Contents

Chapter 3

Languages as Accepted Sets of Words

In the preceding chapter we have defined languages as sets of words generated by grammars. This means that we have considered a mechanism which produces the words of the languages. In this chapter we will go the other way around. We define devices which, for a given word, tell whether the word belongs to a language. A little bit more precise we associate with any such device a set of words for which the device gives a positive answer.

3.1 Turing Machines versus Phrase Structure Grammars

3.1.1 Turing Machines and Their Accepted Languages

The first question is to define the device. we aim at a very simple device. We only require that it changes the contents of storage cells in dependence on the information it has. The contents of a storage cell is a letter of some word, and the operations performed by the device is a simple change of the letter and a move to another storage cell. Formally, we get the following notion which was first introduced and investigated by Alan Turing[1] in 1935.

Definition 3.1 *A* Turing machine *is a sextuple*

$$\mathcal{M} = (X, Z, z_0, Q, F, \delta),$$

with

- *the alphabet X of input symbols,*
- *the alphabet Z of states,*
- *the initial state $z_0 \in Z$,*
- *the set Q of halting states, where $\emptyset \subseteq Q \subseteq Z$,*

[1]1912 – 1954, English logician, mathematician, and cryptoanalyst, worked in Princeton (USA) and Cambridge (England), during World War II involved in the decoding of the Enigma, invented mathematical methods in biology

– *the set F of accepting states, where $\emptyset \subseteq F \subseteq Q$, and*
– *the transition function $\delta : (Z \setminus Q) \times (X \cup \{*\}) \to Z \times (X \cup \{*\}) \times \{R, L, N\}$, where the blank symbol $*$ is not a symbol from X.*

The states of $Q \setminus F$ are called rejecting states.

To justify the notion "machine" we give the following interpretation. A Turing machine consists of

– a control box, which at any moment is in a certain state which represents the information stored by the machine,
– a tape, which is unbounded to the right and to the left and is divided into cells where any cell contains a letter of X or the blank symbol, and at any moment only a finite number of cells contains a letter of X,
– a read/write head, which can read the symbol in a square and can overwrite a symbol.

At any moment, the machine is in some state $z \in Z \setminus Q$ and the head scans some symbol $x \in X \cup \{*\}$ in some cell. Figure 3.1 gives an illustration of an Turing machine.

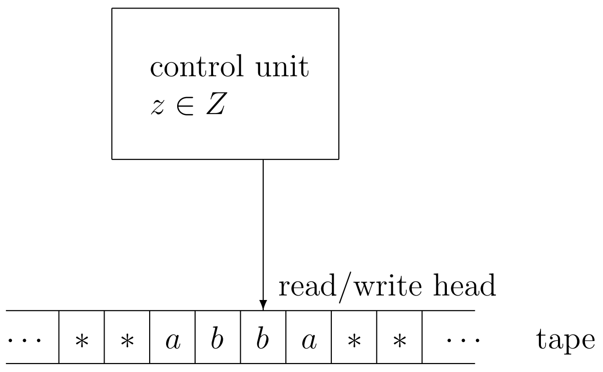

Figure 3.1: Turing machine

Thus the situation of a Turing machine is given by its current state, by the word (or the words) over X on the tape, and the position, where the head is. We formalize this intuitive idea by the following definition.

Definition 3.2 *Let $\mathcal{M}$ be a Turing machine as in Definition 3.1. A* configuration *K of the Turing machine $\mathcal{M}$ is a triple*

$$K = (w_1, z, w_2),$$

where w_1 and w_2 are words over $X \cup \{*\}$ and $z \in Z$.

A configuration is called initial *if $w_1 = \lambda$ and $z = z_0$.*

A configuration is called halting *if $z \in Q$.*

A configuration is called accepting *if $z \in F$.*

The interpretation of a configuration (w_1, z, w_2) is as follows: The machine is in state z, the word w_1w_2 is written on the tape (and all remaining cells are filled with the blank symbol $*$), and the head is above the first letter of w_2. We note that any configuration describes a unique situation of the machine, but a given situation can be described by an infinite number of configurations. For instance, any of the configurations (λ, z, ab), $(*, z, ab)$ und $(**, z, ab*)$ describes the situation, where the Turing machine is in state z, ab stands on the tape, and the read/write head is above a. It is easy to see that one gets a unique configuration for each situation if one requires that the first letter of w_1w_2 is $*$ and the head is above the first letter or the first letter of w_1w_2 is contained in X and the last letter of w_1w_2 is $*$ and the head is above the last letter or the last letter of w_1w_2 is contained in X.

An initial configuration is given if the Turing machine is in its initial state z_0 and the head is above the first letter.

A halting configuration is characterised by the requirement that the machine is in an halting state of Q.

The machine works as follows. If the Turing machine is in the state $z \in Z \setminus Q$, the head scans the symbol $x \in X \cup \{*\}$, and $\delta(z, x) = (z', x', r)$ holds, then the machine changes its current state z to z', overwrites the read symbol x by x' and moves the head in direction $r \in \{R, L, N\}$, where R and L stand for a move to right and left neighbouring cell, respectively, and N stands for no move. Formally, we get the following definition.

Definition 3.3 *Let $\mathcal{M}$ be a Turing machine as in Definition 3.1. Let $K_1 = (w_1, z, w_2)$ und $K_2 = (v_1, z', v_2)$ be two configurations of $\mathcal{M}$. We say that $\mathcal{M}$ transforms K_1 in K_2 (written as $K_1 \models K_2$), if one of the following conditions is satisfied:*

$$v_1 = w_1, w_2 = xu, v_2 = x'u, \delta(z, x) = (z', x', N)$$

or

$$w_1 = v, v_1 = vx', w_2 = xu, v_2 = u, \delta(z, x) = (z', x', R)$$

or

$$w_1 = vy, v_1 = v, w_2 = xu, v_2 = yx'u, \delta(z, x) = (z', x', L)$$

for some $x, x', y \in X \cup \{*\}$ und $u, v \in (X \cup \{*\})^*$.

The reflexive and transitive closure of $\models$ is denoted by $\models^*$.

We note that the machine cannot perform a working step if it is in a halting configuration because δ is not defined for halting states.

Definition 3.4 *Let $\mathcal{M} = (X, Z, z_0, Q, F, \delta, F)$ be a Turing machine as in Definition 3.1. The* language $T(\mathcal{M})$ accepted by $\mathcal{M}$ *is defined by*

$$T(\mathcal{M}) = \{w : w \in X^*,\ (\lambda, z_0, w) \models^* (v_1, q, v_2) \text{ für ein } q \in F\}.$$

According to Definition 3.4 the language $T(\mathcal{M})$ consists of all words w such that $\mathcal{M}$ starting in the initial configuration (λ, z_0, w) (i. e., the word w is written on the tape) finally reaches a halting configuration with an accepting state. We say that the words of $T(\mathcal{M})$ are accepted by $\mathcal{M}$. If a word w is not accepted, then exactly one of the following situations occurs: the machine stops in an rejecting state or it does not halt (i. e., it does not enter a halting state and works infinitely in time).

We now consider some examples. Thereby we shall present the function δ by a table where we write the value $\delta(z, x)$ in the meet of the column associated with $z \in Z \setminus Q$ and the row associated with the symbol $x \in X \cup \{*\}$.

Example 3.5 Let

$$\mathcal{M}_1 = (\{a,b\},\ \{z_0,q,q',z_a,z_a',z_b,z_b'\},\ z_0,\ \{q,q'\},\ \{q\},\delta)$$

be a Turing machine where δ is given by

δ	z_0	z_a	z_b	z_a'	z_b'
$*$	$(q',*,N)$	$(z_a',*,N)$	(z_b',b,N)	$(z_a',*,N)$	$(z_b',*,N)$
a	(z_a,a,R)	(z_a,a,R)	(z_b,a,R)	$(q,*,N)$	$(q',*,N)$
b	(z_b,b,R)	(z_a,b,R)	(z_b,b,R)	$(q',*,N)$	$(q,*,N)$.

Let *abba* be written on the tape. Then we get the following sequence of configurations.

$$\begin{aligned}(\lambda,z_0,abba) &\models (a,z_a,bba) \models (ab,z_a,ba) \models (abb,z_a,a) = (abb,z_a,a*)\\ &\models (abba,z_a,*) \models (abb,z_a',a) \models (abb,q,*).\end{aligned}$$

Consequently, $abba \in T(\mathcal{M}_1)$.

If we start with *baba* on the tape, we obtain

$$\begin{aligned}(\lambda,z_0,baba) &\models (b,z_b,aba) \models (ba,z_b,ba) \models (bab,z_b,a) \models (baba,z_b,*)\\ &\models (bab,z_b',a) \models (bab,q',*)\end{aligned}$$

and thus $baba \notin T(\mathcal{M})$.

In general, the Turing machine $\mathcal{M}_1$ starts in state z_0 and reads the letter $x \in X$. Then it remembers the read letter in the state z_x. Now it moves the read/write head to the right until it scans the first letter $*$ preserving the state z_x and therefore remembering the first letter. Now it moves the head to the left and scans the last letter in state z_x'. If the last letter is x, then it enters state q and accepts. If the last letter is different from x, then q' is the new state and the input word is rejected. If the tape contains the empty word (i. e., all cells are filled with $*$), then the rejecting state q' is obtained. Therefore

$$T(\mathcal{M}_1) = \{x_1x_2\dots x_n \mid n \geq 1,\ x_i \in \{a,b\} \text{ for } 1 \leq i \leq n,\ x_1 = x_n\}.$$

Example 3.6 We consider the Turing machine

$$\mathcal{M}_2 = (\{a,b\},\{z_0,z_1,q\},z_0,\{q\},\{q\},\delta)$$

with δ given by

δ	z_0	z_1
$*$	$(z_0,*,N)$	$(q,*,N)$
a	(z_1,a,R)	(z_0,a,R)
b	(z_1,b,R)	(z_0,b,R).

For *abb* und *abba*, we get

$$(\lambda,z_0,abb) \models (a,z_1,bb) \models (ab,z_0,b) \models (abb,z_1,*) \models (abb,q,*)$$

and

$$\begin{aligned}(\lambda,z_0,abba) &\models (a,z_1,bba) \models (ab,z_0,ba) \models (abb,z_1,b)\\ &\models (abba,z_0,*) \models (abba,z_0,*) \models (abba,z_0,*) \models \dots\end{aligned}$$

(i. e., the configuration $(abba, z_0, *)$ is not changed by $\mathcal{M}_2$ and therefore $\mathcal{M}_2$ does not halt). Therefore

$$aab \in T(\mathcal{M}_2) \quad \text{and} \quad abba \notin T(\mathcal{M}_2).$$

It is easy to see that $\mathcal{M}_2$ moves its head from left to right, is in state z_0 or z_2 if it has read a word of even or odd length, respectively. If $\mathcal{M}_2$ has read the complete word it enters the accepting q from z_1 and does not change the configuration if it is in state z_0. Hence

$$T(\mathcal{M}_2) = \{w \mid w \in \{a,b\}^*,\ |w| \text{ is odd}\}.$$

Example 3.7 We construct a Turing machine which accepts the non-context-free language $\{a^nb^nc^n \mid n \geq 1\}$. The intuitive idea is as follows: First the machine moves from left to right over the word and checks whether the input word has the form $a^nb^mc^r$ with $n \geq 1,\ m \geq 1$ and $r \geq 1$. We use the states z_a, z_b and z_c for this procedure; scanning the first a we enter z_a which is not changed as long as we read as on the tape, then we change to z_b if we read a b and so on. Then it checks that the number of as and bs and cs is equal to each other. For this it goes back to the beginning of the word (by state z_1) and cancels one a, one b and one c if it moves to the end, again, using states z_2 (no cancellation is done), z'_a (one a is cancelled), z'_b and z'_c. This is repeated as long as a distinction in the number of occurrences of letters is noticed or all letters are cancelled. There are some problems in the procedure. If the cancellation is done by writing a blank symbol $*$ on the tape, we have no knowledge anymore where the first and the last letter occurs since the letter $*$ also occurs inside the word. Hence the cancellation will be done by a writing of a new symbol d on the tape. Formally we obtain the following Turing machine

$$\mathcal{M}_3 = \{a,b,c,d\}, \{z_0, z_1, z_2, z_a, z_b, z_c, z'_a, z'_b, z'_c, q, q'\}, z_0, \{q,q'\}, \{q\}, \delta)$$

with accepting state q and rejecting state q' and the transition function δ given by the tables

	z_0	z_1	z_2	z_a	z_b	z_c
$*$	$(q', *, N)$	$(z_2, *, R)$	$(q, *, N)$	$(q', *, N)$	$(q', *, N)$	$(z_1, *, L)$
a	(z_a, a, R)	(z_1, a, L)	(z'_a, d, R)	(z_a, a, R)	$(q', *, N)$	$(q', *, N)$
b	$(q', *, N)$	(z_1, b, L)	$(q', *, N)$	(z_b, b, R)	(z_b, b, R)	$(q', *, N)$
c	$(q', *, N)$	(z_1, c, L)	$(q', *, N)$	$(q', *, N)$	(z_c, c, R)	(z_c, c, R)
d	$(q', *, N)$	(z_1, d, L)	$(q', *, N)$	$(q', *, N)$	$(q', *, N)$	$(q', *, N)$

and

	z'_a	z'_b	z'_c
$*$	$(q', *, N)$	$(q', *, N)$	$(z_1, *, L)$
a	(z'_a, a, R)		
b	(z'_b, d, R)	(z'_b, b, R)	
c	$(q', *, N)$	(z'_c, d, R)	(z'_c, c, R)
d	(z'_a, d, R)	(z'_b, d, R)	

We did not fill the second table completely, some entrances are empty. The reader can easily verify that the corresponding situation (e. g. scanning a in state z'_c) cannot occur and therefore any entrance can be done without effecting the work of the Turing machine.

From Example 3.6 we can see that it is not necessary to have rejecting states, and in all examples we have only used one accepting state. We now prove that both properties can be assumed without loss of generality.

Lemma 3.8 *For any Turing machine $\mathcal{M}$, there is a Turing machine $\mathcal{M}'$ with only one halting state which is accepting (and thus without rejecting states) such that $T(\mathcal{M}) = T(\mathcal{M}')$.*

Proof. Let $\mathcal{M} = (X, Z, z_0, Q, F, \delta)$ be a Turing machine. We construct from $\mathcal{M}$ the Turing machine $\mathcal{M}' = (X, Z', z_0, \{q'\}, \{q'\}; \delta')$ where

$$\begin{aligned} Z' &= Z \cup \{q'\} \text{ with } q' \notin Z, \\ \delta'(z, x) &= \delta(z, x) \text{ for } z \in Z \setminus Q,\ x \in X \cup \{*\}, \\ \delta'(z, x) &= (z, x, N) \text{ for } z \in Q \setminus F,\ x \in X \cup \{*\}, \\ \delta'(z, x) &= (q, x, N) \text{ für } z \in F,\ x \in X \cup \{*\}, . \end{aligned}$$

According to these settings
- the work of $\mathcal{M}'$ and $\mathcal{M}$ coincide as long as no halting state of Q is reached,
- $\mathcal{M}'$ enters an infinite loop, if it reaches an rejecting state of Q,
- if $\mathcal{M}'$ reaches an accepting state, it performs a further step which only changes the state to q',

Since rejection is replaced by non-halting, it easily follows that $T(\mathcal{M}') = T(\mathcal{M})$. □

By Lemma 3.8, the question arises why we have introduced in Definition 3.1 a possible distinction between halting and accepting states. The proof of Lemma 3.8 is based on a replacement of rejecting states by a non-halting situation. This idea cannot be used if we consider other (special) type of Turing machines where we require a halting on any input. It such cases we need rejecting states (because otherwise all input would be accepted).

If we require halting on any input, then we come to the notion of a recursive set.

Definition 3.9 *A language $L \subseteq X^*$ is called* recursive*or* decidable*, if there is a Turing machine $\mathcal{M} = (X, Z, z_0, Q, F, \delta)$ which accepts L and halts on any input word of X^*.*

We now present a characterization of recursive languages.

Theorem 3.10 *A language $L \subseteq X^*$ is recursive if and only if L as well as $X^* \setminus L$ are accepted by Turing machines.*

Proof. Let first L be a recursive language. Then there is a Turing machine $\mathcal{M} = (X, Z, z_0, Q, F, \delta)$, which accepts L and halts on any input. Obviously, the Turing machine $\mathcal{M}' = (X, Z, z_0, Q, Q \setminus F, \delta)$ accepts if and only if $\mathcal{M}$ rejects. Thus $T(\mathcal{M}') = X^* \setminus L$.

Let L and $X^* \setminus L$ be accepted by the Turing machines N and N', respectively. Without loss of generality we assume that N and N' are in the normal form given in Lemma 3.8. We construct the Turing machine N'' which works as follows. First N'' writes a second copy of the input word on the tape (both copies are separated by at least one blank symbol). Then N'' simulates N' on the input word and N' on the second copy (in order to separate both words it is possible that they have to e shifted on the tape). The single steps during the simulations are done alternately, i. e. a step simulating N is followed by one step simulating N'. It stops if a halting state of N or N' is obtained. The input word is accepted if and only if a halting state of N is reached. Because $w \in X$ or $w \in X^* \setminus L$ holds, N'' halts on any input word. Moreover, it accepts exactly the words which are accepted by N. Therefore $T(N'') = T(N) = L$. Hence L is recursive. $\square$

We now show that the recursive language are a properly special case of languages which are accepted by Turing machines. This means that the requirement to halt on any input leads to a proper decrease of the power of the machines.

Theorem 3.11 *There is a language which can be accepted by a Turing machine and is not recursive.*

Proof. Intuitive, we show that the set of all pairs $(\mathcal{M}, w)$ where $\mathcal{M}$ is a Turing machine, w is an input word for $\mathcal{M}$ and $\mathcal{M}$ halts on w is not recursive. This means that given a Turing machine $\mathcal{M}$ and an input w for $\mathcal{M}$, we cannot decide whether or not $\mathcal{M}$ halts on input w. However, this requires that any pair $(\mathcal{M}, w)$ has to be an input for a fixed Turing machine which decides the problem. Thus we have to encode Turing machines $\mathcal{M}$ as a word over a fixed alphabet.

Let $\mathcal{M} = (X, Z, z_0, Q, F, \delta)$ be a Turing machine. If we are only interested in acceptance, then we can assume without loss of generality that $Q = F$. Let

$$X = \{x_1, x_2, \ldots, x_n\},\ \ Z = \{z_0, z_1, \ldots, z_m\},\ \text{ and }\ Q = F = \{z_{k+1}, z_{k+2}, \ldots, z_m\},$$

for some $n \geq 1$ and $0 \leq k+1 \leq l \leq m$. Then $Z \setminus Q = \{z_0, z_1, \ldots, z_k\}$. We set $x_0 = *$. Moreover, instead of $\delta(z, x) = (z', x', r)$ we can use the quintuple (z, x, z', x', r) (i. e., we consider the function δ as a relation). Then δ can be described completely as the set of all these quintuples $\delta_{i,j} = (z_i, x_j, z'_{ij}, x'_{ij}, r_{ij})$ with $0 \leq i \leq k$ and $0 \leq j \leq n$. Altogether we can describe $\mathcal{M}$ as the word

$$\{x_0, x_1, \ldots, x_n\}, \{z_0, \ldots, z_m\}, \{z_{k+1}, \ldots, z_m\}, \delta_{00}, \ldots, \delta_{0,n}, \delta_{10}, \ldots, \delta_{m,n}$$

over the alphabet $\{x_0, x_1, \ldots, x_n, z_0, z_1, \ldots, z_m, \{, \}, (,), ,\}$ (the last given symbol of the alphabet is the comma). For the Turing machine of Example 3.6, we get

$$\begin{aligned} &\{*, a, b\}, \{z_0, z_1, q\}, \{q\}, (z_0, *, z_0, *, N), (z_0, a, z_1, a, R), (z_0, b, z_1, b, R), \\ &(z_1, *, q, *, N), (z_1, a, z_0, a, R), (z_1, b, z_0, b, R). \end{aligned} \tag{3.1}$$

Because the alphabet is different for all Turing machines we use the following encoding over $\{0, 1\}$:

$$\begin{aligned} &x_j \to 01^{j+1}0 \quad \text{for } 0 \leq j \leq n, \\ &z_i \to 01^{i+1}0^2 \quad \text{for } 0 \leq i \leq k, \\ &R \to 010^3, \quad L \to 01^20^3, \quad N \to 01^30^3, \\ &(\to 010^4, \quad) \to 01^20^4, quad \{ \to 01^30^4, \quad \} \to 01^40^4 \\ &, \to 010^5. \end{aligned}$$

By this encoding we obtain a description of $\mathcal{M}$ as a word over $\{0,1\}$. From (3.1) with $a = x_1$, $b = x_2$, and $q = z_2$, we yield

$$
\begin{array}{l}
01^30^4\ 010\ 010^5\ 01^20\ 010^5\ 01^30\ 01^40^4\ 010^5\ 01^30^4\ 010^2\ 010^5\\
01^20^2\ 010^5\ 01^30^2\ 01^40^4\ 010^5\ 01^30^4\ 01^30^2\ 01^40^4\ 010^5\\
010^4\ 010^2\ 010^5\ 010\ 010^5\ 010^2\ 010^5\ 010\ 010^5\ 01^30^3\ 01^20^4\ 010^5\\
010^4\ 010^2\ 010^5\ 01^20\ 010^5\ 01^20^2\ 010^5\ 01^20\ 010^5\ 010^3\ 01^20^4\ 010^5\\
010^4\ 010^2\ 010^5\ 01^30\ 010^5\ 01^20^2\ 010^5\ 01^30\ 010^5\ 010^3\ 01^20^4\ 010^5\\
010^4\ 01^20^2\ 010^5\ 010\ 010^5\ 01^30^2\ 010^5\ 010\ 010^5\ 01^30^3\ 01^20^4\ 010^5\\
010^4\ 01^20^2\ 010^5\ 01^20\ 010^5\ 010^2\ 010^5\ 01^20\ 010^5\ 010^3\ 01^20^4\ 010^5\\
010^4\ 01^20^2\ 010^5\ 01^30\ 010^5\ 010^2\ 010^5\ 01^30\ 010^5\ 010^3\ 01^20^4
\end{array}
$$

(where the first two rows describe the sets and each of the remaining six lines a quintuple of the transition function).

Given a Turing machine $\mathcal{M}$, we denote the associated encoding as a word over $\{0,1\}$ by $w_{\mathcal{M}}$. Clearly, given $\mathcal{M}$, then w_M uniquely determined. Conversely, if we have a word over $\{0,1\}$ which describes a Turing machine, then we can uniquely reconstruct from w a Turing machine $\mathcal{M}$ such that $w = w_{\mathcal{M}}$.

By $\mathcal{S}$ we denote the set of all Turing machines $\mathcal{M} = (X, Z, z_0, Q, Q\delta)$ with the input set $X = \{0,1\}$, an arbitrary set $Z = \{z_0, z_1, \dots, z_m\}$, $m \geq 1$, of states and a single halting state z_m which is an accepting one and some transition function δ (by Lemma 3.8 we can restrict to these machines without loss of generality). If $\mathcal{M} \in \mathcal{S}$, then its encoding $w_{\mathcal{M}}$ as an word over the input set of $\mathcal{M}$ can be used as an input of $\mathcal{M}$.

We now consider the set

$$U = \{w \mid w \in \{0,1\}^*,\ w = w_{\mathcal{M}} \text{ for some Turing machine } \mathcal{M} \in \mathcal{S}, w = w_{\mathcal{M}} \in T(\mathcal{M})\}.$$

We prove that U is not a recursive set by contradiction. Thus let us assume that U is recursive. Then the complement

$$\begin{aligned} C(U) \;=\; & \{w \mid w \in \{0,1\}^*,\ w \text{ is not an encoding of a Turing machine of } \mathcal{S}, \textit{or} \\ & \quad (w = w_{\mathcal{M}} \text{ for some Turing machine } \mathcal{M} \in \mathcal{S} \text{ and } w = w_{\mathcal{M}} \notin T(\mathcal{M}))\} \end{aligned}$$

of U is acceptable by some Turing machine $\mathcal{N}$, i. e.,

$$T(\mathcal{N}) = C(U). \tag{3.2}$$

Obviously, $\mathcal{N} \in \mathcal{S}$.

Let us assume that $w_{\mathcal{N}} \in T(\mathcal{N})$. Since w_N is the encoding of a Turing machine and $w_{\mathcal{N}} \in T(\mathcal{N})$, $w_{\mathcal{N}}$ is not in $C(U)$. By (3.2), this gives $w_{\mathcal{N}} \notin T(\mathcal{N})$ in contrast to our assumption.

Let us assume that $w_{\mathcal{N}} \notin T(\mathcal{N})$. Then by the definition of $C(U)$, we have $w_{\mathcal{N}} \in C(U)$. Thus $w_{\mathcal{N}} \in T(\mathcal{N})$ by (3.2). Again, we have a contradiction to our assumption.

Since we get a contradiction in all possible cases, our assumption that U is recursive has to be false.

It remains to prove that U is accepted by a Turing machine $\mathcal{K}$. We only explain the behaviour of $\mathcal{K}$, a detailed description of K is left to the reader. The machine $\mathcal{K}$ scans the input and checks whether or not it is an encoding of a Turing machine (first it looks whether or not the word has the right structure, i. e., whether a set of three input symbols $*, 0, 1$, a state set of some size and a singleton set of the halting state is followed by a sequence of $\delta_{i,j}$ which are all in encoded form and separated by encoding of comma; then it checks whether, for every pair of a state z from $Z \setminus Q$ and an input symbol x, there is a corresponding quintuple (z, x, z', x', r) in encoded form). In the non-affirmative case, K rejects the input. In the affirmative case, $w = w_{\mathcal{T}}$ holds for some Turing machine $\mathcal{T}$. Now K copies w on the tape and simulates the work of T on the second copies (where the first copy is used to look which transformation has to be done on the second copy). K halts if this simulation comes to the halting state of $\mathcal{T}$. Thus $\mathcal{K}$ halts if and only if $\mathcal{T}$ halts if and only if $w = w_{\mathcal{T}}$ is accepted by $\mathcal{T}$. □

We now present the first relation between Turing machines and grammars considered in the preceding chapter.

Lemma 3.12 *For any Turing machine $\mathcal{M}$, there is a phrase structure grammar G such that $L(G) = T(\mathcal{M})$.*

Proof. Let the Turing machine $\mathcal{M} = (X, Z, z_0, Q, F, \delta)$ be given. Then w belongs to the set $T(\mathcal{M})$ if

$$(\lambda, z_0, w) \models^* (v_1, q, v_2) \tag{3.3}$$

for some accepting state $q \in F$ and some words v_1 and v_2 over $X \cup \{*\}$ (the reader should confirm that v_1 and v_2 can contain some occurrences of $*$). The intuitive idea for the grammar which generates $T(\mathcal{M})$ is as follows: first, in some steps, we generate $v_1 q z_2$ (a word representing the halting configuration) from which we derive $z_0 w$ in some steps, where each step is reverse to the transformation of a configuration into the successor configuration, and finally from $z_0 w$ we generate w and check that $w \in X^*$. However, this idea needs some additional things to be realized. For instance, to derive w from $z_0 w$ we have to know that z_0 is the first letter of the word. Since a rule $\alpha \to \beta$ can be applied at any occurrence of α in a word and not only to α in the beginning of the word. Thus we use additional symbols § and # which are placed in the start and end position of a word (or in other words § and # are left and right markers). Furthermore, we have to recognize the unique state z in the word $u_1 z u_2$ representing the configuration (u_1, z, u_2). Therefore we require that $X \cap Z = \emptyset$, $*, \S, \# \notin Z$ which can be done without loss of generality (if necessary we can rename the states).

We now give the phrase structure grammar $G = (N, T, P, S)$. We set

$$\begin{aligned} N &= Z \cup \{\S, \#, *, S, S', A, A', A'', A'''\}, \\ T &= X \end{aligned}$$

and define P as the set of all rules of the following forms:

$$\begin{array}{ll} (i) & S \longrightarrow \S S'\#, \\ & S' \longrightarrow xS', S' \longrightarrow S'x \quad \text{for } x \in X \cup \{*\}, \\ & S' \longrightarrow q \quad \text{für } q \in F \end{array}$$

(by these rules we realize a derivation $S \Longrightarrow^* \S v_1 q v_2 \#$ which gives a description of a halting configuration),

$$\begin{array}{lll}(ii) & z'ab' \longrightarrow azb & \text{for } z,z' \in Z',\ a,b,b' \in X \cup \{*\},\ \delta'(z,b) = (z',b',L),\\ & z'b' \longrightarrow zb & \text{for } z,z' \in Z',\ b,b' \in X \cup \{*\},\ \delta'(z,b) = (z',b',N),\\ & b'z' \longrightarrow zb & \text{for } z,z' \in Z,\ b,b' \in X \cup \{*\},\ \delta'(z,b) = (z',b',R)\end{array}$$

(by these rules we simulation the transformations of configurations in the opposite direction; for instance we have $\S v_1 z'ab'v_2\# \Longrightarrow \S v_1 azbv_2\#$ if $(v_1a, z, bv_2) \models (v_1, z', ab'v_2)$ by $\delta(z,b) = (z',b',L)$); using the third type of rules from (ii), we can get a subword $z\#$ which is impossible in a configuration, however, then – as we can see by the construction of all rules, then no rule is applicable and the derivation is blocked; but if we produce additional symbol $*$ before $\#$ in the first phase of the derivation by rules from (i), the derivation can be continued),

$$\begin{array}{ll}(iii) & z_0* \longrightarrow A,\\ & *A \longrightarrow A,\\ & \S A \longrightarrow A''',\\ & z_0x \longrightarrow A'' \text{ for } x \in X,\\ & *A' \longrightarrow A',\\ & \S A' \longrightarrow A'',\\ & A''x \longrightarrow xA'' \text{ for } x \in X,\\ & A''* \longrightarrow A''',\\ & A''\# \longrightarrow \lambda,\\ & A'''* \longrightarrow A''',\\ & A'''\# \longrightarrow \lambda\end{array}$$

(by these rules we realize derivations $\S *^n z_0 *^m \# \Longrightarrow \lambda$ and $\S *^n z_0 w *^m \# \Longrightarrow w$ where $w \in X^+$; therefore we obtain the word input word).

By the explanations given to the groups of rules it is easy to see that any derivation in G has the form

$$S \overset{*}{\Longrightarrow} u_1 = \S v_1 q v_2 \# \overset{*}{\Longrightarrow} u_2 = \S z_0 w \# \overset{*}{\Longrightarrow} w \qquad (3.4)$$

where any derivation step of the subderivation $u_1 \overset{*}{\Longrightarrow} u_2$ consist in a backwards simulation of a transformation of configurations.

Thus we have shown that a terminating derivation is of form (3.4) and exists if and only if (3.3) holds. Therefore $w \in L(G)$ if and only if $w \in T(\mathcal{M})$. Hence $L(G) = T(\mathcal{M})$. □

The consequence of Lemma 3.12 is that any language which can be accepted by a Turing machine is recursively enumerable.

3.1.2 Nondeterministic Turing Machines and Their Accepted Languages

The aim of this subsection is the proof that any recursively enumerable language is acceptable by a Turing machine. If we want to give a proof analogously to the proof of

Lemma 3.12 – the Turing machine simulates the derivation of the grammar backwards – then there occurs the problem that the process is not deterministic. For example the derivations $abACa \Longrightarrow ababCa$ and $aBbCa \Longrightarrow ababCa$ are valid by application of the rules $A \rightarrow ab$ and $B \rightarrow ba$, respectively. Thus the machine has – essentially – to realize $ababCa \models abACa$ and $ababCa \models aBbCa$ which is impossible since any configuration has a unique successor configuration. In order to have also two or more choices for the successor configuration (and thus to have a possibility for an analogous proof) we modify the definition of a Turing machine.

Definition 3.13 *A* nondeterministic Turing machine *is a sixtuple*

$$\mathcal{M} = (X, Z, z_0, Q, F, \tau),$$

where X, Z, z_0, Q und F are specified as in the case of a Turing machine and τ is a function

$$\tau : (Z \setminus Q) \times (X \cup \{*\}) \rightarrow 2^{Z \times (X \cup \{*\}) \times \{R,N,L\}}.$$

According to this definition $\tau(z, x)$, $z \in Z \setminus Q$, $x \in X \cup \{*\}$ is a finite set of triples (z', x', r) with $z' \in Z$, $x' \in X \cup \{*\}$, $r \in \{R, L, N\}$.

The Turing machine defined in Definition 3.1 can be considered as a special case of the nondeterministic Turing machine of Definition 3.13 where any set $\tau(z, x)$, $z \in Z \setminus Q$, $x \in X \cup \{*\}$, consist only of the element $\delta(z, x)$. Turing machines as presented in Definition 3.1 are sometimes called deterministic Turing machines

We define the configuration of a nondeterministic Turing machine as for a (deterministic) Turing machine (see Definition 3.2) and the relation $K_1 \models K_2$ for configurations K_1 and K_2 as in Definition 3.3, where we replace $\delta(z, x) = (z', x', r)$ by the requirement that $(z', x', r) \in \tau(z, x)$.

Obviously, according to these settings a configuration K_1 has as many successor configurations as the set $\tau(z, x)$ has elements.

Definition 3.14 *Let $\mathcal{M} = (X, Z, z_0, Q, F, \tau)$ be a nondeterministic Turing machine and $w \in X^*$. A halting (or accepting) computation path for w is a sequence $K_0, K_1, \ldots, K_t$, $t \geq 0$, of configurations of $\mathcal{M}$ such that*
– $K_0 = (\lambda, z_0, w)$,
– $K_0 \models K_1 \models \cdots \models K_t$, and
– K_t is halting configuration (or an accepting configuration, respectively).
The number t is called the length of the computation path.

Note that the notion of a computation path can be used for deterministic Turing machines, too. In case of deterministic Turing machines, for any word w, there is at most one computation path for w, whereas there can exist some computation paths for w in case of nondeterministic Turing machines.

We now define the language accepted by a nondeterministic Turing machine in analogy to Definition 3.4.

Definition 3.15 *Let $\mathcal{M} = (X, Z, z_0, Q, \tau, F)$ be a nondeterministic Turing machine as in Definition 3.13. The* language $T(\mathcal{M})$ accepted by $\mathcal{M}$ *is defined by*

$$T(\mathcal{M}) = \{w \mid w \in X^*, \text{ there is a computation path for } w\}.$$

According to Definition 3.15, a word w is accepted if there exists a sequence of configurations starting with the initial configuration (λ, z_0, w) which ends in an accepting configuration. However, there can also be a sequence of configurations starting with (λ, z_0, w) and halting in a rejecting configuration or there can be an infinite sequence of configurations starting with (λ, z_0, w). This means that we require the existence of one computation path for acceptance, and we do not care what is with the other existing sequences of configurations. Thus, a word v is not accepted if all existing sequences of configurations starting with (λ, z_0, v) do not end or end in a configuration with a rejecting state.

We give an example.

Example 3.16 We consider the nondeterministic Turing machine

$$\mathcal{M} = (\{a,b\}, \{z_0, z_0', z_0'', z_{0,2}z_{1,2}, z_2, z_2', z_2'', z_{0,3}z_{1,3}, z_{2,3}, z_3, z_3', z_3'', q\}, z_0, \{q\}, \tau, \{q\})$$

where

$$\begin{array}{rcl}
\tau(z_0,a) & = & \{(z_0,a,R)\}, \\
\tau(z_0,b) & = & \{(z_0,b,N)\}, \\
\tau(z_0,*) & = & \{(z_0',*,L)\}, \\
\tau(z_0',a) & = & \{(z_0',a,L)\}, \\
\tau(z_0',*) & = & \{(z_0'',*,R)\}
\end{array}$$

(the machine decides checks whether there are only as on the tape; in the non-affirmative case, it enters a loop),

$$\begin{array}{rcll}
\tau(z_0'',*) & = & \{(z_o'',*,N)\}, & \\
\tau(z_0'',x) & = & \{(z_2,x,N),(z_3,x,N)\} & \text{for } x \in \{a,b\}
\end{array}$$

(if there is no letter on the tape, the machine enters a loop, too; otherwise it chooses nondeterministically one of the two possibilities fixed by the index 2 or 3, respectively),

$$\begin{array}{rcll}
\tau(z_i,a) & = & \{(z_i',a,R)\} & \text{for } i \in \{2,3\}, \\
\tau(z_i,b) & = & \{(z_i,b,R)\} & \text{for } i \in \{2,3\}, \\
\tau(z_i',a) & = & \{(z_i'',a,R)\} & \text{for } i \in \{2,3\}, \\
\tau(z_i',b) & = & \{(z_i',b,R)\} & \text{for } i \in \{2,3\}, \\
\tau(z_i'',x) & = & \{(z_i'',x,R)\} & \text{for } x \in \{a,b\}, \\
\tau(z_i,*) & = & \{(z_i,*,N)\} & \text{for } i \in \{2,3\}, \\
\tau(z_i',*) & = & \{(q,*,N)\} & \text{for } i \in \{2,3\}, \\
\tau(z_i'',*) & = & \{(z_{0,i},*,L)\} & \text{for } i \in \{2,3\}
\end{array}$$

(the machine reads from left to right the word on the tape and checks whether it contains no a; exactly one a or at least two occurrences of a, which is done by the states z_i, z_i' und z_i''; if there is no a on the tape, the machine goes into a loop and does not halt; if there is exactly one a on the tape, it enters the accepting state; if there are at least two as, the next phase is started),

$$\begin{aligned}\tau(z_{0,2},a) &= \{(z_{1,2},b,L)\},\\ \tau(z_{1,2},a) &= \{(z_{0,2},a,L)\},\\ \tau(z_{j,2},b) &= \{(z_{i,2},b,L)\} \quad \text{for } j\in\{0,1\}\end{aligned}$$

(the machine scans the word on the tape from the left to the right and alternately it changes an a into a b or does not change the letter a; thus it reduces the number of occurrences of a to the half; the number j in the state $z_{j,2}$ gives the number of the read symbols a modulo 2),

$$\begin{aligned}\tau(z_{0,3},a) &= \{(z_{1,2},b,L)\},\\ \tau(z_{1,3},a) &= \{(z_{2,3},b,L)\},\\ \tau(z_{2,3},a) &= \{(z_{0,3},a,L)\},\\ \tau(z_{j,3},b) &= \{(z_{j,3},b,L)\} \quad \text{for } j\in\{0,1,2\}\end{aligned}$$

(analogously, the number of occurrences of a is reduced to the third and the number j in $z_{j,3}$ gives the number of the read symbols a modulo 2),

$$\begin{aligned}\tau(z_{0,i},*) &= \{(z_i,*,R)\} \quad \text{for } i\in\{2,3\},\\ \tau(z_{j,i},*) &= \{(z_{j,i},*,N)\} \quad \text{for } j\in\{1,2\}\ , i\in\{2,3\}\end{aligned}$$

(if the reduction can be done without a remainder, i. e., $z_{0,2}$ or $z_{0,3}$ is reached, we iterate the process and enter a loop in the opposite case such that the machine does not halt).

By the explanations given after the parts of the transition function, it is obvious that the iterated reductions of the occurrences of a to the half or the third finally lead to a word on the tape which contains exactly one a and the machine accepts. Thus we get the set

$$T(\mathcal{M}) = \{w : \#_a(w) = 2^n \text{ oder } \#_a(w) = 3^n \text{ for an } n \geq 0\}$$

of accepted words.

We are now in the position to prove the converse of Lemma 3.12 but using nondeterministic Turing machines instead of (deterministic) Turing machines.

Lemma 3.17 *For any phrase structure grammar G, there is a nondeterministic Turing machine $\mathcal{M}$ such that $T(\mathcal{M}) = L(G)$.*

Proof. We give no detailed proof; we only explain the essential idea of the construction.

Let the phrase structure grammar $G = (N, T, P, S)$ be given. We construct the nondeterministic Turing machine $\mathcal{M}$ with the input alphabet $N \cup T \cup \{\S\}$ and the following behaviour on an input word w.

– *Phase* 1.
  Since we want to accept only words over T, $\mathcal{M}$ checks first whether $w \in T^*$ holds. In the non-affirmative case, $\mathcal{M}$ enters a loop (and therefore it does not accept w); if $w \in T^*$, $\mathcal{M}$ enters the configuration (λ, z_1, w), which is the start configuration of the second phase.

– *Phase* 2.
  In the second phase $\mathcal{M}$ checks first whether only the letter S is on the tape. In the affirmative case, $\mathcal{M}$ halts; if the tape contents is different from S, the machine enters the configuration (λ, z_2, w), which is the start of the third phase.
– *Phase* 3.
  This phase is a simulation of a derivation according G but – as in the proof of Lemma 3.12 – we reverse the direction, i. e., we simulate the derivation

$$xuy \Longrightarrow xu'y = w$$

by the application of the rule $u \longrightarrow u'$ by some transformations resulting in

$$(\lambda, z_2, w) = (\lambda, z_2, xu'y) \models^* (\lambda, z_1, xuy).$$

For this $\mathcal{M}$ determines nondeterministically the position in the word w where $\mathcal{M}$ wants to apply the rule $p = u \longrightarrow u'$. This is done by going in the configuration (x, z_p, x'), where z_p marks the begin of the simulation. $\mathcal{M}$ checks whether u' is the prefix of the word x'. In the non-affirmative case, $\mathcal{M}$ enters a loop and does not accept. If u' is a prefix of x', say $x' = u'y$, $\mathcal{M}$ continues as follows. If $|u'| - |u| = m \geq 0$, $\mathcal{M}$ replaces the prefix u' by the word $u\S^m$, which results in $(xu\S^m, z_p', y)$. Then y is shifted m cells to the left which is accompanied by a cancellation of $\S^m$ and moves the head to the beginning of the word and enters the state z_1. If $|u| - |u'| = m' > 0$, the word y is shifted m' cells to the right and moves the head to the beginning of the prefix u', which yields $(x, z_p'', u' *^{m'} y$. Then $\mathcal{M}$ replaces $u' *^{m'}$ by u and returns the head to the begin of the word on the tape and in the state z_1. Thus we obtain the configuration (λ, z_1, xuy) from $(\lambda, z_2, xu'y)$. Obviously, we have performed the simulation of a derivation step in reversed direction.
Now we start Phase 2, again.

Thus any derivation

$$S \Longrightarrow w_1 \Longrightarrow w_2 \Longrightarrow \ldots \Longrightarrow w_{n-1} \Longrightarrow w_n = w$$

has a corresponding sequence of configurations such that

$$\begin{aligned}(\lambda, z_0, w_n) &\models^* (\lambda, z_1, w_n) \models^* (\lambda, z_2, w_n)\\ &\models^* (\lambda, z_1, w_{n-1}) \models^* (\lambda, z_2, w_{n-1})\\ &\models^* \cdots \models^* (\lambda, z_1, w_2) \models^* (\lambda, z_2, w_2)\\ &\models^* (\lambda, z_1, w_1) \models^* (\lambda, z_2, w_1)\\ &\models^* (\lambda, z_1, S) \models^* (\lambda, q, S).\end{aligned}$$

Furthermore, $\mathcal{M}$ can only reach a halting state if $\mathcal{M}$ has simulated a derivation since $\mathcal{M}$ in all other case goes into a loop. If any halting state is an accepting one, we get $T(\mathcal{M}) = L(G)$. □

Lemma 3.12 and Lemma 3.12 are converse to each other because different types of Turing machines are involved. To get a better relation one needs knowledge on the relation between the families of languages accepted by deterministic and nondeterministic Turing machines. The following lemma shows that both these language families coincide.

Lemma 3.18 *A language can be accepted by a (deterministic) Turing machine if and only if it can be accepted by a nondeterministic Turing machine.*

Proof. Obviously, since any (deterministic) Turing machine $N = (X, Z, z_0, Q, F, \delta)$ can be interpreted as a nondeterministic Turing machine $N' = (X, Z, z_0, Q, F, \tau)$ by setting $\tau(z, x) = \{\delta(z, x)\}$ and this setting does not change the accepted language, we have that any language which can be accepted by some (deterministic) Turing machine N can also be accepted by some nondeterministic Turing machine N'.

Let $\mathcal{M} = (X, Z, z_0, Q, F, \tau)$ be a n0ndeterministic Turing machine. If $\mathcal{M}$ accepts the word w, then there is a sequence of configurations with the initial configuration (λ, z_0, w) and the halting configuration (v_1, q, v_2) where $q \in F$. The idea for the construction of a deterministic Turing machine whichs also accepts $T(\mathcal{M})$ is construct all possible sequences of configurations for $\mathcal{M}$ and to check which of them are accepting. This idea has to be modified a little bit since there are are non-halting computations by $\mathcal{M}$. Thus we first consider all configurations which can be obtained by one transition, then all which can be got by two transitions etc. Since any accepting sequence has finite length we get all of them in this way.

Let

$$D = \{(z, x, z', x', r) \mid z \in Z \backslash Q,\ z' \in Z,\ x, x' \in X \cup \{*\},\ r \in \{R, N, L\},\ (z', x', r) \in \tau(z, x)\}$$

The set D can be understood as the set of all instructions which it can apply in a transformation of configurations. We introduce an order on the set D (which can be chosen arbitrarily). This order can be extended to a lexicographic order of the set D^* of all finite sequences over D. For an element f of D^*, we denote the successor of f in the lexicographic order by $nseq(f)$. Let $K = (w_1, z, w_2)$ and $d = (z_1, x_1, z_2, x_2, r)$ be an element of D, then we say that d can be applied to K if and only if $z = z_1$ and x_1 is the first letter of w_2. Thus any sequence of configurations such that $K_1 \models K_2 \models \cdots \models K_n$ is accompanied by a sequence $d_1 d_2 \ldots d_{n-1}$ where, for $1 \leq i < n$, d_i is applied to K_i to obtain K_{i+1}.

We now give an informal description of a deterministic Turing machine $\mathcal{M}'$ which accepts $T(\mathcal{M})$ by describing the behaviour of $\mathcal{M}'$. We start with w on the tape and write two times the special separator \$ in the cells following those where w is written. This yields $w\$\$$ on the tape and represents w and the empty word between the two separators on the tape. Now let us have the tape contents $w\$f\$$ where f is a sequence over D. Then $\mathcal{M}$ performs as given in the following table: $\mathcal{M}'$ writes additionally a copy of w and two copies of $f' = nseq(f)$ on the tape yielding $w\$f\$w\$f'\$f'\$$. Then it deletes $f\$$ which gives $w\$w\$f'\$f'\$$. Now it performs in succession the instructions of f' on the first copy of w as it is done by $\mathcal{M}$ and cancels the already used instructions of f' obtaining $w'\$w\$\$f'\$$. If the obtained state of $\mathcal{M}$ is accepted, then $\mathcal{M}'$ also accepts. If the obtained state is not a halting one or an rejecting state, then we cancel $w'\$$ and the \$ before f' which results in $w\$f'\$$ and we repeat the process. Note that it is not sure that any instruction of f' can be applied to the current word w'' which we get by the already used instructions. Then we cancel $w''\$$ and an \$ and obtain $w\$f'\$$, too, from which we start the process, again.

It is easy to see that these actions can be performed by a deterministic Turing machine. Obviously, $\mathcal{M}'$ checks for a word w all possible sequences of instructions and therefore all sequences of configurations until an accepting sequence is obtained or it does not halt. Thus $T(\mathcal{M}') = T(\mathcal{M})$. □

Summarizing we get the main result of this subsection which relates Turing machines and phrase structure grammars to each other.

Theorem 3.19 *For a language L, the following three statements are equivalent:*
i) *There is a phrase structure grammar G such that $L = L(G)$.*
ii) *There is a Turing machine $\mathcal{M}$ such that $L = T(\mathcal{M})$.*
iii) *There is a nondeterministic Turing machine $\mathcal{M}$ such that $L = T(\mathcal{M})$.* □

Corollary 3.20 *The family of recursive languages is properly contained in the family $\mathcal{L}(RE)$.*

Proof. By the Definition 3.9 of recursive languages and Theorem 3.19, any recursive language is in $\mathcal{L}(RE)$. Thus we have that the family of recursive languages is contained in $\mathcal{L}(RE)$. By Theorem 3.11, the inclusion is proper. □

Let us now consider the case of monotone grammars (or from the point of languages we can also consider a context-sensitive language). Then the proof of Lemma 3.17 gets the following "simplification": We start with w on the tape and in all subsequent steps we do not increase the length of the word on the tape since the derivations steps of a monotone grammar do not decrease the length of the sentential forms and the simulation takes the opposite direction. Therefore the cells which are scanned by the head during the work of the Turing machine are only those where letters of w are written in and – eventually – the cell before and the cell after those containing w (which is necessary, if we have to look for the beginning and ending of w). Thus $|w| + 2$ is a bound for the number of cells scanned by the machine.

Definition 3.21 *A nondeterministic Turing machine is called a* linearly bounded automaton *if there is a linear function $f : \mathbb{N} \to \mathbb{N}$ such that, for any input w, the Turing machine scans at most $f(|w|)$ cells during its work on w.*

A linearly bounded automaton is said to be strong*, if its linear function f has the form $f(x) = x + 2$.*

The above considerations can now be reformulated as follows: Any context-sensitive language can be accepted by a strong linearly bounded automaton. In order to prove the converse statement we need the following notion and statement.

Let $G = (N, T, P, S)$ be a phrase structure grammar. For a derivation

$$D : S \Longrightarrow w_1 \Longrightarrow w_2 \Longrightarrow \dots \Longrightarrow w_r = w$$

of $w \in T^*$ in G, we define the *workspace of w by D* by

$$Ws_G(w, d) = max\{|w_i| \mid 1 \leq i \leq r\}$$

and the workspace of w by

$$Ws_G(w) = \min\{Ws_G(w, D) \mid D \text{ is a derivation of } w \text{ in } G\}\,.$$

Theorem 3.22 *If $G = (N, T, P, S)$ is a phrase structure grammar and k is a positive integer such that $Ws_G(w) \leq k|w|$ for any $w \in L(G)$, then $L(G)$ is a context-sensitive language.* □

We omit the very technical and long proof of the Workspace Theorem; we refer to [28] for a proof.

The Workspace Theorem can be interpreted as follows. Monotone and context-sensitive grammars do not allow a shortage of the length of the sentential forms. However, if the shortage is not to large (more precisely, linearly bounded), then a phrase structure grammar with non-monotone rules also generates only context-sensitive languages.

We are now in the position to present a characterization of context-sensitive languages in terms of automata/machines.

Theorem 3.23 *For a language L, the following three statements are equivalent:*

i) *The language L is context-sensitive (i. e., L is generated by a context-sensitive or monotone grammar).*
ii) *The language L is accepted by a strong linearly bounded automaton.*
iii) *The language L is accepted by a linearly bounded automaton.*

Proof. i) $\rightarrow$ ii) follows from the considerations given above.

ii) $\rightarrow$ iii) holds by definition.

iii) $\rightarrow$ i) Let $\mathcal{M}$ a linearly bounded automaton be given. Let f be its associated function. We look on the proof of Lemma 3.12. First we modify it that it also works for nondeterministic Turing machines (the easy modifications are left to the reader). Now we can apply it to $\mathcal{M}$. Then we note that it is sufficient to generate $\S *^n v_1qv_2 *^m \#$ in the first of the derivation such that $2+n+m+|v_1zv_2| = f(|w|)+2$ which ensures that all derivation steps can be performed and the length of all sentential forms is bounded by $f(|w|)+2$. Now by the Workspace Theorem 3.22 we get that the constructed phrase structure grammar generates a context-sensitive language. □

In the case of Turing machines we have considered deterministic and nondeterministic variants. For (strong) linearly bounded automata, we have only introduced a nondeterministic one. It is an open question whether or not any context-sensitive language can be generated by a deterministic (strong) linearly bounded automaton. This problem is studied since more than 40 years, but no answer is known hitherto.

3.1.3 A Short Introduction to Computability and Complexity

In Subsection 3.1.1, we have introduced the notion of a Turing machine as a device accepting (recursively enumerable) languages. However, Turing machines can also be used to compute functions.

Definition 3.24 *Let $\mathcal{M} = (X, Z, z_0, Q, F, \delta)$ be a Turing machine. The function $f_{\mathcal{M}} : X^* \rightarrow (X \cup \{*\})^*$ induced by $\mathcal{M}$ is defined as follows: $f_m(w) = v$ holds if and only if there is a sequence of configurations $K_0, K_1, \dots, K_t$, $t \geq 0$, such that*

– *$K_0 = (\lambda, z_0, w)$,*
– *$K_0 \models K_1 \models K_2 \models \dots \models K_t$,*
– *$K_t = (v_1, q, v_2)$ for some $q \in Q$,*
– *$v_1v_2 = *^r v *^s$ for some $v \in (X \cup \{*\})^*$, $r \geq 0$, $s \geq 0$ and $v = \lambda$ or v starts and ends with a letter of X.*

Intuitively, if we initially write w on the tape and let the Turing machine work until it reaches a halting state, then the result $f_{\mathcal{M}}(w)$ is the word written on the tape which starts and ends with a letter of X, i. e., we ignore the infinitely many symbols $*$ to the left or to the right. Note that there is no value $f_{\mathcal{M}}(w)$ defined, if $\mathcal{M}$ enters no halting states working on the input word w.

If we consider the Turing machines $\mathcal{M}_1$ and $\mathcal{M}_2$ from the Examples 3.5 and 3.6, we get

$$f_{\mathcal{M}_1}(aw) = wa \text{ for } w \in \{a,b\}^*$$

(note that $f_{\mathcal{M}_1}(\lambda)$ is not defined), and

$$f_{\mathcal{M}_2}(w) = w \text{ for } w \in \{a,b\}^*,\ |w| \text{ is odd}$$

(for words of even length $f_{\mathcal{M}_2}$ is undefined).

Obviously, considering a Turing machine as a device computing functions, we can omit the set of accepting states in the tuple specifying the Turing machine, since we are only interested in the result on the tape if a halting state is reached, independent from acceptance or rejection by the halting state.

Definition 3.25 *Let X_1 and X_2 be two alphabets. A function $f : X_1^* \to X_2*$ is called* Turing computable *if there is a computing Turing machine $\mathcal{M} = (X, Z, z_0, Q, \delta)$ such that $X_1 \cup X_2 \subseteq X$ and*

$$f_{\mathcal{M}}(w) = \begin{cases} f(w) & \textit{for } w \in X_1^* \\ \textit{undefined} & \textit{otherwise} \end{cases} .$$

Using the notion of a Turing computable function, we get some further characterizations of the sets of recursively enumerable languages and recursive languages.

Theorem 3.26 *A language L is recursively enumerable if and only if L is the domain of a Turing computable function.*

Proof. Let L be a recursively enumerable language. By Theorem 3.19, there is a (deterministic) Turing machine M such that $T(\mathcal{M}) = L$. By Lemma 3.8, we can assume that $\mathcal{M}$ has exactly one halting state which is accepting. Thus $w \in L = T(\mathcal{M})$ holds if and only if $f_{\mathcal{M}}(w)$ is defined. Therefore $T(\mathcal{M})$ is the domain of $f_{\mathcal{M}}$.

Conversely, let L be the domain of the Turing computable function f. Then there is a computing Turing machine $\mathcal{M} = (X, Z, z_0, Q, \delta)$ such that $\mathcal{M}$ halts on w if and only if $f(w)$ is defined. Therefore, the (accepting) Turing machine $(X, Z, z_0, Q, Q, \delta)$ accepts L. Hence L is recursively enumerable by Theorem 3.19. □

Theorem 3.27 *For any non-empty recursively enumerable language $L \subseteq X^*$, there is a Turing machine $\mathcal{M}$ such that $f_{\mathcal{M}} : \{0, 1, 2, \ldots, 9\}^* \to X^*$, the domain of $f_{\mathcal{M}}$ is the set of decimal representations of elements of $\mathbb{N}_0$ (without leading zeros), and the range of $f_{\mathcal{M}}$ is L.*

Proof. We only give the idea of the work of $\mathcal{M}$ and leave a detailed description to the reader.

Let $L \subseteq X^*$ be a recursively enumerable language. By Theorem 3.19, there is a deterministic Turing machine $\mathcal{M}'$ which accepts L. Moreover, let u be a fixed element of L (which exists since $L \neq \emptyset$.

First we note that we can define an order on $\mathbb{N}_0 \times \mathbb{N}_0$ as follows: $(a_1, b_1) \prec (a_2, b_2)$ if and only if $a_1 + b_1 < a_2 + b_2$ or $a_1 + b_1 = a_2 + b_2$ and $a_1 < a_2$. This means that we first order according to the sum of the components and if the sums are equal we order according to the first component. Obviously,

$$(0,0) \prec (0,1) \prec (1,0) \prec (0,2) \prec (1,1) \prec (2,0) \prec (0,3) \prec (1,2) \prec \ldots.$$

Moreover, it is easy to see that there is a Turing machine $\mathcal{M}_1$ such that, for a given number $n \in mathbbN_0$, $\mathcal{M}_1$ computes the n-th element $(a,b) \in \mathbb{N}_0 \times \mathbb{N}_0$ according to the order $\prec$.

Furthermore, there is a deterministic Turing machine $\mathcal{M}_2$ such that, for a given number $a \in mathbbN_0$, $\mathcal{M}_2$ computes the a-th element w_a of X^*.

By a composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ we get a Turing machine $\mathcal{M}_3$ which computes (w_a, b) for a given number n. More precisely, one can say that $\mathcal{M}_3$ computes the n-th element of $X^* \times \mathbb{N}_0$, which is ordered according to $\prec$ and the lexicographic order.

We now construct a deterministic Turing machine $\mathcal{M}$ which satisfies the requirements of the statement. Let $n \in mathbbN_0$. The machine $\mathcal{M}$ first computes (w_a, b) as $\mathcal{M}_3$ and copies w_a. Then it simulates $\mathcal{M}'$ and computes the configuration of $\mathcal{M}'$ after b steps on the input w_a. If the obtained configuration is an accepting configuration of $\mathcal{M}'$, then $\mathcal{M}$ gives the output w_a. Otherwise, $\mathcal{M}$ gives the output u.

From the construction it follows that, for any input n, $\mathcal{M}$ gives an output which is a word in L (the accepted word w_a or u). Thus the range of $f_{\mathcal{M}}$ is a subset of L.

Moreover, if $w \in L$, then $\mathcal{M}'$ accepts w after a certain number of steps, say k steps. If w is the l-th element of X^* according to the lexicographic order, then there is a number q such that $\mathcal{M}_3$ computes (w,k) on input q. Then $\mathcal{M}$ gives the output w. Hence any word of L occurs in the range of $f_{\mathcal{M}}$. Hence L is contained in the range of $f_{\mathcal{M}}$. □

We can rewrite the preceding theorem as follows: *For any recursively enumerable language L, there is a Turing computable function which is a total function and has domain* $\mathbb{N}_0$ *and range L.* This statement justifies the notation "recursively enumerable".[2]

Theorem 3.28 *A language L is recursively enumerable if and only if L is the range of a Turing computable function.*

Proof. By Theorem 3.27, we have only to prove that the range of a Turing computable function is a recursively enumerable set.

Let $f : X^* \to Y^*$ be a function computed by a Turing machine $\mathcal{M}$. In the proof of Theorem 3.27 we have defined an order on $X^* \times \mathbb{N}_0$. We construct the Turing machine $\mathcal{M}'$ which works as follows: Let $w \in Y^*$ be given. The machine $\mathcal{M}'$ computes the pair

[2]In the beginning of a theory of computability and algorithms, one has considered functions mapping $\mathbb{N}_0^r$ to $\mathbb{N}_0^s$ (for some numbers $r \geq 1$ and $s \geq 1$) and has considered partially recursive functions as a model for computable function. A subset U of $\mathbb{N}_0$ was called *recursively enumerable*, if there is a total partially recursive function $f : \mathbb{N}_0 \to \mathbb{N}_0$ such that the range of f is U. Taking into consideration that partially recursive functions and Turing computable functions coincide up to a coding of inputs and outputs, the justification of our notion of recursively enumerable sets follows.

$(\lambda, 0)$. Given a pair (v, b), $\mathcal{M}'$ computes the configuration of $\mathcal{M}$ after b steps on the input v. If this is a halting configuration and $\mathcal{M}$ has computed w, then $\mathcal{M}$ accepts. Otherwise, it consider the pair (v', b') which follows on (v, b).

Obviously, if $\mathcal{M}$ accepts a word w, then w is in the range of $\mathcal{M}$. If w is not in the range of $\mathcal{M}$, then $\mathcal{M}'$ does not enter a halting configurations and performs an infinite computation. Furthermore, if w is in the range of $\mathcal{M}$, then there is an input word v and a number b such that $\mathcal{M}$ stops after b steps on input v with the result w. Hence any word of the range of $\mathcal{M}$ is accepted. □

Theorem 3.29 *A language L is recursive if and only if the characteristic function of L is a Turing computable function.*

Proof. Let L be a recursive language and let $\mathcal{M}$ be a Turing function which stops on any input and accepts L. We construct the Turing machine $\mathcal{M}'$ as follows: First $\mathcal{M}'$ simulates the work of $\mathcal{M}$. If $\mathcal{M}$ enters an accepting configuration, then $\mathcal{M}'$ continues by a cancellation of all symbols different from $*$ on the tape and writing a 1 on the tape. If $\mathcal{M}$ enters a rejecting configuration, then $\mathcal{M}'$ continues by a cancellation of all symbols different from $*$ on the tape and writing a 0. Obviously, $\mathcal{M}'$ computes the characteristic function of L.

Conversely, if $\mathcal{M}'$ computes the characteristic function of L, then we construct $\mathcal{M}$ which enters an accepting state q_a if a 1 was computed by $\mathcal{M}'$ and enters a rejecting state q_r if $\mathcal{M}$ computes a 0. Thus $\mathcal{M}$ stops on any input and accepts L. □

By Theorem 3.29, decidability of a set means that we have an Turing computable functions which answers correctly the question whether or not a given element belongs to a given language or not.

Any language L over some alphabet X can be described by a property characterizing the elements of L or equivalently L can be given in the form

$$\{x \mid x \text{ has property P}\},$$

(e. g. we can take the property that $\varphi_L(x) = 1$). Hence in the sequel we shall formulate the decidability of a language L given by a property P as a problem: Given an instance (or equivalently an element), decide whether or not it has a given property (which characterises a given language). The decision will be formulated as a question which has the answer "yes" if and only the instance has the property. Moreover, we shall give the instance and we shall formulate the question without referring directly to a language (i. e., a set of words); however, it is easy to present a reformulation which refers to languages. Furthermore, we shall We give some examples.

Halting Problem for Turing machines

Given:	a Turing machine $\mathcal{M}$ with input alphabet X and a word $w \in X^*$
Question:	Does $\mathcal{M}$ halt on the input w?

Using the encoding $w_{\mathcal{M}}$ of a Turing machine $\mathcal{M}$ given in the proof of Theorem 3.11 a corresponding formulation as a language is given by

$$L_{halt} = \{w_{\mathcal{M}}\$w \mid w \in X^*,\ w_{\mathcal{M}} \in \{0,1\}^* \text{ is the encoding of } \mathcal{M},\ w \in T(\mathcal{M})\},$$

where L_{halt} is a language over $X \cup \{0, 1, \$\}$ and \$ is a separator.

The

Post Correspondence Problem
Given: natural number n, alphabet X,
pairs (u_i, v_i) with $u_i, v_i \in X^*$ for $1 \le i \le n$
Question: Does there exist a natural number k and a sequence $i_1 i_2 \dots i_k$ of natural numbers with $1 \le i_j \le n$ for $1 \le j \le k$ such that

$$u_{i_1} u_{i_2} \dots u_{i_k} = v_{i_1} v_{i_2} \dots v_{i_k}$$

holds?

has the language description

$$\begin{aligned} L_{Post} \;=\; & \{(u_1\$v_1)(u_2\$v_2)\dots(u_n\$v_n) \mid n \in \mathbb{N},\ u_i, v_i \in X^* \text{ for } 1 \le i \le n, \\ & \text{there is a sequence } i_1 i_2 \dots i_k \text{ with } 1 \le i_j \le n \text{ for } 1 \le j \le k \\ & \text{such that } u_{i_1} u_{i_2} \dots u_{i_k} = v_{i_1} v_{i_2} \dots v_{i_k}\} \end{aligned}$$

over the alphabet $X \cup \{\$, (,)\}$.

We now consider the

Hamiltonian Path Problem
Given: a graph $G = (V, E)$ and two nodes v and v' from V
Question: Is there is a path containing each node exactly once and starting in v and ending in v'?

Let $V = \{v_1, v_2, \dots, v_n\}$ and $E = \{(v_{i_1}, v_{j_1}), (v_{i_2}, v_{j_2}), \dots, (v_{i_m}, v_{j_m})\}$. We consider the alphabet consisting of V, the separator \$ which is used instead of the comma, and the brackets (and). Then the Hamiltonian path problem can be described as

$$\begin{aligned} L_{Ham} \;=\; & \{v_1\$v_2\$\dots v_n\$(v_{i_1}\$v_{j_1})\$(v_{i_2}\$v_{j_2})\$\dots\$(v_{i_m}\$v_{j_m})\$v\$v' \mid \text{there is a path} \\ & v, u_1, u_2, \dots, u_{n-2}, v' \text{ such that } \{v_1, v_2, \dots, v_n\} = \{v, v', u_1, u_2, \dots, u_{n-2}\}\}. \end{aligned}$$

Definition 3.30 *We say that a problem is decidable if the corresponding language is decidable (or equivalently, recursive).*

Concerning the first two problems we have the following status of decidability.

Theorem 3.31 *The halting problem for Turing machines is undecidable.*

Proof. It is easy to show that the decidability of the halting problem implies the decidability of the language U given in the proof of Theorem 3.11. However, in the proof of Theorem 3.11, it was shown that U is not decidable. □

Theorem 3.32 *The Post correspondence problem for alphabets with at least two letters is undecidable.* □

For a proof of Theorem 3.32, we refer to [12].

We now present some notions from complexity theory.

Definition 3.33 *i) Let $L \subset X^*$ be a recursive language, and let $t : \mathbb{N} \to \mathbb{N}$ be a function. We say that L can be decided in time t, if there is a Turing machine $\mathcal{M}$ with $T(\mathcal{M}) = L$ such that, for any input $w \in X^*$, the halting computation path for w has a length bounded by $t(|w|)$.*

ii) Let L be a recursively enumerable language, and let $t : \mathbb{N} \to \mathbb{N}$ be a function. We say that L is nondeterministically accepted in time t, if there is a nondeterministic Turing machine $\mathcal{M}$ with $T(\mathcal{M}) = L$ such that, for any word $w \in L$, there is an accepting computation path for w with a length bounded by $t(|w|)$.

iii) Let f be a Turing computable function, and let $t : \mathbb{N} \to \mathbb{N}$ be a function. We say that f can be computed in time t, if there is a deterministic Turing machine $\mathcal{M}$ with $f_{\mathcal{M}} = f$ such that, for any w in the domain of f, the halting computation path for w has a length bounded by $t(|w|)$.

iv) Let L be recursive language or a recursively enumerable language, and let f be a Turing computable function. We say that L can be decided in polynomial time or L can be nondeterministically accepted in polynomial time and f can be computed in polynomial time, if there is a polynomial p such that L can be decided in time p or nondeterministically accepted in time p and f can be computed in time p, respectively.

Definition 3.34 *i) By $\mathbf{P}$ we denote the set of all recursive languages which can be decided in polynomial time.*

ii) By $\mathbf{N}P$ we denote the set of all recursively enumerable languages which can be nondeterministically accepted in polynomial time.

Obviously,

$$\mathbf{P} \subseteq \mathbf{N}P. \tag{3.5}$$

It is an open question – one of the most important open questions in theoretical computer science – whether this inclusion is proper or whether equality holds. Nowadays, the conjecture in the community is that the inclusion is proper.

Intuitively, we have equality if the hardest languages in $\mathbf{N}P$ can be decided in polynomial time. We now formalize this approach.

Definition 3.35 *Let $L_1 \subseteq X_1^*$ and $L_2 \subseteq X_2^*$ be two languages.*

i) We say that L_1 can be transformed to L_2, if there is a Turing computable function f which maps X_1^* onto X_2^* such that $w \in L_1$ if and only if $f(w) \in L_2$.

ii) We say that L_1 can be polynomially transformed to L_2, if the function f can be computed in polynomial time.

Lemma 3.36 *If L_1 can be polynomially transformed to $L_2 \in \mathbf{P}$ (or $L_2 \in \mathbf{N}P$), then $L_1 \in \mathbf{P}$ (or $L_1 \in \mathbf{N}P$, respectively).*

Proof. We give the proof for languages in $\mathbf{P}$.

Let $w \in X_1^*$ be given. We first compute $f(w)$ where f is the Turing computable function which polynomially transforms L_1 to L_2. Then we decide whether $f(w) \in L_2$. By assumption both steps can be done in polynomial time. Because $w \in L_1$ if and only if $f(w) \in L_2$, it is easy to see that L_1 can be decided in polynomial time. □

By the proof of Lemma 3.36, the decidability of L_1 cannot be more complex than the the decidability of L_2.

Definition 3.37 *i) A language L is called* $\mathbf{N}P$*-complete, if*

– $L \in \mathbf{N}P$,

– *any language* $L \in \mathbf{N}P$ *can be polynomially transformed to* L.

ii) We say that a problem is $\mathbf{N}P$*-complete, if the corresponding language is* $\mathbf{N}P$*-complete.*

By the above remark, intuitively, the $\mathbf{N}P$-complete languages are the hardest languages with respect to their decidability in the set $\mathbf{N}P$.

The importance of $\mathbf{N}P$-complete languages comes from the following statement.

Theorem 3.38 *The following three statements are equivalent.*

i) $\mathbf{P} = \mathbf{N}P$.

ii) $L \in \mathbf{P}$ *for any* $\mathbf{N}P$*-complete language* L.

iii) $L \in \mathbf{P}$ *for some* $\mathbf{N}P$*-complete language* L.

Proof. i) $\Longrightarrow$ ii) Let L be a $\mathbf{N}P$-complete language. By Definition 3.37, we get $L \in \mathbf{N}P$. By our assumption $\mathbf{P} = \mathbf{N}P$, we have $L \in \mathbf{P}$.

ii) $\Longrightarrow$ iii) holds trivially.

iii) $\Longrightarrow$ i). Let L be a $\mathbf{N}P$-complete language, and let L' be a language in $\mathbf{N}P$. By Definition 3.37. L' can be polynomially transformed to L. By our assumption iii) and Lemma 3.36, we get $L' \in \mathbf{P}$. Thus any language of $\mathbf{N}P$ is in $\mathbf{P}$. Hence $\mathbf{N}P \subseteq \mathbf{P}$. Together with (3.5), we get the statement i). □

Finally we give some examples of $\mathbf{N}P$-complete problems which will be used in the sequel. For the proofs of the $\mathbf{N}P$-completeness we refer to [7].

Theorem 3.39 *The*

Satisfiability Problem SAT of Propositional Calculus

Given:	*a formulae A of propositional calculus in conjunctive normal form*
Question:	*Does there exist an assignment of A such that A will get the value "true".*

is $\mathbf{N}P$*-complete.* □

Theorem 3.40 *The*

Restricted Satisfiability Problem $3 - SAT$ *of Propositional Calculus*

Given:	*a formulae A of propositional calculus in conjunctive normal form, where each disjunction contains exactly three literals*
Question:	*Does there exist an assignment of A such that A will get the value "true".*

is $\mathbf{N}P$*-complete.* □

Theorem 3.41 *The Hamilton path problem is* $\mathbf{N}P$*-complete.* □

We mention that a lot of $\mathbf{N}P$-complete problems is known from very different fields, e. g. from number theory, graph theory, elementary combinatorics, theory of data bases. Although some of them look very easy on the first view, scientist were not able to find polynomial algorithms for their decision (which would imply $\mathbf{P} = \mathbf{N}P$) or to prove that no polynomial algorithm exists (which would show $\mathbf{P} \subset \mathbf{N}P$).

3.2 Finite Automata versus Regular Grammars

In the preceding section we have given a characterization of the families of recursively enumerable languages and context-sensitive languages by means of Turing machines and linearly bounded automata, respectively. The aim of this section is the presentation of an analogous characterization of the family of regular languages. We start with the definition of the corresponding type of automata/machines.

Definition 3.42 *i) A* finite automaton *is a quintuple*

$$\mathcal{A} = (X, Z, z_0, F, \delta),$$

where

– *X and Z are alphabets,*
– *$z_0 \in Z$ and $F \subseteq Z$, and*
– *$\delta : (Z \times X) \to Z$ is a function.*

ii) The extension δ^* of δ to the domain $Z \times X^*$ is defined by

$$\delta^*(z, \lambda) = z \text{ for } z \in Z,$$
$$\delta^*(z, wx) = \delta(\delta^*(z, w), x) \text{ for } z \in Z, w \in X^*, x \in X.$$

iii) The language $T(\mathcal{A})$ accepted by $\mathcal{A}$ *is defined as*

$$T(\mathcal{A}) = \{w : w \in X^*, \delta^*(z_0, w) \in F\}.$$

As in the case of Turing machines, the elements of X and Z are called input symbols and states, respectively; z_0 is the initial state, and F is the set of accepting states; δ is also called transition function.

The function $\delta^*(z, w)$ gives the state which is entered after reading the word w and starting in z. This follows by induction on the length of the word. the setting $\delta^*(z, \lambda) = z$ ensures that no change of the state is done if no symbol is read. For $x \in X$, we have $\delta^*(z, x) = \delta(\delta^*(z, \lambda), x) = \delta(z, x)$ which proves that reading of one symbol gives the new state according to the transition function. Let z' be the state obtained by reading of w starting in z. If we read the word wx for some $x \in X$, then $\delta^*(z, wx) = \delta(\delta^*(z, w), x) = \delta(z', x)$ and thus $\delta^*(z, w)$ is the state in which the automaton is after reading wx. Therefore a word w is accepted by a finite automaton if it is in an accepting state after reading the word.

The behaviour of a finite automaton can be interpreted as follows: A finite automaton reads the input on a tape from left to right and changes its state according to its transition function in dependence of the read symbol and the current state. It starts in its initial states. A word is accepted if after reading the complete input word the finite automaton is in an accepting state.

By this interpretation, the finite automaton can be considered as a Turing machine with the following restrictions. The read/write head moves only to the right. It halts if it reads the first $*$ right of the input word, i. e., if it has read the complete input word, and accepts, if it is in an accepting state. Therefore halting states can be omitted since the halting is performed after reading the word. Furthermore, we can omit the writing of a symbol in the cell which is scanned, because the head is always moved to the right, and therefore the automaton will never see the written symbol which is left from the head. Therefore, the writing has no influence on the acceptance.

In order to describe a finite automaton $\mathcal{A}$ according to its definition it is necessary to give the sets X, Z and F, the state z_0 and the transition function δ. We shall also use a description by means of a directed graph $G = (Z, E)$ with labelled edges. The set of nodes coincides with the set of states, and there is a directed edge $(z, z'$ from z to z' labelled by $x \in X$ if and only if $\delta(z, x) = z'$. We distinguish the initial state by an arrow directed to the node z_0 and mark the accepting states by two circles (instead of one circle for the other states). In this description $\delta(z, x_1x_2 \dots x_n)$ with $x_i \in X$ for $1 \leq i \leq n$ holds if and only if there is a path from z to z' where the labels of the edges of the path are $x_1, x_2, \dots, x_n$. Therefore a word $x_1x_2 \dots x_n$ with $x_i \in X$ for $1 \leq i \leq n$ is accepted if and only if there is a path from the initial state z_0 to an accepting state where the labels of the edges of the path are $x_1, x_2, \dots, x_n$.

Let us consider some examples.

Example 3.43 Let the finite automaton $\mathcal{A} = (X, Z, z_0, F, \delta)$ be given by

$$\begin{aligned}
X &= \{a, b, c\} \\
Z &= \{z_0, z_1, z_2, z_3\}, \\
F &= \{z_2\}, \\
\delta(z, x) &= \begin{cases} z_1 & \text{for } z = z_0,\ x = a \\ z_2 & \text{for } z = z_1,\ x = a \\ z_0 & \text{for } z \in \{z_0, z_2\},\ x = c \\ z_3 & \text{otherwise} \end{cases} .
\end{aligned}$$

The representation of $\mathcal{A}$ by a directed graph is shown in Figure 3.2.

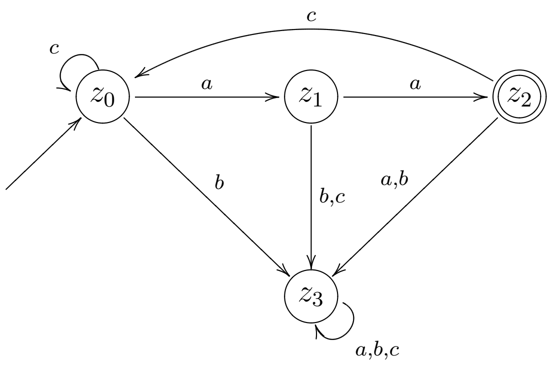

Figure 3.2: Automaton $\mathcal{A}$

We determine the set of words accepted by $\mathcal{A}$. We always use in our argumentation the representation of $\mathcal{A}$ by its components; however, the read is asked to follow the explanation also in the description by Figure 3.2.

We first note that, by $\delta(z_3, x) = z_3$ for all $x \in X$, the automaton cannot leave the state z_3. Therefore, if $\mathcal{A}$ enters z_3 after reading some word w, then no prolongation of w, i. e., no word wv can be accepted. Because $\delta(z, b) = z_3$ for all $z \in Z$, the automaton enters z_3 if a b is read. Consequently, the accepted words do not contain the letter b. Furthermore, we mention that

– the state z_0 is not changed as long as we read only only cs (or formally $\delta(z_0, c^{n-1}) = z_0$ for $n \geq 0$),
– if $\mathcal{A}$ is in state z_0 and reads an a, it has to read a second a (since the reading of b or c in state z_1 lead to state z_3), after reading both as, the automaton $\mathcal{A}$ is in state z_2,
– if $\mathcal{A}$ is in state z_2, it only avoids z_3 by reading a c which results in $\mathcal{A}$ in state z_0.

Therefore $T(\mathcal{A})$ consists of all words where a certain number of cs is always followed by aa and no b is present. Formally we get

$$T(\mathcal{A}) = \{c^{n_1}aac^{n_2}aa\dots c^{n_k}aa :\ k \geq 1,\ n_1 \geq 0,\ n_i \geq 1 \text{ für } 1 \leq i \leq k\}.$$

Example 3.44 We consider the finite automaton

$$\mathcal{B} = (\{a,b\}, \{z_0, z_a, z'_a, z'_b\}, \{z'_a, z'_b\}, \delta)$$

with

$$\delta(z_0, x) = z_x,\ \delta(z_x, x) = z'_x,\ \delta(z_x, y) = z_x,\ \delta(z'_x, x) = z'_x,\ \delta(z'_x, y) = z_x$$

for $x, y \in \{a,b\}$, $x \neq y$. A graphical representation of $\mathcal{B}$ is given in Figure 3.3.

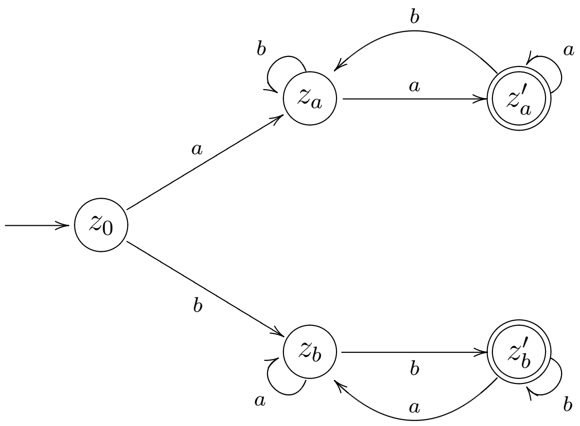

Figure 3.3: Automaton $\mathcal{B}$

Since $\delta(z_x, u), \delta(z'_x, u) \in \{z_x, z'_x\}$ for any input symbol u and any $x \in \{a,b\}$ we do not leave the set of states $\{z_x, z'_x\}$, $x \in \{a,b\}$, if the automaton $\mathcal{B}$ has entered it. Reading a or b as the first letter, $\mathcal{B}$ goes into $\{z_a, z'_a\}$ or $\{z_b, z'_b\}$. Furthermore, the finite automaton $\mathcal{B}$ is in state z'_x, if the last read letter is x. Thus the first and last letter of an accepted word have to coincide. Therefore we have

$$T(\mathcal{B}) = \{xzx \mid x \in \{a,b\},\ z \in \{a,b\}^*\}.$$

Example 3.45 We want to determine a finite automaton $\mathcal{A}$ such that

$$T(\mathcal{A}) = \{a^n b^m : n \geq 1, m \geq 2\}$$

holds.[3]

Obviously, we can choose $X = \{a, b\}$ for the input alphabet. Moreover, we use states in order to count the number of letters a and b which are read already. The following states reflect the following situations:

– z_1 – the automaton reads at least one a and no b,
– z_2 – the automaton reads at least one a and exactly one b,
– z_3 – the automaton reads at least one a and at least two bs.

In addition, we have to ensure that, for an accepted word, the first letter is a and no a is read, if the finite automaton has already read a b. A finite automaton satisfying all this requirements is presented in Figure 3.4.

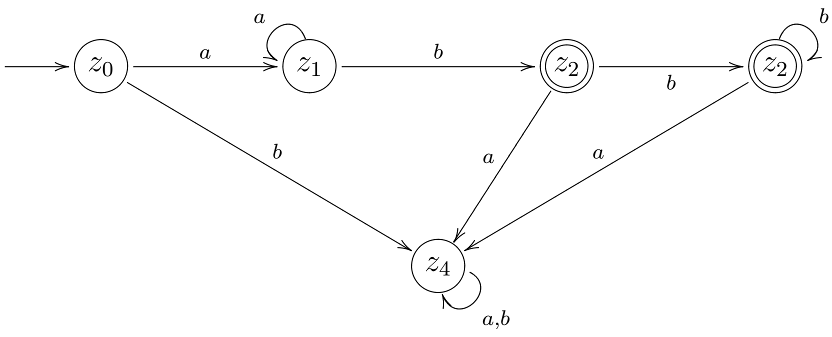

Figure 3.4: A finite automaton accepting $\{a^n b^m : n \geq 1, m \geq 2\}$

We also define a nondeterministic variant of the finite automaton. We follow the same line as in the case of Turing machines, i. e., the transition function maps on a set of states in stead of a single state.

Definition 3.46 *i) A* nondeterministic finite automaton *is a quintuple $\mathcal{A} = (X, Z, z_0, F, \delta)$, where X, Z, z_0, F are specified as Definition 3.42 and δ is a function which maps $Z \times X$ in the set 2^Z of subsets of Z.*

ii) We define the extension δ^* of δ as follows:

– *We set $\delta^*(z, \lambda) = \{z\}$ for $z \in Z$.*
– *For $w \in X^*$, $x \in X$ and $z \in Z$, $z' \in \delta^*(z, wx)$ if and only if there is a state $z'' \in \delta^*(z, w)$ such that $z' \in \delta(z'', x)$.*

iii) The language $T(\mathcal{A})$ accepted by the nondeterministic finite automaton $\mathcal{A}$ is defined as

$$T(\mathcal{A}) = \{w : \delta^*(z_0, w) \cap F \neq \emptyset\}.$$

Again, a finite automaton $\mathcal{A} = (X, Z, z_0, F, \delta)$ can be considered as a special case of the nondeterministic finite automaton $\mathcal{A}' = (X, Z, z_0, F, \delta')$ by taking $\delta'(z, x) = \{\delta(z, x)\}$. Then it is easy to show by induction on the length of the word that, for any $Z \in Z$ and $w \in X^*$, $(\delta')^*(z, xw)$ coincides with the single element set $\{\delta(z, w)\}$. Consequently, $\mathcal{A}$ and $\mathcal{A}'$ accept the same language.

[3]Here we do not discuss the question whether or not there is a finite automaton at all which accepts that language; but it will generally be solved in this subsection.

We now prove the equivalence of (deterministic) finite automata and nondeterministic finite automata with respect to their acceptance power which we have shown for Turing machines in Lemma 3.18.

Theorem 3.47 *For a language L, the following two statements are equivalent:*
i) *L is accepted by a (deterministic) finite automaton.*
ii) *L is accepted by a nondeterministic finite automaton.*

Proof. i) $\Rightarrow$ ii) follows immediately by the remarks given above that a (deterministic) finite automaton can be considered as a special nondeterministic finite automaton.

ii) $\Rightarrow$ i). Let $\mathcal{A} = (X, Z, z_0, F, \delta)$ be a nondeterministic finite automaton. We construct the (deterministic) finite automaton $\mathcal{A}' = (X, Z', z_0', F', \delta')$ by setting

$$\begin{aligned} Z' &= \{U : U \subseteq Z\}, \\ z_0' &= \{z_0\}, \\ F' &= \{U : U \in Z',\ U \cap F \neq \emptyset\}, \\ \delta'(U, x) &= \cup_{z \in U} \delta(z, x). \end{aligned}$$

By induction on the length of the word we show

$$(\delta')^*(\{z_0\}, w) = \delta^*(z_0, w) \tag{3.6}$$

f0r all words $w \in X^*$.

For $w = \lambda$, we obtain (3.6) directly from the definition of the transition functions. Thus the induction basis is done.

Let $w = w'x$. By the induction assumption $(\delta')^*(\{z_0\}, w') = \delta^*(z_0, w')$, we get

$$(\delta')^*(\{z_0\}, w'x) = \delta'((\delta')^*(\{z_0\}, w'), x) = \delta'(\delta^*(z_0, w'), x) = \cup_{z \in \delta^*(z_0, w')} \delta(z, x) = \delta^*(z_0, w'x).$$

Therefore $(\delta')^*(\{z_0\}, w) = \delta^*(z_0, w)$, and the induction step has been proved, too.

Let now $w \in T(\mathcal{A})$. Then $\delta^*(z_0, w) \cap F \neq \emptyset$. By definition of F', we get $\delta^*(z_0, w) \in F'$. By (3.6), we have $(\delta')^*(\{z_0\}, w) \in F'$, which proves that $w \in T(\mathcal{A}')$ also holds.

By inverting the arguments, we obtain that $w \in T(\mathcal{A}')$ implies $w \in T(\mathcal{A})$. Hence $T(\mathcal{A}) = T(\mathcal{A}')$ is shown. □

We now present the main result of this section which shows the regular grammars generate the same languages as deterministic and nondeterministic finite automata accept.

Theorem 3.48 *For a language L, the following three statements are equivalent:*
i) *L is regular.*
ii) *L is accepted by a nondeterministic finite automaton.*
iii) *L is accepted by a (deterministic) finite automaton.*

Proof. i) $\Rightarrow$ ii). We first prove the statement for the case that L does not contain the empty word.

Let $G = (N, T, P, S)$ be a regular grammar such that $L(G) = L$. According to Theorem 2.28, we can assume without loss of generality that all rules of P have the form

$A \to xB$ or $A \to x$ with $A, B \in N,\ x \in T$. We start with the construction of a regular grammar $G' = (N', T, P', S)$ with

$$\begin{aligned} N' &= N \cup \{\$\}, \\ P' &= \{A \to xB : A \to xB \in P\} \cup \{A \to x\$: A \to x \in P\} \cup \{\$ \to \lambda\}, \end{aligned}$$

where \$ is an additional symbol ($\$ \notin N \cup T$). Since the terminating derivations in G and G' have the forms

$$S \Longrightarrow^* wA \Longrightarrow wa$$

and

$$S \Longrightarrow^* wA \Longrightarrow wa\$ \Longrightarrow wa,$$

respectively, it is easy to see that $L(G) = L(G') = L$.

We now construct a nondeterministic finite automaton $\mathcal{A}$ such that $T(\mathcal{A}) = L$. This proves the assertion.

To do this we set $\mathcal{A} = (T, N', S, \{\$\}, \delta)$, where the transition function δ is defined by

$$\delta(A, x) = \{B : A \to xB \in P\}.$$

By induction on the length of the words, we prove the following statement (*).

(*) A derivation $A \Longrightarrow^* x_1x_2 \dots x_nB$ exists in G' if and only if $B \in \delta(A, x_1x_2 \dots x_n)$ holds.

The induction basis ($n = 1$) holds by definition of δ.

Let $A \Longrightarrow x_1x_2 \dots x_{n-1}B' \Longrightarrow x_1x_2 \dots x_{n-1}x_nB$ be a derivation in G'. By induction assumption and definition of δ, we have $B' \in \delta(A, x_1x_2 \dots x_{n-1})$ and $B \in \delta(B', x_n)$. Therefore $B \in \delta(A, x_1x_2 \dots x_{n-1}x_n)$ is valid.

Conversely, if $B \in \delta(A, x_1x_2 \dots x_n)$, then there is a state B' (i. e., a nonterminal B') such that $B \in \delta(B', x_n)$ and $B' \in \delta(A, x_1x_2 \dots x_{n-1})$. By induction assumption, there is a derivation $A \Longrightarrow^* x_1x_2 \dots x_{n-1}B'$ in G'. Moreover, by the definition of δ, $B' \Longrightarrow x_nB$ holds. Therefore there is a derivation $A \Longrightarrow^* x_1x_2 \dots x_{n-1}B' \Longrightarrow x_1x_2 \dots x_{n-1}x_nB$ in G'.

Thus the induction step also holds.

We consider a word $w \in L(G')$. Then there is a derivation $S \Longrightarrow^* w\$ \Longrightarrow w$ in G'. According to the assertion (*) shown above, $\$ \in \delta(S, w)$ and hence $w \in T(\mathcal{A})$.

Conversely, $w \in T(\mathcal{A})$ or equivalently $\$ \in \delta(S, w)$ implies the existence of a derivation $S \Longrightarrow^* w\$$ in G' by (*), and taking into consideration $\$ \to \lambda \in P'$ we get $S \Longrightarrow^* w$ or equivalently $w \in L(G')$.

Combining these facts we obtain $T(\mathcal{A}) = L(G')$. Since $L = L(G) = L(G')$, we have proved that $T(\mathcal{A}) = L$.

If $\lambda \in L$, we modify the construction as follows. The regular grammar in the normal form of Theorem 2.28 contains additionally the rule $S \to \lambda$ and S does not occur on the right hand side of a rule of P. We add this additional rule to P', too. Since this rule is only responsible for the generation of the empty word, we have also to add S to the set of accepting states of $\mathcal{A}$. Now we can repeat the above argumentation.

ii) $\Rightarrow$ iii) is valid by Theorem 3.47.

iii) $\Rightarrow$ i). Let a be a deterministic finite automaton $\mathcal{A} = (X, Z, z_0, F, \delta)$. We construct the regular grammar $G = (Z, X, P, z_0)$ with

$$P = \{z \to az' : z' \in \delta(z, a)\} \cup \{z \to \lambda : z \in F\}.$$

As in the first part of this proof we can show that $z \in \delta(z_0, w)$ for some $z \in F$ if and only if there is a derivation $z_0 \Longrightarrow^* wz \Longrightarrow w$ in G, which implies $T(\mathcal{A}) = L(G)$. $\square$

We illustrate the constructions given in the proofs of Theorems 3.47 and 3.48 by two examples.

Example 3.49 We consider the grammar

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

with P consisting of the rules

$$S \to \lambda, S \to aA, S \to a, S \to b, S \to bB, A \to a,$$
$$A \to b, A \to aA, A \to bB, B \to bB, B \to bB, B \to b.$$

First we construct the associated grammar

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

with

$$\begin{aligned} P' &= \{S \to \lambda, S \to aA, S \to a\$, S \to b\$, S \to bB, A \to a\$, \\ &\quad A \to b\$, A \to aA, A \to bB, B \to bB, B \to b\$, \$ \to \lambda\}. \end{aligned}$$

The nondeterministic finite automaton $\mathcal{B}$ accepting $L(G)$ is then given by

$$\mathcal{B} = (\{a, b\}, \{S, A, B, \$\}, S, \{S, \$\}, \delta)$$

with

$$\begin{aligned} &\delta(S, a) = \delta(A, a) = \{A, \$\}, \\ &\delta(S, b) = \delta(A, b) = \delta(B, b) = \{B, \$\}, \\ &\delta(B, a) = \delta(\$, a) = \delta(\$, b) = \emptyset. \end{aligned}$$

Finally, we construct the (deterministic) finite automaton $\mathcal{B}'$, which accepts the same set as $\mathcal{B}$. Following the construction given in the proof of Theorem 3.47, the set of states Z of $\mathcal{B}'$ is given by all subsets of $\{S, A, B, \$\}$ and the set F of accepting states is defined as all subsets which contain S or \$. Thus we get

$$\mathcal{B}' = (\{a, b\}, Z, \{S\}, F, \delta),$$

where δ is given by

$$\begin{aligned} &\delta'(\{S\}, a) = \delta'(\{A\}, a) = \delta'(\{S, A\}, a) = \delta'(\{S, B\}, a) = \delta'(\{A, B\}, a) \\ &\qquad = \delta'(\{S, A, B\}, a) = \{A, \$\}, \\ &\delta'(\{B\}, a) = \delta'(\emptyset, a) = \delta'(\emptyset, b) = \emptyset, \\ &\delta'(\{S\}, b) = \delta'(\{A\}, b) = \delta'(\{B\}, b) = \delta'(\{S, A\}, b) = \delta'(\{S, B\}, b) \\ &\qquad = \delta'(\{A, B\}, b) = \delta'(\{S, A, B\}, b) = \{B, \$\}, \\ &\delta'(U \cup \{\$\}, x) = \delta'(U, x) \cup \{\S\} \quad \text{for } U \subseteq \{S, A, B\}, x \in \{a, b\}. \end{aligned}$$

Example 3.50 In the preceding example, for a given regular grammar G, we have constructed a (nondeterministic) finite automaton which accepts $L(G)$. Here we are interested in the opposite direction. We present a regular grammar G which generates the set of words accepted by the finite automaton $\mathcal{A}$ of Example 3.43. According to the construction in the proof (part iii)) of Theorem 3.48, we obtain

$$G = (\{z_0, z_1, z_2, z_3\}, \{a, b, c\}, P, z_0)$$

with

$$\begin{aligned} P \;=\; &\{z_0 \to az_1, z_0 \to bz_3, z_0 \to cz_0, z_1 \to az_2, z_1 \to bz_3, z_1 \to cz_3, \\ &z_2 \to az_3, z_2 \to bz_3, z_2 \to cz_0, z_3 \to az_3, z_3 \to bz_3, z_3 \to cz_3\}. \end{aligned}$$

3.3 Push-Down Automata versus Context-Free Languages

In the two preceding sections we have presented characterizations of recursively enumerable, context-sensitive, and regular languages by Turing machines, linearly bounded automata, and finite automata, respectively. In this section we give an analogous characterization of context-free languages by a further device. Essentially finite automata cannot accept the context-free, but not regular languages $\{a^nb^n : n \geq 1\}$ or $\{wcw^R : w \in \{a, b\}^*\}$ because by means of a finite set of states we cannot remember the length or the structure of the word already read. In order to accept such languages we have to add a possibility to store information on the subword which has already been read. We use a work tape in addition to the input tape.

If we do not pose some restrictions to the work tape, then we can copy the input word from the input tape on work tape and then we work on this input word as in case of a Turing machine. Thus then we can obviously accept all languages which can be accepted by Turing machines, i. e., all recursively enumerable languages. However, we are looking for a device which only accepts context-free languages. Therefore we have to restrict the use of the work tape.

We consider the following restrictions:

- The symbol of the input can be read only from left to right, i. e., moves of the read head to the right are forbidden (however, in contrast to finite automata we allow that the head remains positioned on a cell for some time, which allows changes of the work tape without reading a symbol).

- A cell of the work tape contains the special symbol #. This symbol cannot be overwritten; thus it remains all the time of working in the cell. The read/write head of the work tape cannot move to the right of the cell marked with #. Therefore the work tape can be considered as a one sided infinite tape which is only infinite to the left.

- The work tape is handle like the data structure *Keller*. This means that we can change only the left most symbol of the tape and that we can add letters only to the left. Thus the symbols right from the left most symbol can only be involved in the work if all symbol left from it have been changed and finally cancelled. This type of work is also called *last in - first out* or abbreviated by LIFO).

The work tape is also called push-down tape.

This intuitive idea is graphically shown in Figure 3.5.

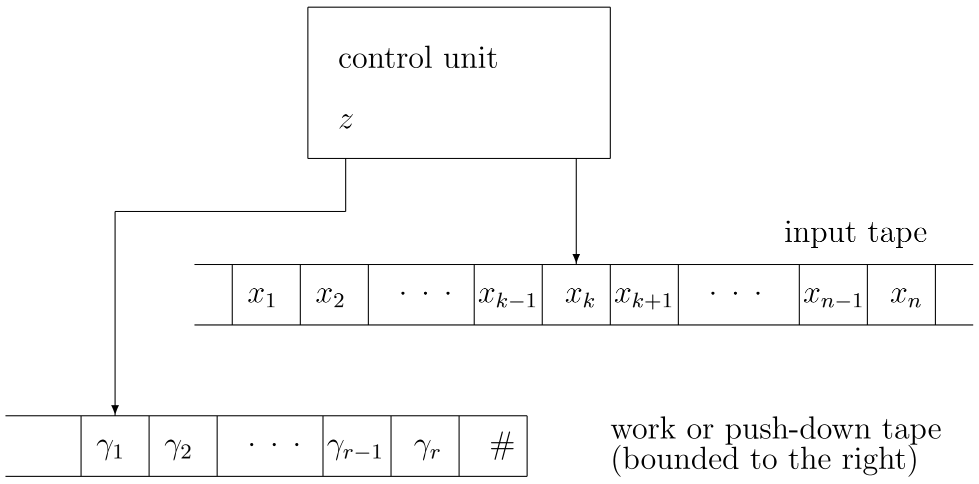
control unit
z
input tape
x_1 x_2 $\cdots$ x_{k-1} x_k x_{k+1} $\cdots$ x_{n-1} x_n
γ_1 γ_2 $\cdots$ γ_{r-1} γ_r #
work or push-down tape
(bounded to the right)

Figure 3.5: Schematic representation of a push-down automaton

We now give the formal definition of our new variant of automata.

Definition 3.51 *A* push-down automaton *is a sixtuple*

$$\mathcal{M} = (X, Z, \Gamma, z_0, F, \delta)$$

with

– *X is an input alphabet,*

– *Z is a finite set of states, - Γ is the alphabet of push-down symbols which can be written on the push-down tape, - $z_0 \in Z$ is the initial state and $F \subseteq Z$ is the set of accepting states,*
- δ is a function which maps $Z \times X \times (\Gamma \cup \{\#\})$ into the set of finite subsets of $Z \times \{R, N\} \times \Gamma^*$ where $\# \notin \Gamma$, R and N are additional symbols.

Definition 3.52 *Let $\mathcal{M} = (X, Z, \Gamma, z_0, F, \delta)$ be a push-down automaton as in Definition 3.51.*

A configuration *K of the push-down automaton $\mathcal{M}$ is a triple $(w, z, \alpha\#)$, where $w \in X^*$, $z \in Z$ and $\alpha \in \Gamma^*$.*

The transformation from a configuration K_1 to the (successor) configuration K_2 (denoted by $K_1 \models K_2$, again) is defined as follows: For $x \in X, v \in X^*, z \in Z, z' \in Z, \gamma \in \Gamma, \beta \in \Gamma^*, \alpha \in \Gamma^*$, we set

$$\begin{array}{lll}(xv,z,\gamma\alpha\#)\models(v,z',\beta\alpha\#), & \textit{for} & (z',R,\beta)\in\delta(z,x,\gamma),\\ (xv,z,\gamma\alpha\#)\models(xv,z',\beta\alpha\#), & \textit{for} & (z',N,\beta)\in\delta(z,x,\gamma),\\ (xv,z,\#)\models(v,z',\beta\#), & \textit{for} & (z',R,\beta)\in\delta(z,x,\#),\\ (xv,z,\#)\models(xv,z',\beta\#), & \textit{for} & (z',N,\beta)\in\delta(z,x,\#).\end{array}$$

Given a configuration $K=(w,z,\alpha\#)$, w is the suffix of the input word which is not read up that moment, z is the current state, α is the word on the push-down tape.

Intuitively, in dependence of the state which the push-down automaton is in at some moment, the symbol it reads on the input tape and the first symbol of the push-down tape a new state is determined, the push-down tape is changed by a replacement of its first letter by a word. More precisely, we have:

- $(z',R,\beta)\in\delta(z,x,\gamma)$ means that the push-down automaton, which is in state z, reads the symbol x on the input tape, and has γ as the first symbol on the push-down tape, enters the state z', moves the head of the input tape one position to the right, and replaces γ by the word β,

- $(z',N,\beta)\in\delta(z,x,\gamma)$ means that the that the push-down automaton, which is in state z, reads the symbol x on the input tape, and has γ as the first symbol on the push-down tape, enters the state z', does not move the head of the input tape[4], and replaces γ by the word β,

- $(z',R,\beta)\in\delta(z,x,\#)$ means that the push-down automaton, which is in state z, reads the symbol x on the input tape, and has only # written on the push-down tape, enters the state z', moves the head of the input tape one position to the right, and writes word β left from # on the push-down tape,

- $(z',N,\beta)\in\delta(z,x,\#)$ means that the push-down automaton, which is in state z, reads the symbol x on the input tape, and has only # written on the push-down tape, enters the state z', does not move the head of the input tape, and writes word β left from # on the push-down tape.

In all cases the read/write head of the push-down tape is positioned to the left most (or first) symbol on the push-down tape.

By $\models^*$ we denote the reflexive and transitive closure of the relation $\models$.

Definition 3.53 *Let $\mathcal{M}$ be a push-down automaton as in Definition 3.51. The* language accepted by *$\mathcal{M}$ is defined as*

$$T(\mathcal{M})=\{w:(w,z_0,\#)\models^*(\lambda,q,\#)\ \textit{for some}\ q\in F\}.$$

According to this definition, a word w is accepted, if
- in the beginning of its work the push-down automaton starts in z_0, has w on its input tape and has an empty push-down tape (i. e., the push-down tape contains only #),

[4]In some textbooks the move of the head of the input tape is interpreted in a different way: the symbol on the input tape is only read if it moves to the right, otherwise the automaton does not read the symbol.

– after reading w completely it is in an accepting state and has an empty push-down tape.

(Obviously, one can consider other variants of acceptance; for instance one can only ask for an accepting state independent from the content of the push-down tape or one can require that the push-down tape is empty, but one does not care on the state. One can prove that the change of the acceptance condition does not change the set of acceptable languages.)

Example 3.54 We consider the push-down automaton $\mathcal{M} = (X, Z, \Gamma, z_0, F, \delta)$ with

$$\begin{aligned}
&X = \{a, b\}, \quad \Gamma = \{a\}, \quad Z = \{z_0, z_1, z_2\}, \quad F = \{z_1\},\\
&\delta(z_0, a, \#) = \{(z_0, R, aa)\}, \quad \delta(z_0, a, a) = \{(z_0, R, aaa)\},\\
&\delta(z_0, b, a) = \{(z_1, R, \lambda)\}, \qquad \delta(q, b, a) = \{(z_1, R, \lambda)\}
\end{aligned}$$

and

$$\delta(z, x, \gamma) = \{(z_2, R, \gamma)\}$$

in all remaining cases. Then, for the input words $aabbbb$ und aba, we obtain the following sequences of configurations:

$$\begin{aligned}
(aabbbb, z_0, \#) &\models (abbbb, z_0, aa\#) \models (bbbb, z_0, aaaa\#) \models (bbb, z_1, aaa\#)\\
&\models (bb, z_1, aa\#) \models (b, z_1, a\#) \models (\lambda, z_1, \#)
\end{aligned}$$

and

$$(aba, z_0, \#) \models (ba, z_0, aa\#) \models (a, z_1, a\#) \models (\lambda, z_2, \#).$$

Thus we have $aabbbb \in T(\mathcal{M})$ and $aba \notin T(\mathcal{M})$.

It is easy to see that,

– if $\mathcal{M}$ is in state z_0, reads an a on the input tape and an a or $\#$ as first letter of the push-down tape, then it writes in addition two letters on the push-down tape,

– if $\mathcal{M}$ reads the first b on the input tape, it enters state z_1 and any reading of b is accompanied by the cancellation of the leftmost a of the push-down tape,

– in all other situations, $\mathcal{M}$ enters z_2 and remains in this state.

.

Because we two as on the push-down tape if we read one a is, and we cancel only one a if we read a b, we the number of bs we read has to be the double of the number os as we read in order to obtain an empty push-down tape. Hence we get

$$T(\mathcal{M}) = \{a^n b^{2n} : n \geq 1\}.$$

Essentially, the intuitive idea which $\mathcal{M}$ follows can be described as follows: The structure of some part of the input word is stored on the push-down tape during the reading of this part, and then the remaining part is compared with the stored part if it is read.

An essentially different idea for the construction of a push-down automaton accepting $L = \{a^n b^{2n} : n \geq 1\}$ is based on a simulation of a derivation of a grammar generating L and a comparison of the obtained word on the push-down tape with the input word. However, there is a problem in the realization of this idea, because in a derivation we can replace nonterminals in arbitrary position of the current sentential form whereas we can only change the leftmost symbol of the push-down tape. This problem can be solved by a restriction to leftmost derivations (see Definition 2.29) and a comparison of the input word and the sentential form, if the leftmost symbol of the sentential form is a terminal one.

In order to give a push-down automaton which realizes the above mentioned idea, we need a context-free grammar which generates L. Such a grammar is

$$G = (\{S\}, \{a, b\}, \{S \to aSbb, S \to abb\}, S).$$

Then we get the push-down automaton

$$\mathcal{M}' = (\{a, b\}, \{z_0', z_1', z_2'\}, \{S, a, b\}, z_0', \{z_1'\}, \delta')$$

with

$$\delta'(z_0', x, \#) = \{(z_1', N, S)\} \quad \text{for } x \in \{a, b\}$$

(we initialize the push-down tape with the start symbol S of G, which is the first sentential form of any derivation),

$$\delta'(z_1', x, S) = \{(z_1', N, aSbb), (z_1', N, abb)\} \quad \text{for } x \in \{a, b\}$$

(we simulate the application of a rule for S on the push-down tape, i. e., we replace the leftmost symbol S on the push-down tape by a right hand side of a rule with left hand side S),

$$\delta'(z_1', x, x) = \{(z_1', R, \lambda)\} \quad \text{for } x \in \{a, b\}$$

(we compare the first symbol of the push-down tape with the letter read on the input tape) and

$$\delta'(z, x, \gamma) = \{(z_2', R, \lambda)\}$$

in all remaining cases. For the above considered input words $aabbbb$ and aba we obtain as possible sequences of configurations

$$\begin{aligned}(aabbbb, z_0', \#) &\models (aabbbb, z_1', S\#) \models (aabbbb, z_1', aSbb\#) \models (abbbb, z_1', Sbb\#)\\ &\models (abbbb, z_1', abbbb\#) \models (bbbb, z_1', bbbb\#) \models (bbb, z_1', bbb\#)\\ &\models (bb, z_1', bb\#) \models (b, z_1', b\#) \models (\lambda, z_1', \#)\end{aligned}$$

and

$$(aba, z_0', \#) \models (aba, z_1', S\#) \models (aba, z_1', abb\#) \models (ba, z_1', bb\#) \models (a, z_1', b\#) \models (\lambda, z_2', \#).$$

However, we have to mention that we have only give one possible sequence of configurations; there are further ones since there are further possible derivations. It is easy to see that, for aba in all cases we get finally $(\lambda, z_2', \#)$, too.

Let $Sb^{2n}\#$ be written on the push-down tape (for $n = 0$ we get this situation from initial configuration $(w, z_0, \#)$ by the first step $\mathcal{M}'$). Now the application of $S \to aSbb$ or $S \to abb$ is simulated on the push-down tape which yields $aSb^{2(n+1)}\#$ or $ab^{2(n+1)}\#$ on the push-down tape. In the former case we compare a with the scanned symbol of the input tape. If the comparison is affirmative, we get $Sb^{2(n+1)}\#$, i. e., a word of the same form as we started from); if the comparison is not affirmative, then we enter state z_2' and the word will not be accepted. In the latter case, we have to compare the (remaining) word of the input tape with ab^{2n+1} and accept or we enter z_2'.

Therefore an input word is accepted if and only if it coincides with a terminal sentential form of G. Hence $T(\mathcal{M}) = L(G) = L$.

The idea presented in the second part of Example 3.54 can be generalized to an arbitrary context-free grammar. This leads to the following lemma.

Lemma 3.55 *For any context-free language L, there is a push-down automaton $\mathcal{M}$ such that $T(\mathcal{M}) = L$.*

Proof. Let $\lambda \notin L$. Then there is a context-free grammar G which contains only rules $A \to v$ with $v \neq \lambda$ and satisfies $L(G) = L$. From G, we construct the push-down automaton

$$\mathcal{M} = (T, \{z_0, z_1, z_2\}, N' \cup T, z_0, \{z_1\}, \delta)$$

with

$$\begin{aligned}
&\delta(z_0, x, \#) = \{(z_1, N, S)\} \quad \text{for } x \in T,\\
&\delta(z_1, x, A) = \{(z_1, N, v) : A \to v \in P\} \quad \text{for } x \in T,\\
&\delta(z_1, x, x) = \{(z_1, R, \lambda)\} \quad \text{for } x \in T
\end{aligned}$$

and

$$\delta(z, x, \gamma) = \{(z_2, R, \lambda)\}$$

in all remaining cases.

First we note that – besides the initial configuration – the push-down automaton is in state z_1 or z_2. If $\mathcal{M}$ is in state z_2, the state is not changed by the transition function. Because z_2 is not an accepting state, we can accept the input, if the state z_2 is entered sometimes. Thus we investigate which configurations with state z_1 can be obtained. We show that

$$(w_1 w_2, z_0, \#) \models^* (w_2, z_1, v\#) \tag{3.7}$$

holds if and only if there is a leftmost derivation

$$S \Longrightarrow^* w_1 v \tag{3.8}$$

in G. If we choose $w_1 = w, w_2 = \lambda, v = \lambda$, we get that we can obtain the accepting configuration $(\lambda, z_1, \#)$ if and only if there is a leftmost derivation $S \Longrightarrow^* w$. Thus a word is accepted if and only it can be generated by a leftmost derivation in G. By Theorem 2.30, we have $T(\mathcal{M}) = L_l(G) = L(G) = L$, which proves the lemma.

(3.7) $\to$ (3.8). We use induction on the length of the word w_1 which is already read. For $w_1 = \lambda, w_2 = w, v = S$ the assertion is valid since starting from the initial configuration $(w, z_0, \#)$ we can only come to the configuration $(w, z_1, S\#)$ by one transformation and $S \Longrightarrow^* S$ is a leftmost derivation (with zero derivation steps).

Let now $(w_1 w_2, z_0, \#) \models^* (w_2, z_1, v\#)$ be a transformation such that $w_2 \neq \lambda$ (we need an input word which is longer than w_1 to perform the induction step) and there is a leftmost derivation $S \overset{*}{\underset{l}{\Longrightarrow}} w_1 v$. We distinguish three cases:

Case 1. $v = av'$ for some $a \in T$. If we also have $w_2 = aw_2'$, then

$$(w_1aw_2', z_0, \#) \models^* (aw_2', z_1, av'\#) \models (w_2', z_1, v'\#)$$

and

$$S \underset{l}{\overset{*}{\Longrightarrow}} w_1v = w_1av'$$

are valid which proves the assertion for the longer word w_1a. If $w_2 = bw_2'$ for some $b \in T$ with $a \neq b$ or w_2 is the empty word, then $\mathcal{M}$ moves in state z_2 and we cannot accept.

Case 2. $v = Av'$ for some $A \in N$. Furthermore, let $A \to Xx$ be a rule of P. Then we get by a simulation of this rule

$$(w_2, z_1, Av'\#) \models (w_2, z_1, Xxv'\#).$$

If $X \in T$, we obtain the situation discussed in Case 1. If $X \in N$, we continue by simulation until we simulate a rule where the first letter of the right hand side is a terminal, which to Case 1, again.

Case 3. $v = \lambda$. Since $w_2 \neq \lambda$, the push-down automaton enters z_2.

(3.8) $\to$ (3.7). We give a proof by induction on the length of the derivation in (3.8).

For $n = 0$, the assertion by choosing $w_1 = \lambda, w_2 = w$, the only initial transformation $(w_2, z_0, \#) \models (w_2, z_1, S\#)$ and the fact that only S can be generated in zero derivation steps.

Assume that the statement is already shown for $n \geq 0$. Let now

$$S \underset{l}{\overset{*}{\Longrightarrow}} w_1Au \underset{l}{\Longrightarrow} w_1v_1Bv_2u$$

be a leftmost derivation of length $n+1$, where the last step is an application of the rule $A \to v_1Bv_2$ or $A \to v_1$ with $v_1 \in T^*$, $B \in N$, $v_2 \in (N \cup T)^*$ (the nonterminal has to be present since otherwise the derivation has not length $n+1$). Because we consider leftmost derivation, we have $w_1 \in T^*$. By induction assumption, we get

$$(w_1w_2, z_0, \#) \models^* (w_2, z_1, Au\#).$$

By the definition of $\mathcal{M}$, we obtain

$$(w_2, z_1, Au\#) \models (w_2, z_1, v_1Bv_2u\#) \text{ or } (w_2, z_1, Au\#) \models (w_2, z_1, v_1u\#).$$

If $w_2 = v_1w_3$ for some w_3, this leads to

$$(v_1w_3, z_1, v_1Bv_2u\#) \models^* (w_3, z_1, Bv_2u\#) \text{ or } (v_1w_3, z_1, u\#) \models^* (w_3, z_1, Bv_2u\#).$$

By a combination of these relations we yield

$$(w_1v_1w_3, z_0, \#) \models^* (w_3, z_1, Bv_2u\#) \text{ or } (w_1v_1w_3, z_0, \#) \models^* (w_3, z_1, u\#).$$

In the former case, we have the wanted relation. In the latter case,

– if u is a terminal word and $w_3 = u$, we get

$$(w_1v_1w_3, z_0\#) \models^* (w_3, z_1, u\#) \models^* (\lambda, z_1, \#)$$

and thus the assertion,

– if u is a terminal word and $w_3 \neq u$, $\mathcal{M}$ enters z_2,
– if u contains a nonterminal C and there is a word $p \in T^*$ such that $w_3 = pCw_4$ and $u = pCu'$, we get

$$(w_1v_1w_3, z_0\#) \models^* (pCw_4, z_1, pCu'\#) \models^* (w_4, z_1, u'\#)$$

and thus the statement,
– if u contains a nonterminal C and there is no $p \in T^*$ such that $w_3 = pCw_4$ and $u = pCu'$, $\mathcal{M}$ enters z_2.

Thus, for all accepted words, the statement holds for derivations of length $n+1$, too.

If $\lambda \in L$, we have the additional rule $S \to \lambda$. Thus it is sufficient to add z_0 to the set of accepted states in the above construction to ensure the acceptance of λ. By the proof given above, by z_1 we accept all words of $L(G) \setminus \{\lambda\}$. Consequently, L is accepted. □

The converse statement of Lemma 3.55 also holds.

Lemma 3.56 *For any push-down automaton $\mathcal{M}$, there is a context-free grammar such that $L(G) = T(\mathcal{M})$.*

Proof. ???? □

If we combine the two preceding lemmas we get the main result of this section.

Theorem 3.57 *For a language L, the following two statements are equivalent:*
i) *The language L is context-free (i. e., L is generated by a context-free grammar).*
ii) *The language L is accepted by a push-down automaton.*