, z) \nvdash_M$ if for no q', w' and z' we have $(q, w, z) \vdash_M (q', w', z')$. In such a situation we say that the Turing machine M blocks or halts.

At the start of an execution of a reactive Turing machine, we will assume the Turing machine is in the initial state, and that the memory tape is empty. Thus, the initial configuration of a reactive Turing machine with initial state q_0 and the string w on input is $(q_0, w, \langle\#\rangle)$.

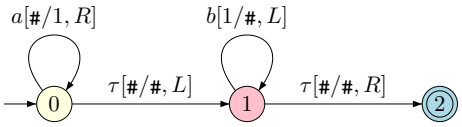


Figure 4.3: Another simple reactive Turing machine.

Example 4.3. Consider the Turing machine depicted in Figure 4.3 with input alphabet $\Sigma = \{a, b\}$, tape alphabet $\Delta = \{1\}$ and blank #. When started in q_0, any number of a's can be input, each time writing the symbol 1 on the tape and moving the tape head to the right. At any time, a move to the state q_1 can occur, reversing the direction of the

tape head. Then, the same number of b's can be input, each time erasing a 1. Coming to the beginning of the string of 1's, while having erased all of them in transit, the tape is empty again. The tape head is scanning a blank, and termination can take place after a silent transition from state q_1 to q_2.

A computation showing that M accepts the string $aaabbb$ is the following:

$$(q_0, aaabbb, \langle \# \rangle) \vdash_M (q_0, aabbb, 1\langle \# \rangle) \vdash_M (q_0, abbb, 11\langle \# \rangle) \vdash_M (q_0, bbb, 111\langle \# \rangle) \vdash_M$$
$$(q_1, bbb, 11\langle 1 \rangle) \vdash_M (q_1, bb, 1\langle 1 \rangle) \vdash_M (q_1, b, \langle 1 \rangle) \vdash_M (q_1, \varepsilon, \langle \# \rangle) \vdash_M (q_2, \varepsilon, \langle \# \rangle)$$

Note, if after a number of a's, a lower number of b's is offered on input, the machine cannot reach state q_2. Also, when a higher of number of b's is offered, the machine will get stuck in state q_2. If, after a number of a's, a lower number of b's follows, followed again by an a, the machine gets stuck in state q_1. Finally, if after a number of a's, followed by the same number of b's, yet another a follows, the machine will get stuck in q_2. Thus, for example for the string $aabb$ we have the computation

$$(q_0, aabba, \langle \# \rangle) \vdash_M (q_0, abba, 1\langle \# \rangle) \vdash_M$$
$$(q_0, bba, 11\langle \# \rangle) \vdash_M (q_1, bba, 1\langle 1 \rangle) \vdash_M (q_1, ba, \langle 1 \rangle) \vdash_M (q_1, a, \langle \# \rangle) \nvdash_M$$

The language of the Turing machine of Figure 4.3, a notion formally defined below, is the set of strings $\{ a^n b^n \mid n \geqslant 0 \}$.

Definition 4.4. Let $M = (Q, \Sigma, \Delta, \#, \rightarrow, q_0, F)$ be a reactive Turing machine. Then the language $\mathcal{L}(M) \subseteq \Sigma^*$ accepted by M is defined by

$$\mathcal{L}(M) = \{ w \in \Sigma^* \mid \exists q \in F \exists z \in \mathcal{Z} \colon (q_0, w, \langle \# \rangle) \vdash_M^* (q, \varepsilon, z) \}$$

A language $L \subseteq \Sigma^*$ is called *recursively enumerable* if $L = \mathcal{L}(M)$ for a reactive Turing machine with input alphabet Σ.

Note that it is required that the computation reaches a final state and consumes all of the input. However, nothing specific is required for the tape content z in of the end configuration (q, ε, z).

In the definition above, $\vdash_M^*$ denotes the reflexive and transitive closure of the relation $\vdash_M$. Thus

$$\begin{aligned}
&(q, w, z) \vdash_M^* (q', w', z') \quad \text{iff} \\
&\quad \exists n \geqslant 0 \, \exists q_0, \ldots, q_n \, \exists w_0, \ldots, w_n \, \exists z_0, \ldots, z_n : \\
&\qquad (q_0, w_0, z_0) = (q, w, z) \wedge \\
&\qquad\quad \forall i,\, 1 \leqslant i \leqslant n \colon (q_{i-1}, w_{i-1}, z_{i-1}) \vdash_M (q_i, w_i, z_i) \wedge \\
&\qquad\qquad (q_n, w_n, z_n) = (q', w', z')
\end{aligned}$$

For a number of languages, Turing machines can be constructed. We start out easy.

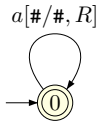


Figure 4.4: A reactive Turing machine for $\{a\}^*$.

Example 4.5. A Turing machine that accepts the language $\{a\}^* = \{\, a^n \mid n \geqslant 0 \,\}$ is shown in Figure 4.4. As long as the input consists of a's, we move to the right. At any time, termination can occur. Although the tape head may move, not data symbol is written on tape.

Example 4.6. Every finite language is accepted by a Turing machine. For example the finite language $\{a, ba, cba\}$ over $\{a, b, c\}$ is accepted by the Turing machine given in Figure 4.5.

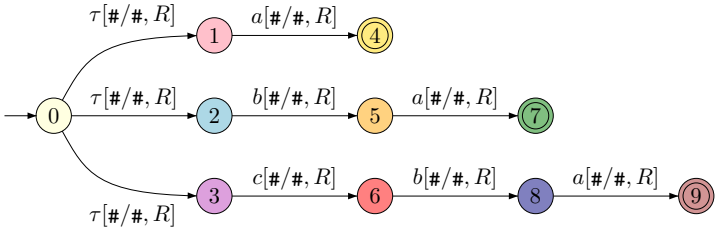


Figure 4.5: A reactive Turing machine for $\{a, ba, cba\}$.

In fact, a stronger result than Example 4.6 holds: for every regular language L, there exists a Turing machine that accepts L. Thus the class of regular languages is contained in the class of recursive enumerable languages.

Theorem 4.7. Let $L \subseteq \Sigma^*$ be a language such that $L = \mathcal{L}(D)$ for a deterministic finite automaton D, then there exists a Turing machine M such that $L = \mathcal{L}(M)$.

Proof. Suppose $D = (\, Q,\, \Sigma,\, \delta,\, q_0,\, F,\,)$ is a DFA that accepts L. Define the Turing machine $M = (\, Q,\, \Sigma,\, \emptyset,\, \#,\, \rightarrow,\, q_0,\, F)$ with the same set of states, the same input alphabet, the empty tape alphabet, the standard blank, the same initial state and the same set of final states and a transition relation $\rightarrow$ given by

$$\rightarrow \;=\; \{\, (q, a, \#, \#, R, q') \mid \delta(q, a) = q' \,\}$$

Thus, the transitions $q \xrightarrow{a[\#/\#,R]} \delta(q, a)$, for $q \in Q$, $a \in \Sigma$, are the only transitions of M. Note, since the tape alphabet of M is the empty set no other symbol than a blank can be written on tape. So, the tape will remain empty, i.e. all tape cells will contain a blank, for all computations of M.

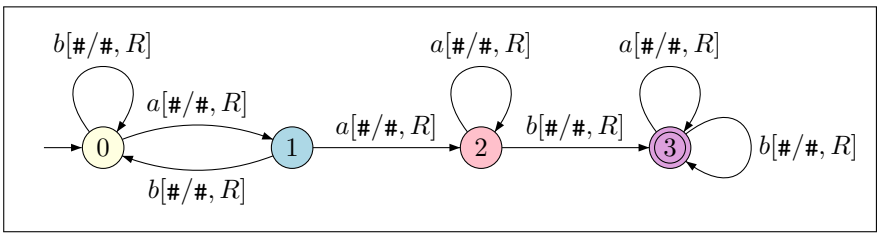


Figure 4.6: Reactive Turing machine for DFA of Example **??**

By definition of $\rightarrow$ it holds that

$$\begin{aligned}
&(q, w) \vdash_D (q', w') \\
&\quad \Leftrightarrow \quad \exists a : w = aw' \wedge \delta(q, a) = q' \\
&\quad \Leftrightarrow \quad \exists a : w = aw' \wedge q \xrightarrow{a[\#/\#,R]} q' \\
&\quad \Leftrightarrow \quad (q, w, \langle \# \rangle) \vdash_M (q', w', \langle \# \rangle)
\end{aligned}$$

Thus it also holds, as can be shown by induction, that

$$(q_0, w) \vdash_D^* (q, \varepsilon) \iff (q_0, w, \langle \# \rangle) \vdash_M^* (q, \varepsilon, \langle \# \rangle)$$

From this it follows that $\mathcal{L}(M) = \mathcal{L}(D)$, which proves the theorem. □

A reactive Turing machine obtained following the recipe of the proof of Theorem 4.7 for the DFA of Figure **??**, a DFA that accepts the language $\{w \in \{a, b\}^* \mid aab \text{ substring of } w\}$. Note, since the language $\{ a^n b^n \mid n \geqslant 0 \}$ is a recursively enumerable language as we have seen in Example 4.3, it follows from the theorem that the class of recursively enumerable languages is strictly larger than the class of regular languages.

Example 4.8. We have seen that the language $\{ a^n b^n c^n \mid n \geqslant 0 \}$ is not a context-free language. However, an extension of the simple Turing machine of Figure 4.3 accepts this language. See Figure 4.7. Together with Theorem 4.10 below, this implies that the class of languages accepted by a Turing machine, i.e. the recursively enumerable languages, is strictly larger than the class of context-free languages.

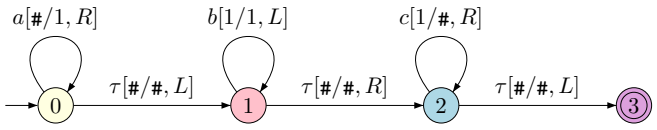


Figure 4.7: A reactive Turing machine for $\{ a^n b^n c^n \mid n \geqslant 0 \}$.

Example 4.9. With a variation on the Turing machine in Figure 4.7, also a Turing machine for the language $\{ ww \mid w \in \{a,b\}^* \}$ can be constructed. In the initial state q_0 for each a and b on input an A or B is written on tape. The Turing machine can non-deterministically go from state q_0 to state q_1. There the tape head moves left to the first blank left of the sequence of A's and B's. The Turing machine goes to state q_2. Then a sequence of a's and b's is input, and it is checked that it matches the sequence of A's and B's on the tape. If the match is exact and all input is read, the input string is accepted. If the match is not exact the Turing machine blocks.

We have $(q_0, abaaba, \langle \# \rangle) \vdash^* (q_3, \varepsilon, \langle \# \rangle)$ since

$$
\begin{array}{lll}
(q_0, abaaba, \langle \# \rangle) & & \\
\quad \vdash & (q_0, baaba, A\langle \# \rangle) & \vdash \quad (q_0, aaba, AB\langle \# \rangle) \\
\quad \vdash & (q_0, aba, ABA\langle \# \rangle) & \vdash \quad (q_1, aba, AB\langle A \rangle) \\
\quad \vdash & (q_1, aba, A\langle B \rangle A) & \vdash \quad (q_1, aba, \langle A \rangle BA) \\
\quad \vdash & (q_1, aba, \langle \# \rangle ABA) & \vdash \quad (q_2, aba, \langle A \rangle BA) \\
\quad \vdash & (q_2, ba, \langle B \rangle A) & \vdash \quad (q_2, a, \langle A \rangle) \\
\quad \vdash & (q_2, \varepsilon, \langle \# \rangle) & \vdash \quad (q_3, \varepsilon, \langle \# \rangle)
\end{array}
$$

Thus, indeed, the string $abaaba$ is accepted by the Turing machine. There are many more computations for the configuration $(q_0, abaaba, \langle \# \rangle)$, e.g.

$$
\begin{array}{l}
(q_0, abaaba, \langle \# \rangle) \vdash^* (q_0, ba, ABAA\langle \# \rangle), \\
(q_0, abaaba, \langle \# \rangle) \vdash^* (q_0, a, ABAAB\langle \# \rangle), \\
(q_0, abaaba, \langle \# \rangle) \vdash^* (q_1, a, ABAA\langle B \rangle), \\
(q_0, abaaba, \langle \# \rangle) \vdash^* (q_2, a, \langle A \rangle BAAB), \\
(q_0, abaaba, \langle \# \rangle) \vdash^* (q_2, \varepsilon, \langle B \rangle AAB),
\end{array}
$$

but these do not lead to the accepting state q_3.

The configuration $(q_0, abaabab, \langle \# \rangle)$ does have a derivation sequence to q_3. It holds that $(q_0, abaabab, \langle \# \rangle) \vdash^* (q_3, b, \langle \# \rangle)$. However, not all input has been read; the right-most symbol b is not processed. Therefore, the string $abaabab$ is *not* accepted by the Turing machine.

The crucial point is the non-determinism in state q_0. With a blank at the tape head and a symbol a or b on input, either the input symbol can be processed by the transition looping on q_0, or a silent step can be taken leading to the state q_1.

Next we prove a result for context-free languages similar to Theorem 4.7 for regular languages.

Theorem 4.10. Let $L \subseteq \Sigma^*$ be a language such that $L = \mathcal{L}(P)$ for a push-down automaton P, then there exists a Turing machine M such that $L = \mathcal{L}(M)$.

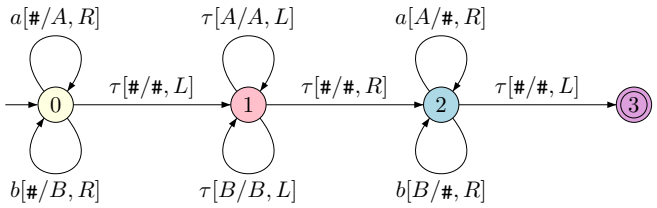


Figure 4.8: A reactive Turing machine for $\{\, ww \mid w \in \{a, b\}^* \,\}$.

Proof. Let $P = (\, Q,\, \Sigma,\, \Delta,\, \emptyset,\, \rightarrow_P,\, q_0,\, F \,)$ be a push-down automaton accepting L. Define the reactive Turing machine $M = (\, Q',\, \Sigma,\, \Delta_\varnothing,\, \#,\, \rightarrow_M,\, q_0',\, F \,)$ with an extended set of states $Q' \supseteq Q$, the same input alphabet, the same tape alphabet to which the empty stack symbol $\emptyset$ is added, a new initial state q_0', and the same set of final states. We put

$$\begin{aligned} Q' = Q \cup \{q_0', q_0''\} \cup \\ \{\, [\, move, x, q'\,] \mid \exists q \in Q \exists \alpha \in \Sigma_\tau \exists d \in \Delta_\varnothing : q \xrightarrow{\alpha[d/x]}_P q' \,\} \cup \\ \{\, [\, write, y, q'\,] \mid \exists q \in Q \exists \alpha \in \Sigma_\tau \exists d \in \Delta : q \xrightarrow{\alpha[d/x]}_P q' \wedge y \preccurlyeq x \,\} \end{aligned}$$

for two fresh states $q_0', q_0'' \notin Q$. The states of the Turing machine include the states of the PDA. However, instead of stepping from one state q of P to another state q' of P directly, a number of extra states, viz. states of the form $[\, move, x, q'\,]$ and $[\, write, y, q'\,]$, dependent on the particular transition of P involved. The intuition of states $[\, move, x, q'\,]$ is that the tape head should be moved right first, before the string x is written on tape, and state q' is entered. The intuition of states $[\, write, y, q'\,]$ is that the string y should be written on tape, from right to left, after which state q' is entered. The states q_0', q_0'' are only visited once and are there to write an empty stack symbol on tape.

For M we have the transitions

$$q_0' \xrightarrow{\tau[\#/\varnothing, L]} q_0'' \quad \text{and} \quad q_0'' \xrightarrow{\tau[\#/\#, R]} q_0$$

which together write an empty stack symbol $\varnothing$ on tape, positions the tape head on the empty stack symbol, and moves control to $q_0 \in Q'$, the state of M that is the initial state of P.

For each transition $q \xrightarrow{\alpha[\emptyset/x]}_P q'$, with $q, q' \in Q$, $\alpha \in \Sigma_\tau$, and $x \in \Delta^*$, we have for M the transitions

$$\begin{aligned} q &\xrightarrow{\alpha[\emptyset/\emptyset, L]}_M [\, write, x, q'\,] \\ [\, write, yd, q'\,] &\xrightarrow{\tau[\#/d, L]}_M [\, write, y, q'\,] \\ [\, write, \varepsilon, q'\,] &\xrightarrow{\tau[\#/\#, R]}_M q' \end{aligned}$$

for $yd \in \Delta^* \cdot \Delta$ a prefix of $x \in \Delta^*$. Thus, when the empty stack symbol $\varnothing$ is read on tape, it is written back, while the tape head moves left to an empty cell. Then, starting from

state $[\, write, x, q' \,]$, the symbols of the string x are written on tape, symbol by symbol, from right to left, while the tape head moves left too, with control moving through states $[\, write, y, q' \,]$ where string y is a prefix of string x. When all symbols of x are written and state $[\, write, \varepsilon, q' \,]$ is reached, the emulation of the transition of P is done, the tape head moves right, on the leftmost symbol of the 'stack-on-tape', and controls moves to target state q'.

For each transition $q \xrightarrow{\alpha[d/x]}_P q'$, with $q, q' \in Q$, $\alpha \in \Sigma_\tau$, $d \in \Delta$ and $x \in \Delta^*$, and we have the following transitions for M.

$$\begin{aligned} q &\xrightarrow{\alpha[d/\#,L]}_M [\, move, x, q' \,] \\ [\, move, x, q' \,] &\xrightarrow{\tau[\#/\#,R]}_M [\, write, x, q' \,] \\ [\, write, ye, q' \,] &\xrightarrow{\tau[\#/d,L]}_M [\, write, y, q' \,] \\ [\, write, \varepsilon, q' \,] &\xrightarrow{\tau[\#/\#,R]}_M q' \end{aligned}$$

for $ye \in \Delta^* \cdot \Delta$ a prefix of $x \in \Delta^*$. In this case, an intermediate step via state $[\, move, x, q' \,]$ is needed to position the tape head on the rightmost blank that is left of the stack-on-tape. After the symbol d as been erased, also here the string x is added on the stack-on-tape from right to left, such that the first element of x is on 'top' of the stack-on-tape, i.e., the first element of x is the leftmost non-blank on the tape.

Now, define the tape content function $tc : \Delta^* \to \mathcal{Z}$ by

$$tc(\varepsilon) = \langle \emptyset \rangle \quad \text{and} \quad tc(dx) = \langle d \rangle x \emptyset$$

Then it can be shown by induction that

$$(q, w, x) \vdash_P (q', w', x') \iff (q, w, tc(x)) \vdash_M^* (q', w', tc(x'))$$

for all $q, q' \in Q$, $w, w' \in \Sigma^*$ and $x, x' \in \Delta^*$. From this it follows that

$$(q, w, x) \vdash_P^* (q', w', x') \iff (q, w, tc(x)) \vdash_M^* (q', w', tc(x'))$$

In particular, taking $q = q_0$ and $q' \in F$, we obtain

$$(q_0, w, \varepsilon) \vdash_P^* (q', \varepsilon, x') \iff (q_0, w, \langle \emptyset \rangle) \vdash_M^* (q', \varepsilon, tc(x'))$$

Thus, since $(q_0', \varepsilon, \langle \# \rangle) \vdash_M^* (q_0, \varepsilon, \langle \emptyset \rangle)$, we conclude $\mathcal{L}(P) = \mathcal{L}(M)$. □

From the theorem it follows that the class of context-free languages is a subset of the class of recursively enumerable languages. In view of Examples 4.8 and 4.9 this inclusion is strict.

Example 4.11. Consider, for an illustration of the application of Theorem 4.10, the PDA P in Figure 4.9, that accepts the language $\{\, ww^R \mid w \in \{a, b\}^+ \,\}$, i.e., non-empty

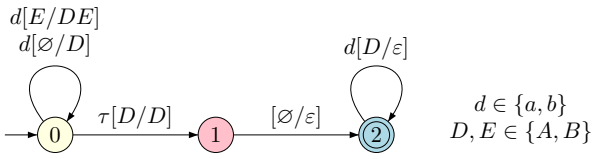


Figure 4.9: PDA accepting non-empty palindromes of even length.

palindromes over $\{a,b\}^*$ of even length. With abuse of notation, for $d \in \{a,b\}$ and $D, E \in \{A,B\}$, we assume the obvious correspondence.

Following the scheme given by the proof we have the following transitions for the reactive Turing machine M that accepts $\mathcal{L}(P)$. To begin with,

$$q_0' \xrightarrow{\tau[\#/\varnothing,L]}_M q_0'' \quad \text{and} \quad q_0'' \xrightarrow{\tau[\#/\#,R]}_M q_0$$

to write down the empty-stack marker $\varnothing$. To emulate the transition $q_0 \xrightarrow{d[\varnothing/D]}_P q_0$, we have

$$\begin{aligned} q_0 &\xrightarrow{d[\varnothing/\varnothing,L]}_M [\,move, D, q_0] \\ [\,move, D, q_0] &\xrightarrow{\tau[\#/\#,R]}_M [\,write, D, q_0] \\ [\,write, D, q_0] &\xrightarrow{\tau[\#/D,L]}_M [\,write, \varepsilon, q_0] \\ [\,write, \varepsilon, q_0] &\xrightarrow{\tau[\#/\#,R]}_M q_0 \end{aligned}$$

To emulate the transition $q_0 \xrightarrow{d[E/DE]}_P q_0$, we similarly have

$$\begin{aligned} q_0 &\xrightarrow{d[E/\#,L]}_M [\,move, DE, q_0] \\ [\,move, DE, q_0] &\xrightarrow{\tau[\#/\#,R]}_M [\,write, DE, q_0] \\ [\,write, DE, q_0] &\xrightarrow{\tau[\#/E,L]}_M [\,write, D, q_0] \\ [\,write, D, q_0] &\xrightarrow{\tau[\#/D,L]}_M [\,write, \varepsilon, q_0] \\ [\,write, \varepsilon, q_0] &\xrightarrow{\tau[\#/\#,R]}_M q_0 \end{aligned}$$

Note that the last two transitions were also used to mimic the PDA transition for q_0 on empty stack. For the transition $q_0 \xrightarrow{\tau[D/D]}_P q_1$ of P we need

$$\begin{aligned} q_0 &\xrightarrow{\tau[D/\#,L]}_M [\,move, D, q_1] \\ [\,move, D, q_1] &\xrightarrow{\tau[\#/\#,R]}_M [\,write, D, q_1] \\ [\,write, D, q_1] &\xrightarrow{\tau[\#/D,L]}_M [\,write, \varepsilon, q_1] \\ [\,write, \varepsilon, q_1] &\xrightarrow{\tau[\#/\#,R]}_M q_1 \end{aligned}$$

Likewise, for the transition $q_1 \xrightarrow{d[D/\varepsilon]}_P q_1$ of the PDA we put in place

$$\begin{aligned} q_1 &\xrightarrow{d[D/\#,L]}_M [\, move, \varepsilon, q_1] \\ [\, move, \varepsilon, q_1] &\xrightarrow{\tau[\#/\#,R]}_M [\, write, \varepsilon, q_1] \\ [\, write, \varepsilon, q_1] &\xrightarrow{\tau[\#/\#,R]}_M q_1 \end{aligned}$$

where the last transition of M was encountered before already. Finally, to emulate the transition $q_1 \xrightarrow{\tau[\varnothing/\varepsilon]}_P q_2$ we have

$$\begin{aligned} q_1 &\xrightarrow{\tau[\varnothing/\varnothing,L]}_M [\, write, \varepsilon, q_2] \\ [\, write, \varepsilon, q_2] &\xrightarrow{\tau[\#/\#,R]}_M q_2 \end{aligned}$$

We have seen that a reactive Turing machine is more powerful than a push-down automaton. However, one may wonder it the computational power of an rTM will be severely less, if instead of using a two-way infinite tape, it comes equiped with a one-way infinite tape. Thus, very much like a stack of a PDA, fresh memory cells can only be recruted unboundly into one direction. This idea brings us to the notion of a reactive Turing machine with semi-infinite tape.

Definition 4.12. A Turing machine with a semi-infinite tape is given by an octuple $M = (\, Q,\, \Sigma,\, \Delta,\, \#,\, \varnothing,\, \rightarrow,\, q_0,\, F\,)$ where Q, Σ, Δ, $\#$, q_0 and F are as before (see Definition 4.2), $\varnothing \notin \Delta_{\#}$, and

$$\rightarrow \subseteq Q \times \Sigma_\tau \times \Delta_{\#} \times \Delta_{\#} \times \{L, R\} \times Q$$

with $\Sigma_\tau = \Sigma \cup \{\tau\}$ and $\Delta_{\#} = \Delta \cup \{\#\}$.

Note, because $\varnothing$ is not an element of $\Delta_{\#}$ by definition, there is no transition possible if the tape-head reads the special symbol $\varnothing$. A Turing machine with a semi-infinite tape can only use the tape cells right of the unique tape cell marked $\varnothing$.

The notion of a configuration for a Turing machine with semi-infinite tape, semi-infinite Turing machine for short, and a reactive Turing machine are similar, as are the notions of a derivation step and of a computation. However, the initial configuration for a semi-infinite Turing machine is of the form $(q_0, w, \langle\varnothing\rangle)$, for some string w. Thus, the tape is marked initially with the special symbol $\varnothing$, and the tape head is initially positioned on the first cell containing the marker. As soon as the tape head reaches the cell marked $\varnothing$, the Turing machine with semi-infinite tape gets stuck. Still, the notion of a language accepted by a Turing machine with semi-infinite tape carries over from a reactive Turing machine. We put $\mathcal{L}(M_1) = \{\, w \in \Sigma^* \mid \exists q \in F, z \in \mathcal{Z}\colon (q_0, w, \langle\varnothing\rangle) \vdash_1^* (q, \varepsilon, z)\,\}$.

Clearly, a reactive Turing machine, with a two-way infinite tape, can simulate a semi-infinite Turing machine. Hence, a reactive Turing machine can accept any language a semi-infinite Turing machine can accept. More precisely,

Theorem 4.13. Let $L \subseteq \Sigma^*$. Suppose L is accepted by a semi-infinite Turing machine M_1, i.e., $\mathcal{L}(M_1) = L$. Then exists a reactive Turing machine M_2 such that $\mathcal{L}(M_2) = L$.

Proof. Put $M_2 = (\, Q,\, \Sigma,\, \Delta \cup \{\varnothing\},\, \#,\, \rightarrow,\, q_0,\, F\,)$. In particular, M_2 has exactly the same transitions and set of final states as M_1 does. Therefore, $L = \mathcal{L}(M_2)$. □

Maybe more surprisingly, the reverse of Theorem 4.13 holds as well.

Theorem 4.14. Let $L \subseteq \Sigma^*$. Suppose L is accepted by a reactive Turing machine M_2. Then exists a semi-infinite Turing machine M_1 such that $\mathcal{L}(M_1) = L$.

Proof. Suppose $M_2 = (\, Q_2,\, \Sigma,\, \Delta_2,\, \#,\, \rightarrow_2,\, q_0,\, F_2\,)$. The semi-infinite Turing machine M_1 will be provided with a 'two-track tape' by augmenting the alphabet Δ with pairs $(e, e') \in \Delta_{\#} \times \Delta_{\#}$. To recognize the cell next to the marker we use new symbols $(e, *)$, for $e \in \Delta_{\#}$, assuming $*$ to be a fresh symbol not in $\Delta_{\#}$. The idea is to fold a two-way infinite tape in two, taking special care at the left-end of the one-way infinite tape. Schematically indicated below, a two-way infinite tape with content $e_{\text{-}3}, e_{\text{-}2}, e_{\text{-}1}, e_0, e_1, e_2, e_3$ by a one-way infinite tape with content $\varnothing\,(e_0, *)(e_1, e_{\text{-}1})(e_2, e_{\text{-}2})(e_3, e_{\text{-}3})$.

$\cdots$	$e_{\text{-}3}$	$e_{\text{-}2}$	$e_{\text{-}1}$	e_0	e_1	e_2	e_3	$\cdots$

$\varnothing$	e_0	e_1	e_2	e_3	$\cdots$
	$*$	$e_{\text{-}1}$	$e_{\text{-}2}$	$e_{\text{-}3}$	$\cdots$

Next, we describe how M_1 will simulate the transitions of M_2. For this we need to keep track whether we are working on the top half of the tape, considering the first symbol e_1 of a pair (e_1, e_2), or on the bottom half of the tape, considering the second symbol e_2 of a pair (e_1, e_2). Moreover, when working on the top half, simulating a movement of M_2 is for M_1 moving in the same direction. However, when working on the bottom half, simulating a movement of M_2 means moving in the *opposite* direction for M_1. To record whether we are working on the top half of the bottom half, we represent may a state q of M_2 as (q, T) or (q, B), where the second component, T and B, indicate top and bottom, respectively. The initial state for M_1 will be state (q_0, T). In case we are at the left end of the semi-infinite take, we need to have special measurement.

We have for a transition $q \xrightarrow{\alpha[e/e',\mu]}_2 q'$ of M_2 the following transitions of M_1.

(i) $(q,T) \xrightarrow{\alpha[(e,\overline{e})/(e',\overline{e}),\mu]}_1 (q',T)$

(ii) $(q,T) \xrightarrow{\alpha[(e,*)/(e',*),R]}_1 (q,T)$ if $\mu = R$

(iii) $(q,T) \xrightarrow{\alpha[(e,*)/(e',*),R]}_1 (q,B)$ if $\mu = L$

(iv) $(q,T) \xrightarrow{\alpha[\#/(e',\#),\mu]}_1 (q,B)$ if $e = \#$

(v) $(q,B) \xrightarrow{\alpha[(\overline{e},e)/(\overline{e},e'),\overline{\mu}]}_1 (q',B)$

(vi) $(q,B) \xrightarrow{\alpha[(e,*)/(e',*),R]}_1 (q,T)$ if $\mu = R$

(vii) $(q,B) \xrightarrow{\alpha[(e,*)/(e',*),R]}_1 (q,B)$ if $\mu = L$

(viii) $(q,B) \xrightarrow{\alpha[\#/(e',\#),\overline{\mu}]}_1 (q,B)$ if $e = \#$

for all $\overline{e} \in \Delta_{\#}$, and where $\overline{\mu}$ is the opposite direction of μ, i.e., $\overline{\mu} = L$ if $\mu = R$, and $\overline{\mu} = R$ if $\mu = L$.

Processing of input for M_1 is exactly the same for M_2. Movement is different because of the tape is only one-way infinite. Moreover, writing needs to be adapted because of the two tracks that are on the tape of M_1.

Transition (i) of M_1 copies directly the transition of M_2, the symbol $\overline{e}$ is ignored since M_1 is working on the upper track. The corresponding transition for the lower

track is transition (v) of M_1. Now, the symbol $\overline{e}$ at the upper track is left untouched; M_1 is operating on the lower track.

When the tape head is at the cell next to the marker, the transitions of M_1 are similar to the situation of other cells. However, M_1 may need to switch from top half to lower half, see transition (iii), or from lower half to top half, see transition (vi). Transitions (iv) and (viii) are in place to deal with the situation that the tape head reads from a cell that wasn't visited before, i.e., the tape head reads an ordinary blank #. This is for example the case for the very first transition in the initial state (q_0, T).

Putting things together, we have for $M_1 = (\, Q_1,\, \Sigma,\, \Delta_1,\, \#,\, \varnothing,\, \to_1,\, (q_0, T),\, F_1\,)$ the following: (i) $Q_1 = \{\, (q,T),\, (q,B) \mid q \in Q_2 \,\}$, (ii) $\Delta_1 = \Delta_\# \times \Delta_\#$, (iii) $\to_1$ as indicated above, (iv) initial state (q_0, T), and (v) final states $F_1 = \{\, (q,T),\, (q,B) \mid q \in F_2 \,\}$. We note, without a formal proof, that for strings $w, w' \in \Sigma^*$ and $q \in F_2$ a computation $(q_0, w, z) \vdash_2^* (q, w', z')$ exists for M_2 for some tape content z, z' iff a computation $((q_0, T), w, \bar{z}) \vdash_1^* ((q, S), w', \bar{z}')$ exists for M_1 for some tape content $\bar{z}, \bar{z}'$ and $S \in \{T, B\}$. □

As another variation of the reactive Turing machine, we consider a restriction on the tape alphabet. However, this doesn't impair the power of acceptance.

Theorem 4.15. Let $L \subseteq \Sigma^*$ be a language such that $L = \mathcal{L}(M)$ for a reactive Turing machine $M = (\, Q,\, \Sigma,\, \Delta,\, \#,\, \to,\, q_0,\, F\,)$ with tape alphabet Δ. Then exists a reactive Turing machine $M' = (\, Q',\, \Sigma,\, \{0,1\},\, \#,\, \to',\, q_0,\, F\,)$ with tape alphabet $\{0,1\}$.

Proof. The general idea is represent the tape alphabet Δ of M by strings of equal length over $\{0,1\}$, and to represent the tape content of M as blank-separated blocks of these strings.

Let $k \geqslant 0$ be such that Δ has strictly less than 2^k elements. Pick a unique string $w_d = d_1 \cdots d_k \in \{0,1\}^k$, $w_d \neq 0^k$, for each element $d \in \Delta$. We use $w_\# = 0^k$ to encode a blank.

We will arrange that in case the tape for M would contain, e.g., $d^1 d^2 d^3 d^4$, and $w_{d^i} = d^i_1 \cdots d^i_k$ for $i = 1, 2, 3, 4$, the tape of M' would look like

#	d^1_1	…	d^1_k	#	d^2_1	…	d^2_k	#	d^3_1	…	d^3_k	#	d^4_1	…	d^4_k	#

If, for example, M would be reading d^2, the tape head of M' is positioned at the blank left of d^2_1. Thus, the bit string $d^2_1 \cdots d^2_k$ is right of the tape head.

A transition $q \xrightarrow{\alpha[d/e',\mu]} q'$ of M, for $d \in \Delta$, $e' \in \Delta_\#$, is emulated by M' by a series of transitions. We assume that the tape head of M' is reading a blank and the bit

representations of d and d' are $w_d = d_1 \cdots d_k$ and $w_{e'} = d'_1 \cdots d'_k$, respectively.

$$\begin{array}{rcl}
q & \xrightarrow{\alpha[\#/\#,R]}{}' & [q, w_d, w_{d'}, \mu, q'] \\
[q, d_j \cdots d_k, e'_j \cdots e'_k, \mu, q'] & \xrightarrow{\tau[d_j/e'_j,R]}{}' & [q, d_{j+1} \cdots d_k, e'_{j+1} \cdots e'_k, \mu, q'] \quad \text{for } 1 \leqslant j \leqslant k \\
[q, \varepsilon, \varepsilon, R, q'] & \xrightarrow{\tau[\#/\#,R]}{}' & q' \\
[q, \varepsilon, \varepsilon, L, q'] & \xrightarrow{\tau[\#/\#,R]}{}' & [\,\ell shift_1, q'] \\
[\,\ell shift_1, q'] & \xrightarrow{\tau[b/b,L]}{}' & [\,\ell shift_1, q'] \quad \text{for } b = 0, 1 \\
[\,\ell shift_1, q'] & \xrightarrow{\tau[\#/\#,L]}{}' & [\,\ell shift_2, q'] \\
[\,\ell shift_2, j, q'] & \xrightarrow{\tau[b/b,L]}{}' & [\,\ell shift_2, j{-}1, q'] \quad \text{for } 1 \leqslant j \leqslant k \text{ and } b = 0, 1, \# \\
[\,\ell shift_2, 0, q'] & \xrightarrow{\tau[\#/\#,L]}{}' & q'
\end{array}$$

In case of a left move of M the tape head of M' needs to move two blocks to the left: one block because of the read of the current block, another block in order to position the tape head at the space preceding the block that needs to be read next. Note, if the block doesn't contain the complete bit string w_d, M' will get stuck.

The explicit counting in the second left shift is in place to cover a situation where the block doesn't contain a bit string, but k blanks instead. Similarly, for a transition $q \xrightarrow{\alpha[\#/e',\mu]} q'$ of M, with $e \in \Delta_{\#}$, we need to take into account that the block hasn't been visited earlier. For this, we mark the intermediate states with a blank.

$$\begin{array}{rcl}
q & \xrightarrow{\alpha[\#/\#,R]}{}' & [q, \#, 0^k, \mu, q'] \\
[q, \#, 0^j, \mu, q'] & \xrightarrow{\tau[\#/0,R]}{}' & [q, \#, 0^{j-1}, \mu, q'] \quad \text{for } 1 \leqslant j \leqslant k \\
[q, \#, \varepsilon, R, q'] & \xrightarrow{\tau[\#/\#,R]}{}' & q' \\
[q, \#, \varepsilon, L, q'] & \xrightarrow{\tau[\#/\#,R]}{}' & [\,\ell shift_1, q']
\end{array}$$

In view of the above we have $M' = (\, Q', \Sigma, \{0,1\}, \#, \to', q_0, F\,)$ where

$$\begin{array}{rcl}
Q' & = & Q \cup \{\, [\,\ell shift_1, q'], [\,\ell shift_2, j, q'] \mid q' \in Q,\, 0 \leqslant j \leqslant k \,\} \cup \\
& & \{\, [q, v, v', \mu, q'] \mid \exists \alpha{\in}\Sigma_\tau \exists d, d'{\in}\Delta \exists \mu{\in}\{L, R\}: \\
& & \qquad q \xrightarrow{\alpha[d/d',\mu]} q',\, v \preccurlyeq w_d,\, v' \preccurlyeq w_{d'},\, |v| = |v'| \,\}
\end{array}$$

and transition relation $\to'$ as indicated above. We note, without proof, that M' can perform every computation from the initial state $\square$

Exercises for Section 4.1

Exercise 4.1.1. Construct a reactive Turing machine for the language $L = \{\, a^n b^m c^\ell \mid n, m, \ell \geqslant 0 \,\}$. Give an accepting computation sequence for the string *abbccc*. Argue why the strings *aaccbb* and *bca* are not accepted. A proof of correctness is not asked for.

Exercise 4.1.2. Construct, for the language $L = \{ a^n b^m c^{n+m} \mid n, m \geqslant 0 \}$, a reactive Turing machine. Give an accepting computation sequence for the string *aaabcccc*. Argue why the strings *aabbcc* and *abccc* are not accepted. A proof of correctness is not asked for.

Exercise 4.1.3. Construct a reactive Turing machine for the language $L = \{ ww^R \mid w \in \{a, b\}^* \}$. Give an accepting computation sequence for the string *aabbaa*. A proof of correctness is not asked for.

Exercise 4.1.4. Construct a Turing machine for the language $L = \{ a^n b^n c^n \mid n > 0 \}$ that has at most one τ-move. A proof of correctness is not asked for.

Exercise 4.1.5. Construct a reactive Turing machine for the language $L = \{ w \in \{a, b\}^* \mid \#_a(w) = 2 * \#_b(w) \}$ with at most 4 states. A proof of correctness is not asked for.

4.2 The classical Turing machine

The reactive Turing machine introduced in the previous section is used as a language acceptor. However, with some conventions and simplifications a Turing machine can be used to compute a function too. The latter variant of a Turing machine is referred to as the classical Turing machine.

Definition 4.16 (Classical Turing machine)**.** A classical Turing machine is a quintuple $M = (\, Q, \Delta, \#, \rightarrow, q_0 \,)$ where Q, Δ, $\#$, and q_0 are as before (see Definition 4.2), and

$$\rightarrow \subseteq Q \times \Delta_{\#} \times \Delta_{\#} \times \{L, R\} \times Q$$

with $\Delta_{\#} = \Delta \cup \{\#\}$.

If, for a Turing machine M as given above, $(q, e, e', \mu, q') \in \rightarrow$ we write $q \xrightarrow{e/e',\mu}_M q'$. This means that a classical Turing machine, when it is in state q and reads symbol e on the tape, can replace e by e', move one cell left if $\mu = L$ and one cell right if $\mu = R$, and thereby change control to state q'. There is no input alphabet Σ as for the reactive Turing machine; there are no τ-transitions either. Also, there are no final states. For a classical Turing machine there is no notion of acceptance, but there is a notion of termination as we will see below.

Example 4.17. An example of a classical Turing M is given in Figure 4.10. We have $\Delta = \{a, b\}$. Note that the Turing machine has no transition for the symbol b in state q_0 and no transition for the symbols a and $\#$ in state q_2. Thus, if in state q_0 the symbol b

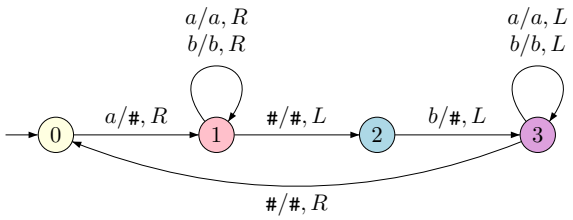


Figure 4.10: A classical Turing machine.

is read on tape, the Turing machine M blocks and the computation halts. Similarly, if in state q_2 either the symbol a or a blank is read, the Turing machine M blocks.

Suppose M starts in state q_0, with the string $aaabbb$ written on tape and with the tape head on the leftmost a. Then M erases this a, the tape head moves to the right, control to state q_1. Then the string $aabbb$ is skipped while the tape head moves to the right, and M reads a blank right after the string of a's and b's. M moves to the left and arrives in state q_2. Then the rightmost b is read, erased, the tape head moves left, control moves to state q_3. Then the Turing machine skips over the remaining string $aabb$, till it reads the first blank on the left (where the cell where the tape head started from initially). Then the tape head moves right reading the symbol a, control is now in state q_0.

To describe the behaviour concisely, we adapt the notion of a configuration to the situation of a classical Turing machine. Let $M = (\, Q,\, \Delta,\, \#,\, \rightarrow,\, q_0\,)$ be a classical Turing machine according to Definition 4.17. A configuration or an ID of M is a pair $(q, z) \in Q \times \mathcal{Z}$ with $\mathcal{Z}$ as defined in the previous section. We write

$$(q, z) \vdash_M (q', z') \quad \text{iff} q \xrightarrow{e/e',\mu}_M q' \quad \text{and} \quad z' = z[e/e', \mu]$$

for some $e, e' \in \Delta_{\#}$ and $\mu \in \{L, R\}$. The update $z[e/e', \mu]$ for $[e/e', \mu]$ on the configuration z is as before. Here, for $z' = z[e/e', \mu]$, the tape content z' is obtained from z by replacing the eye e of z by e' and moving the tape head in the direction indicated by μ. Note that this replacement requires that z is of the form $x\langle e\rangle y$; otherwise the operation is not defined.

We write $(q, z) \nvdash_M$ if for no q' and z' we have $(q, z) \vdash_M (q', z')$. If $z = x\langle e\rangle y$ and there is no transition $q \xrightarrow{e/e',\mu}_M q'$ with $q' \in Q$, $e' \in \Delta_{\#}$ and $\mu \in \{L, R\}$, we say that M halts or blocks in (q, z). As there is no further transition, any computation leading to the configuration (q, z) terminates there.

For the classical Turing machine M of Figure 4.10 we have the following computation:

$$\begin{aligned}
&(q_0, \langle a\rangle aabbb) \vdash_M (q_1, \langle a\rangle abbb) \vdash_M (q_1, a\langle a\rangle bbb) \vdash_M (q_1, aa\langle b\rangle bb) \vdash_M \\
&\quad (q_1, aab\langle b\rangle b) \vdash_M (q_1, aabb\langle b\rangle) \vdash_M (q_1, aabbb\langle \#\rangle) \vdash_M (q_2, aabb\langle b\rangle) \vdash_M \\
&\quad\quad (q_3, aab\langle b\rangle) \vdash_M (q_3, aa\langle b\rangle b) \vdash_M (q_3, a\langle a\rangle bb) \vdash_M (q_3, \langle a\rangle abb) \vdash_M \\
&\quad\quad\quad (q_3, \langle \#\rangle aabb) \vdash_M (q_0, \langle a\rangle abb)
\end{aligned}$$

Thus $(q_0, \langle a\rangle aabbb) \vdash_M^* (q_0, \langle a\rangle abb)$. However, M does not halt in $(q_0, \langle a\rangle abb)$, since $(q_0, \langle a\rangle abb)$ admits a further transition, viz. to $(q_1, \langle a\rangle bb)$. The computation is not complete.

It also holds that $(q_0, \langle a\rangle aabbb) \vdash_M^* (q_0, \langle \# \rangle)$. Since $(q_0, \langle \# \rangle) \nvdash_M$ the computation started in the configuration $(q_0, \langle a\rangle aabbb)$ is complete. It terminates in the configuration $(q_0, \langle \# \rangle)$.

We are about to introduce the notion of a function *computed* by a classical Turing machine. However, we first need a means to relate tape content and strings.

We will use the notation $\langle w\rangle$, for a string $w \in \Delta^*$, to denote a tape containing w and with the tape head at the leftmost symbol of w, if available. Formally, $\langle \varepsilon\rangle = \langle \# \rangle$ and $\langle dw\rangle = \langle d\rangle w$. Thus, we have $\langle\cdot\rangle : \Delta^* \to \mathcal{Z}$. Note, for a tape content z and non-empty string w, if $z = \langle w\rangle$ then z contains a consecutive block of symbols from Δ comprising w, the tape is empty everywhere else. If $w = \varepsilon$ is the empty string, $\langle \varepsilon\rangle = \langle \# \rangle$ represents the empty tape.

Definition 4.18. A classical Turing machine $M = (\, Q, \Delta, \#, \to, q_0\,)$ computes a function $f : \Omega \to \Theta^*$ for $\Omega \subseteq \Sigma^*$ and two alphabets Σ and Θ (with $\Sigma \cup \Theta \subseteq \Delta$) iff, for all strings $w \in \Sigma^*$, we have

(i) termination, i.e., $(q_0, \langle w\rangle) \vdash_M^* (q, z) \nvdash_M$ for some $q \in Q$, $z \in \mathcal{Z}$ with $z = \langle f(w)\rangle$;

(ii) determinacy, i.e., if $(q_0, \langle w\rangle) \vdash_M^* (q_1, z_1) \nvdash_M$ and $(q_0, \langle w\rangle) \vdash_M^* (q_2, z_2) \nvdash_M$, for $q_1, q_2 \in Q$, $z_1, z_2 \in \mathcal{Z}$, then $q_1 = q_2$ and $z_1 = z_2$.

For the classical Turing machine M to compute the function value $f(w) \in \Theta^*$ for the string $w \in \Omega$, the scheme is to first write the string w on tape and start the Turing machine M in its initial state q_0 with the tape head on the leftmost symbol of w, i.e. in the configuration $(q_0, \langle w\rangle)$. When the Turing machine terminates, say in the configuration (q, z), the tape should contain a unique string from Θ^*, called $f(w)$. Moreover the tape head is supposed to be at the leftmost symbol of $f(w)$ is it is non-empty.

For this to work, there must be at least one terminating computation for M starting from $(q_0, \langle w\rangle)$ which yields a result in Θ^*. Moreover, every terminating computation should yield the same result. The former is captured by the first condition of the definition. Regarding the latter, this condition is guaranteed if the transition relation $\to_M$ represents a partial function $Q \times \Delta_\# \to \Delta_\# \times \{L, R\} \times Q$, i.e. for every $q \in Q$, $e \in \Delta_\#$ there is at most one triple $e' \in \Delta_\#$, $\mu \in \{L, R\}$, $q' \in Q$ such that $q \xrightarrow{e/e',\mu}_M q'$.

Example 4.19. Figure 4.11 describes a classical Turing machine, say M, that computes the copying function $copy : \{a, b\}^* \to \{a, b\}^*$ with $copy(w) = ww$. The trick is to use the auxiliary symbols A and B to describe a's and b's that are already processed. Thus we have $\Delta = \{a, b, A, B\}$. However, $\Sigma = \Theta = \{a, b\}$. We have $\Omega = \Sigma^*$. The computation in q_0 starts on the leftmost non-blank symbol on the tape. Then M scans to the right for the first a or b. If there is none, the copying is done and M changes control to state q_5

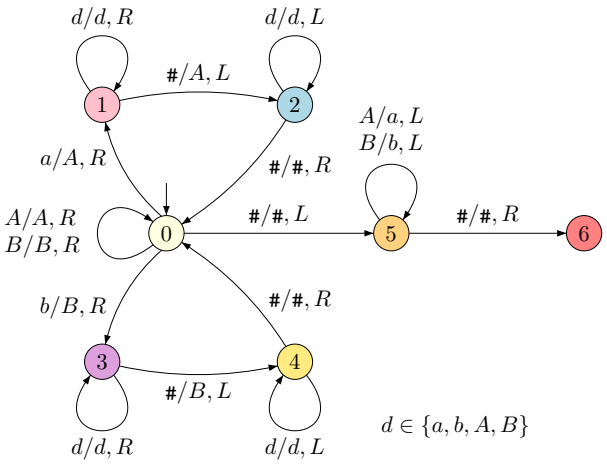


Figure 4.11: Classical Turing machine for the copying function $f(w) = ww$.

where is replaces all A's and B's by a's and b's, respectively. If a blank is in the eye of the tape head, it moves right and then the Turing machine blocks.

If in q_0 an a is read, this symbol needs to be copied at the end of the tape content. The current a is marked as processed, i.e. replaced by A. In state q_1 all non-blanks are skipped, $d/d, R$ abbreviates $a/a, R$, $b/b, R$, $A/A, R$ and $B/B, R$. When the blank at the end of the tape content is reached, an A is written and control is in state q_2. There the tape head is moved to the left of the tape content. When reached, M is in state q_0 with the tape head on the leftmost non-blank symbol (if any).

If in q_0 a b is read, a similar sequence of transitions follows. Now the symbol b is overwritten with B, control changes to state q_3 and a B is written at the end of the tape content, after which the tape head moves the leftmost symbol on tape and control moves to state q_0.

Dependent on the input string w, for suitable $U \in \{A, B\}^*$, we have

$$(q_0, U\langle a\rangle v U) \vdash_M^* (q_1, UAvU\langle \# \rangle) \quad \text{and} \quad (q_0, U\langle b\rangle v U) \vdash_M^* (q_3, UBvU\langle \# \rangle)$$

Functions with more than one argument can be computed too with a classical Turing machine. For example, for a two-ary function $f(w_1, w_2)$, that takes two strings w_1 and w_2 to produce a result, we put both strings on tape, first w_1 then w_2, have them separated by a blank, or an other symbol if desired, and have the tape head positioned at the first, i.e., leftmost symbol of w_1 (or at the separating symbol if $w_1 = \varepsilon$).

Example 4.20. As a first example of a function computing with numbers we consider subtraction for non-negative integer numbers in unary notation. See Figure 4.12. In unary notation a number n is represented by the string 1^n, i.e. a string of n times the symbol 1. For simplicity, we assume that the two numbers involved are positive integers. As discussed above, we choose the numbers to be separated by a minus sign '$-$'. Thus the input to compute $n - m$ is the string $1^n{-}1^m \in \{1, \#\}^*$ with $n, m \geqslant 1$. As we are

dealing with non-negative numbers the convention is that the result is 0 if $n \leqslant m$. More concretely, $4 - 3 = 1$ and $3 - 4 = 0$.

The classical Turing machine of Figure 4.12 can be dissected in four parts: The path from the initial state q_0 up to state q_2, the cycle of state q_2 via state q_5, and the two paths to a state without outgoing transitions, viz. from state q_5 to state q_9 and from state q_2 to state q_{11}.

Starting in state q_0, the first concern is to find the right end of the arguments. So, we first skip to the right over the first number in state q_0, skip over the minus sign separating the two numbers, and skip over the second number in state q_1.

The loop of state q_2 visiting q_3, q_4, q_5, q_6 and q_7, respectively, is erasing a 1-symbol from the second number against a 1-symbol of the first number. This subcomputation starts at the rightmost symbol of the second number. In q_2 if a 1-symbol is read it is erased, the tape head skips to the left, skipping the remaining 1's of the second number in q_3, skipping the separator $-$, skipping the remaining 1's of the first number in q_4 till the blank at the left is found. The tape head moves one position back, control is state q_5. If a matching 1-symbol of the first number is read, this symbol is erased, and the tape head skips to the right, reaching via q_6, where the remaining 1's of the first number are skipped, and via q_7 where the remaining 1's of the second number are skipped the start of the cycle, state q_2 again with the tape head on the rightmost symbol, if any.

However, if in state q_5 no matching 1 is found, i.e. the tape head reads the separator rather than a 1, the result of the computation should be zero, since the second number proves larger than the first number. The Turing machine is in configuration $(q_5, \langle - \rangle 1^{m-n-1})$. Note, in this case $m > n$. So next, all 1 are swiped from the tape in state q_{10}. This stops when a blank is found. The computation terminates and the tape is empty, representing $1^0 = \varepsilon$.

Alternatively, in state q_2 if no 1 is read any more, thus we are reading the separator $-$ rather than a 1-symbol of the second number, we are basically done. The configuration of the Turing machine is $(q_2, 1^{n-m}\langle - \rangle)$. We first need to clean up the separator, as this is no part of the output, and then move the tape head to the leftmost 1 (if any). The computation terminates leaving the result 1^{n-m} on tape.

Example 4.21. A Turing machine computing addition of binary numbers is depicted in Figure 4.13. For simplicity, we assume the two numbers that are to be added to have an equal number of digits, having leading 0's padded to the shortest number if necessary. The numbers are separated by a plus sign '+'. The tape alphabet is the set $\Delta = \{\, 0,\, 1,\, +,\, Z,\, W,\, T \,\}$. The digits 0 and 1 as well as the $+$-sign speak for themselves. The capitals Z, W and T represent processed digits: Z for 0 (zero), W for 1 (one) (to avoid confusion between the digit 0 and the capital O we use W instead), T for 2 (two) being the sum of two digits 1.

The Turing machine computes a function $+ : \Omega \to \Delta^*$. Only in case w is a string of the form $w_1 + w_2$ with $w_1, w_2 \in \{0,1\}^n$ for some $n \geqslant 1$, we assure that the binary representation of the number $w_1 + w_2$ is computed. So, $\Omega = \bigcup_{n=1}^{\infty} \{0,1\}^n \cdot \{+\} \cdot \{0,1\}^n$. For conciseness, in Figure 4.13 we use the symbol d to range over $0, 1$, the symbol D to range over Z, W, T, and the symbol e to range over all non-blanks.

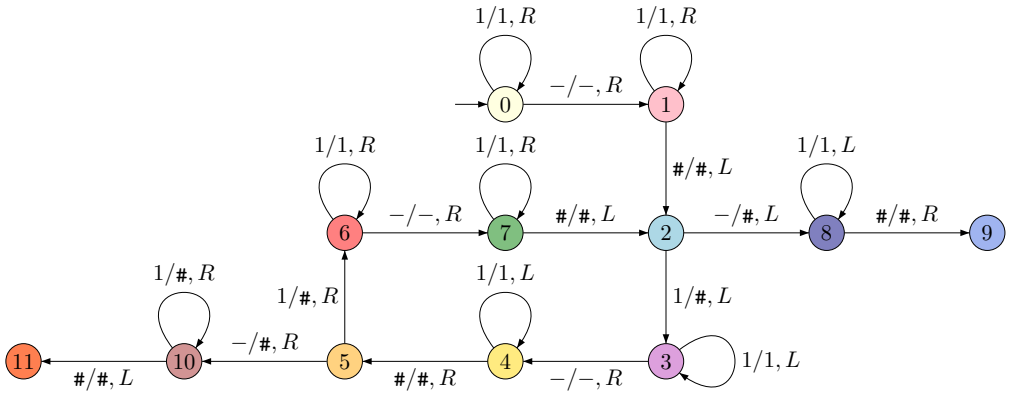


Figure 4.12: Classical Turing machine for unary subtraction.

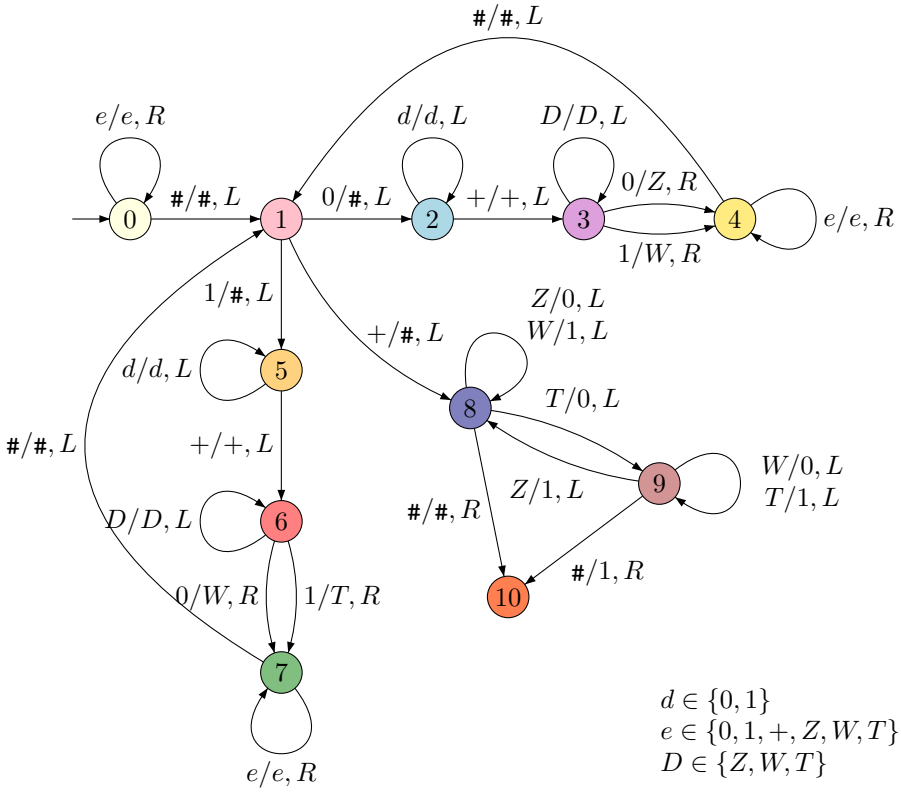


Figure 4.13: A classical Turing machine for binary addition.

- Starting in q_0 with the tape head at the leftmost digit of the first number, the tape head is moved to the blank right of the second number. In state q_1 the tape head is positioned at the rightmost digit of the second number.

- If in q_1 the digit 0 of the second number is read, the idea is to change the corresponding digit of the first number into Z or W, dependent of the value of the latter digit. To this end the tape head wobbles to the left. Skipping over 0's and 1's first in state q_2, then encountering a $+$, then skipping marked digits, i.e. capitals, in state q_3 and encountering the first digit to the left of the $+$. The digit 0 is marked as Z; we have read a 0 of the second number and $0+0=Z$. The digit 1 is marked as W; we have a read a 0 of the second number and $1+0=W$. In state q_4 the tape head moves to the right in search of the rightmost blank. Once found, the tape head is positioned on the rightmost digit of the second number. Control is in state q_1.

- If in q_1 the digit 1 of the second number is read, the Turing machine will change the corresponding digit of the first number into W or T, again dependent on the value of the latter digit. In state q_5 the tape head moves left over the remainder of the second number, changing control to state q_6 if the $+$-sign is scanned. In state q_6 capitals are skipped until the first digit on the left of the $+$-sign is found. If it is the digit 0 the symbol W is written since $0+1=W$. If it is the digit 1 the symbol T is written since $1+1=T$. No carry is processed now. This will be done at a later stage.

- Note in state q_1 the digits of the second number are erased from right to left. If all digits of the second number have vanished the $+$-sign remains as rightmost non-blank. If detected in state q_1 the marking phase is finished, the $+$-sign is erased and control moves to state q_8. There we change capitals back into digits and take a possible carry into account.

- State q_8 represents a situation where there is no carry; state q_9 represents a situation where there is a carry. The tape head moves right to left over the symbols Z, W and T. If in q_8 a Z is read a 0 is written. If in q_8 a W is read a 1 is written. However, if in q_8 a T is read a 0 is written, but since a carry is to be remembered control changes to state q_9. Similarly, in state q_9 on symbol Z the symbol 1 is written, because of the carry, but since the carry has been handled control returns to state q_8. On symbol W a 0 is written but a carry remains. On symbol T a 1 is written and again a carry remains.

- Both in q_8 and in q_9 the computation is done if a blank is encountered. Then we have scanned over all of the marked digits, i.e. the string of Z's, W's and T's that replaced the first number. However, in q_8 nothing is added; in q_9 the carry is processed by writing a final 1 in front of the string of digits written so far.

Exercises for Section 4.2

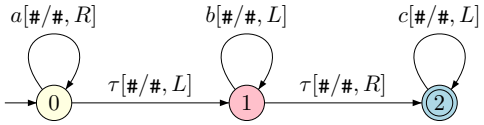
Exercise 4.2.1. Construct a classical Turing machine that computes a function $p : \{a, b\}^* \to \{Y, N\}^*$ such that $p(w) = Y$ if w is a palindrome, i.e., $w = w^R$, and $p(w) = N$ if w is *not* a palindrome. Give a computation sequence for the strings *ababa* and *abba* producing Y, and for the strings the strings *aaba* and *baa* producing N. A proof of correctness is not asked for.

Exercise 4.2.2. Construct a classical Turing machine that computes a function ${}^2\log : \{1\}\cdot\{0,1\}^* \to \{0,1\}^*$ such that ${}^2\log(w) = n$ if $2^n \leqslant w < 2^{n+1}$ with the strings w and n interpreted as a binary numbers. E.g., ${}^2\log(24) = {}^2\log(11000_2) = 100_2 = 4$ since $2^4 = 16 \leqslant 24 < 32 = 2^5$.

Answers to exercises from Chapter 4

Answers to exercises from Section 4.1

Answer to Exercise 4.1.1 The reactive Turing machine below accepts the language $L = \{\, a^n b^m c^\ell \mid n, m, \ell \geqslant 0 \,\}$.



An accepting computation sequence for the string *abbccc* is

$$(q_0, abbccc, \langle\#\rangle) \vdash_M (q_0, bbccc, \langle\#\rangle) \vdash_M (q_1, bbccc, \langle\#\rangle) \vdash_M (q_1, bccc, \langle\#\rangle) \vdash_M (q_1, ccc, \langle\#\rangle) \vdash_M (q_2, ccc, \langle\#\rangle) \vdash_M (q_2, cc, \langle\#\rangle) \vdash_M (q_2, c, \langle\#\rangle) \vdash_M (q_2, \varepsilon, \langle\#\rangle)$$

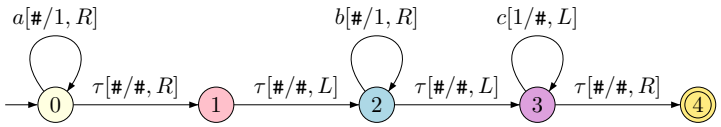
For the strings *aaccbb* and *bca* we have maximally input processing derivations

$$(q_0, aaccbb, \langle\#\rangle) \vdash_M (q_0, accbb, \langle\#\rangle) \vdash_M (q_0, ccbb, \langle\#\rangle) \vdash_M (q_1, ccbb, \langle\#\rangle) \vdash_M (q_2, ccbb, \langle\#\rangle) \vdash_M (q_2, cbb, \langle\#\rangle) \vdash_M (q_2, bb, \langle\#\rangle) \nvdash_M, \quad \text{and}$$

$$(q_0, bca, \langle\#\rangle) \vdash_M (q_1, bca, \langle\#\rangle) \vdash_M (q_1, ca, \langle\#\rangle) \vdash_M (q_2, ca, \langle\#\rangle) \vdash_M (q_2, a, \langle\#\rangle) \nvdash_M$$

that get stuck, but no computations processing all of the input.

Answer to Exercise 4.1.2 The reactive Turing machine below accepts the language $L = \{\, a^n b^m c^{n+m} \mid n, m \geqslant 0 \,\}$.



For the string *aaabcccc* we have the accepting computation sequence

$$(q_0, aaabcccc, \langle \# \rangle) \vdash_M (q_0, aabcccc, 1\langle \# \rangle) \vdash_M (q_0, abcccc, 11\langle \# \rangle) \vdash_M$$
$$(q_0, bcccc, 111\langle \# \rangle) \vdash_M (q_1, bcccc, 111\#\langle \# \rangle) \vdash_M (q_2, bcccc, 111\langle \# \rangle) \vdash_M$$
$$(q_2, cccc, 1111\langle \# \rangle) \vdash_M (q_3, cccc, 111\langle 1 \rangle) \vdash_M (q_3, ccc, 11\langle 1 \rangle) \vdash_M$$
$$(q_3, cc, 1\langle 1 \rangle) \vdash_M (q_3, c, \langle 1 \rangle) \vdash_M (q_3, \varepsilon, \langle \# \rangle)(q_4, \varepsilon, \langle \# \rangle)$$

However, for the strings *aabbcc* and *abccc* we have as maximally input consuming computations

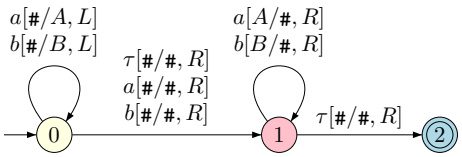
$$(q_0, aabbcc, \langle \# \rangle) \vdash_M (q_0, abbcc, 1\langle \# \rangle) \vdash_M (q_0, bbcc, 11\langle \# \rangle) \vdash_M$$
$$(q_1, bbcc, 11\#\langle \# \rangle) \vdash_M (q_2, bbcc, 11\langle \# \rangle) \vdash_M (q_2, bcc, 111\langle \# \rangle) \vdash_M$$
$$(q_2, cc, 1111\langle \# \rangle) \vdash_M (q_3, cc, 111\langle 1 \rangle) \vdash_M (q_3, c, 11\langle 1 \rangle) \vdash_M (q_3, \varepsilon, 1\langle 1 \rangle) \nvdash_M$$

and

$$(q_0, abccc, \langle \# \rangle) \vdash_M (q_0, bccc, 1\langle \# \rangle) \vdash_M (q_1, bccc, 1\#\langle \# \rangle) \vdash_M (q_2, bccc, 1\langle \# \rangle) \vdash_M$$
$$(q_2, ccc, 11\langle \# \rangle) \vdash_M (q_3, ccc, 1\langle 1 \rangle) \vdash_M (q_3, cc, \langle 1 \rangle) \vdash_M (q_3, c, \langle \# \rangle) \nvdash_M$$

Therefore, these strings are not accepted by the Turing machine.

Answer to Exercise 4.1.3 The reactive Turing machine below accepts the language $L = \{ ww^R \mid w \in \{a, b\}^* \}$.



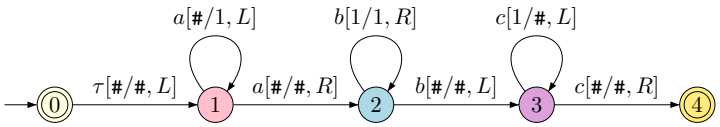
Accepting computation sequences for the string *aaabbbccc* and *ababa* are the following.

$$(q_0, aabbaa, \langle \# \rangle) \vdash_M (q_0, abbaa, \langle \# \rangle A) \vdash_M (q_0, bbaa, \langle \# \rangle AA) \vdash_M$$
$$(q_0, baa, \langle \# \rangle BAA) \vdash_M (q_1, baa, \langle B \rangle AA) \vdash_M (q_1, aa, \langle A \rangle A) \vdash_M$$
$$(q_1, a, \langle A \rangle) \vdash_M (q_1, \varepsilon, \langle \# \rangle) \vdash_M (q_2, \varepsilon, \langle \# \rangle)$$

and

$$(q_0, ababa, \langle \# \rangle) \vdash_M (q_0, baba, \langle \# \rangle A) \vdash_M (q_0, aba, \langle \# \rangle BA) \vdash_M$$
$$(q_1, ba, \langle B \rangle A) \vdash_M (q_1, a, \langle A \rangle) \vdash_M (q_1, \varepsilon, \langle \# \rangle) \vdash_M (q_2, \varepsilon, \langle \# \rangle)$$

Answer to Exercise 4.1.4 The reactive Turing machine below accepts the language $L = \{ a^n b^n c^n \mid n > 0 \}$. Note that the machine has exactly one τ-move.

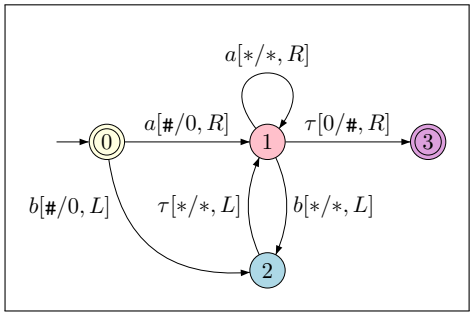


For the string *aaabbbccc* we have the accepting computation sequence

$$(q_0, aaabbbccc, \langle\#\rangle) \vdash_M (q_1, aaabbbccc, \langle\#\rangle) \vdash_M (q_1, aabbbccc, \langle\#\rangle 1) \vdash_M$$
$$(q_1, abbbccc, \langle\#\rangle 11) \vdash_M (q_2, bbbccc, \langle 1\rangle 1) \vdash_M (q_2, bbccc, 1\langle 1\rangle) \vdash_M$$
$$(q_2, bccc, 11\langle\#\rangle) \vdash_M (q_3, ccc, 1\langle 1\rangle) \vdash_M (q_3, cc, \langle 1\rangle) \vdash_M$$
$$(q_3, c, \langle\#\rangle) \vdash_M (q_4, \varepsilon, \langle\#\rangle)$$

Also note, since q_0 is a final state the Turing machine accepts the empty string ε.

Answer to Exercise 4.1.5 The reactive Turing machine below accepts the language $L = \{ w \in \{a, b\}^* \mid \#_a(w) = 2 * \#_b(w) \}$. The idea is that the number of cells that the tape head is to the right of the marker 0 indicates the surplus of a's, the number of cells that the tape head is to the cells indicates twice the surplus of b's.



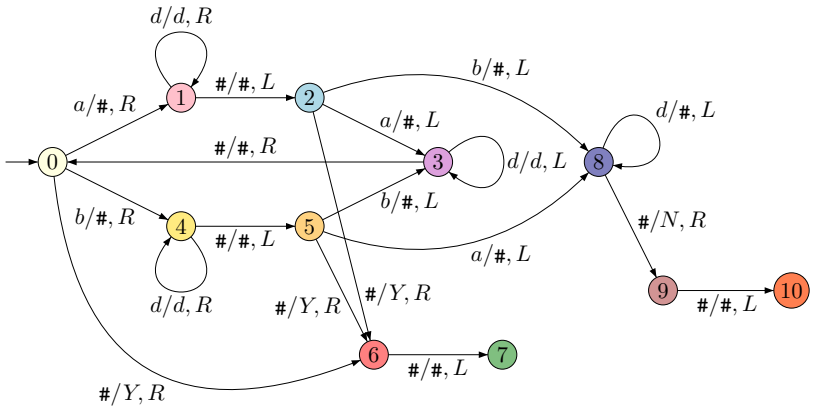
Here, a transition labeled $\alpha[*/*, \mu]$, for $\alpha = a, b$, $\mu = L, R$ abbreviates two transitions, viz. one labeled $\alpha[0/0, \mu]$ and one labeled $\alpha[\#/\#, \mu]$.

An accepting derivation for *baaaab* is the following

$$(q_0, baaaab, \langle\#\rangle) \vdash_M (q_2, aaaab, \langle\#\rangle 0) \vdash_M (q_1, aaaab, \langle\#\rangle\#0) \vdash_M$$
$$(q_1, aaab, \langle\#\rangle 0) \vdash_M (q_1, aab, \langle 0\rangle) \vdash_M (q_1, ab, 0\langle\#\rangle) \vdash_M (q_1, b, 0\#\langle\#\rangle) \vdash_M$$
$$(q_2, \varepsilon, 0\langle\#\rangle) \vdash_M (q_1, \varepsilon, \langle 0\rangle) \vdash_M (q_3, \varepsilon, \langle\#\rangle)$$

Answers to exercises from Section 4.2

Answer to Exercise 4.2.1



1.		$(q_0, \langle a\rangle baba)$	7.	$\vdash$	$(q_0, \langle b\rangle ab)$	13.	$\vdash$	$(q_0, \langle a\rangle)$
2.	$\vdash$	$(q_1, \langle b\rangle aba)$	8.	$\vdash$	$(q_4, \langle a\rangle b)$	14.	$\vdash$	$(q_1, \langle \#\rangle)$
3.	$\vdash^*$	$(q_1, baba\langle \#\rangle)$	9.	$\vdash^*$	$(q_4, ab\langle \#\rangle)$	15.	$\vdash$	$(q_2, \langle \#\rangle)$
4.	$\vdash$	$(q_2, bab\langle a\rangle)$	10.	$\vdash$	$(q_5, a\langle b\rangle$	16.	$\vdash$	$(q_6, Y\langle \#\rangle)$
5.	$\vdash$	$(q_3, ba\langle b\rangle)$	11.	$\vdash$	$(q_3, \langle a\rangle)$	17.	$\vdash$	$(q_7, \langle Y\rangle)$
6.	$\vdash^*$	$(q_3, \langle \#\rangle bab)$	12.	$\vdash$	$(q_3, \langle \#\rangle a)$			

Computations for *abba*, *aaba*, and *baa* not elaborated.

Answer to Exercise 4.2.2 We use the fact that $n = |w| - 1$. So, we increment the starting value 0 as many times as there are digits following w's leading 1.