Formal Languages

An Introduction

Andreas de Vries

Version: March 24, 2012

Contents

Chapter 1

Formal languages

1.1 Basic definitions

The theory of formal languages deals with the systematic analysis, classification, and construction of sets of words generated by finite alphabets. The key ideas of formal languages originate in linguistics. Linguistic objects are structured objects, and a grasp of the meaning of a sentence depends crucially on the speaker's or listener's ability to recover its structure, an ability which is likely to be unconscious for a native speaker of the language in question. A computational device which infers structure from grammatical strings of words is known as a "parser." Formal languages, such as propositional calculus, Java, or Lisp, have well-defined unique grammars. Natural languages, however, are riddled with ambiguities at every level of description, from the phonetic to the sociological. Jokes for instance usually exploit ambiguities, in particular word plays: *"Time flies like an arrow, fruit flies like a banana"* puns on the different meanings of the words flies and like. So, formal languages do not seem appropriate for telling jokes.

Definition 1.1. *(Alphabet, word, language)* An *alphabet* Σ is a nonempty finite set of symbols. With $\Sigma^0 := \{\varepsilon\}$ denoting the set of the *empty word* ε, and $\Sigma^1 := \Sigma$ we define

$$\Sigma^{n+1} := \{xy|\ x \in \Sigma,\ y \in \Sigma^n\} \qquad (n > 0) \tag{1.1}$$

$$\Sigma^+ := \bigcup_{k=1}^{\infty} \Sigma^k \tag{1.2}$$

$$\Sigma^* := \bigcup_{k=0}^{\infty} \Sigma^k = \Sigma^+ \cup \{\varepsilon\}. \tag{1.3}$$

An element $w \in \Sigma^*$ is called a *word* over the alphabet Σ. The *length* of a word w, written $|w|$, is the number of symbols it contains. A subset $L \subset \Sigma^*$ is called a *language* over the alphabet Σ. [10, §4.1], [19, pp 13] $\triangleleft$

Remark 1.2. A usual data type of programming languages has as underlying alphabet Σ the set of all ASCII symbols or all Unicode symbols, where usually the symbols are called *characters* and the words over these alphabets are called *strings*. In bioinformatics, important alphabets are $\Sigma_{\text{DNA}} = \{\text{A, C, G, T}\}$ representing the four DNA nuclebases, $\Sigma_{\text{RNA}} = \{\text{A, C, G, U}\}$ representing the four RNA nuclebases, or $\Sigma_{\text{Amin}} = \{\text{A, C, D, ...}\}$ representing the 22 amino acids. The words over these alphabets are usually called *sequences*. ♢

Example 1.3. For the alphabet $\Sigma = \{a, b\}$ we have

$$\begin{aligned}
\Sigma^0 &= \{\varepsilon\} \\
\Sigma^1 &= \{a, b\} \\
\Sigma^2 &= \{aa, ab, ba, bb\} \\
&\vdots \\
\Sigma^+ &= \{a, b, aa, ab, ba, bb, aaa, aab, aba, \dots\} \\
\Sigma^* &= \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \dots\}
\end{aligned}$$

Then $L = \{aa, ab\}$ is a language over Σ. ♢

Given a language L, is it possible to find a description by which all words can be systematically derived? If the formation of words of a language follow structured rules, they can be generated by a grammar.

Definition 1.4. A *grammar* G is a four-tuple (V, Σ, P, S) consisting of

- a finite set of *variables* V,
- a finite alphabet Σ of *terminals* satisfying $V \cap \Sigma = \emptyset$,
- a finite set P of *productions*, or *substitution rules*, each one having the form $l \to r$ with $l \in (V \cup \Sigma)^+$ and $r \in (V \cup \Sigma)^*$. Alternative options are listed with a logical OR sign $|$, e.g., $l \to a \mid b$ means that l may be substituted either by a or b.
- a start variable $S \in V$.

If two words x, $y \in (V \cup \Sigma)^*$ have the form $x = lur$ and $y = lvr$ with l, $r \in (V \cup \Sigma)^*$, then we say "y can be *derived* from x" and write $x \Rightarrow y$ if the grammar contains a production of the form $u \to v$. Moreover, we write $x \overset{*}{\Rightarrow} y$ if y can be derived from x in finitely many steps. ◁

Definition 1.5. Any grammar G generates a language

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\} \tag{1.4}$$

consisting of the words which can be derived from the start variable S. ◁

Example 1.6. *(Dyck language)* Let be given $\Sigma = \{\texttt{(}, \texttt{)}, \texttt{[}, \texttt{]}\}$, the variable set $V = \{S\}$ with the start variable S as its only element, and the production rules P

$$S \to \varepsilon \mid SS \mid [S] \mid (S). \tag{1.5}$$

Then $G_{\text{Dyck}} = (V, \Sigma, P, S)$ is a grammar. The language $D_2 := L(G_{\text{Dyck}})$ generated by it includes the word `()[()]()`, but `([)]` $\notin D_2$. ♢

Example 1.7. *(Simple English)* Let Σ be the Latin alphabet with the 26 lowercase letters, the space character and the full stop, the variables

$$\begin{aligned} V = \{ &\langle\text{sentence}\rangle, \langle\text{subject}\rangle, \langle\text{predicate}\rangle, \langle\text{object}\rangle, \\ &\langle\text{article}\rangle, \langle\text{adjective}\rangle, \langle\text{noun}\rangle \} \end{aligned} \tag{1.6}$$

the start variable $\langle\text{sentence}\rangle$, and the production rules P

$$\begin{aligned} \langle\text{sentence}\rangle &\to \langle\text{subject}\rangle\ \langle\text{predicate}\rangle\ \langle\text{object}\rangle. \\ \langle\text{subject}\rangle &\to \langle\text{article}\rangle\ \langle\text{adjective}\rangle\ \langle\text{noun}\rangle \\ \langle\text{article}\rangle &\to \texttt{a} \mid \texttt{the} \\ \langle\text{adjective}\rangle &\to \texttt{sweet} \mid \texttt{quick} \mid \texttt{small} \\ \langle\text{noun}\rangle &\to \texttt{duck} \mid \texttt{frog} \mid \texttt{mouse} \mid \texttt{hippopotamus} \\ \langle\text{predicate}\rangle &\to \texttt{likes} \mid \texttt{catches} \mid \texttt{eats} \\ \langle\text{object}\rangle &\to \texttt{cookies} \mid \texttt{chocolate} \mid \texttt{pizza} \end{aligned} \tag{1.7}$$

Then $G_{\text{SE}} = (V, \Sigma, P, \langle\text{sentence}\rangle)$ is a grammar. The language $L(G_{\text{SE}})$ generated by it includes the sentences "`the small duck eats pizza.`" and "`a quick mouse catches cookies.`" ◊

Often a word of a given language is intended to represent a certain object other than a string, such as a graph, a polynomial, or a machine. However, it is easy to find an injective mapping from the set of objects to the strings. (This is what every computer software as a binary string does solving problems dealing with real-life objects such as addresses, persons, invoices, games, ...). Our notation for the encoding of such an object A into its representation as a string is

$$\langle A \rangle \in \Sigma^*. \tag{1.8}$$

(Do not mix up this expression for encoding with the tag notation in the grammar of Example 1.7 above.) Usually, the underlying alphabet is $\Sigma = \{0, 1\}$. If we have several objects A_1, A_2, ..., A_k, we denote their encoding into a single string by $\langle A_1, A_2, \ldots, A_k \rangle$. The encoding can be done in many reasonable ways. If the objects are well-defined, however, strings of different encodings can be translated into each other in a unique way [19, §3.3]. For instance, in the sequel we will denote the pair $\langle M, x \rangle$ for the encoding of a Turing machine M (an "algorithm") and an input string x.

1.2 The Chomsky hierarchy

In 1957 the US linguist Noam Chomsky postulated a framework by which formal grammars can be subdivided into four types, according to the following definitions.

Definition 1.8. (a) *Recursively enumerable*, or *phrase-structure grammars (type-0 grammars).* Any grammar according to Definition 1.4 is a *recursively enumerable grammar*.

(b) *Context-sensitive grammars (type-1 grammars).* A grammar is *context-sensitive* if for any production $l \to r$ we have $|r| \geqq |l|$. In particular, a production can never lead to a shortening of a derived word.

(c) *Context-free grammars (type-2 grammars).* A grammar is *context-free* if for any production $l \to r$ we have $l \in V$. Therefore, the left hand side of a production always consists of a single variable.

(d) *Regular grammars (type-3 grammars).* A grammar is *regular* if for any production $l \to r$ we have $l \in V$ and $r \in \{\varepsilon\} \cup \Sigma V$. Therefore, a regular grammar is context-free and its right hand side is either empty or is a terminal followed by a variable. $\lhd$

In a context-free grammar, the variable on the left side of the production can be rewritten by a certain word on the right regardless of the context it finds itself in the current sentential form: it is independent of its context. In contrast, in context-sensitive grammars a variable can be substituted by a certain variable only when it is in the context of other variables or terminals preceding or following it [Mo].

Definition 1.9. A language is called to be *of type n* if there exists a grammar of type n which generates L. The set of all type-n languages is denoted by $\mathscr{L}_n$, for $n = 0, 1, 2, 3,$ $\lhd$

Lemma 1.10. *Let the following languages be given.*

$$L_{C3} := \{(ab)^n \mid n \in \mathbb{N}\}, \tag{1.9}$$
$$L_{C2} := \{a^n b^n \mid n \in \mathbb{N}\}, \tag{1.10}$$
$$L_{C1} := \{a^n b^n c^n \mid n \in \mathbb{N}\}, \tag{1.11}$$
$$L_{C0} := \{\langle M, x\rangle \mid M \text{ is a Turing machine halting on string } x\}, \tag{1.12}$$
$$L_{d} := \{\langle M, x\rangle \mid M \text{ is a Turing machine not accepting } x\}. \tag{1.13}$$

For $k = 0, 1, 2, 3$ then L_{C_k} is a language of type k, but L_{Ck} is not a language of type $(k+1)$ for $k \leqq 2$. In other words, L_{C3} is regular, L_{C2} is context-free but not regular,

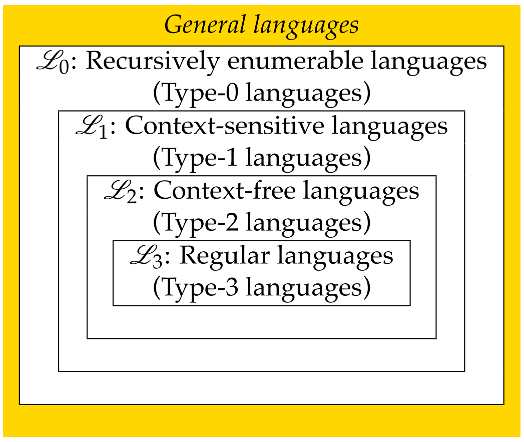

General languages
$\mathscr{L}_0$: Recursively enumerable languages (Type-0 languages)
$\mathscr{L}_1$: Context-sensitive languages (Type-1 languages)
$\mathscr{L}_2$: Context-free languages (Type-2 languages)
$\mathscr{L}_3$: Regular languages (Type-3 languages)

Figure 1.1: The Chomsky hierarchy

L_{C1} is context-sensitive but not context-free, and L_{C0} is recursively enumerable but not context-sensitive. We thus have the strict set inclusions

$$\mathscr{L}_3 \subset \mathscr{L}_2 \subset \mathscr{L}_1 \subset \mathscr{L}_0, \tag{1.14}$$

Moreover, L_d is not recursively enumerable, i.e., $L_d \notin \mathscr{L}_0$.

Proof. The proof is given by the following examples 1.12–1.15, Lemma 1.16, and the corollaries 1.19, 1.28, 1.36 below. [10, §4] □

Remark 1.11. The language $L_{\mathrm{C}0}$ in (1.12) is often also called the "halting problem," and the language L_{d} in (1.13) the "diagonal language." ♢

Example 1.12. Let $G_3 = (\{B, C, S\}, \{a, b\}, P, S)$ be a grammar, with the productions P given by

$$\begin{aligned} S &\to aB \\ B &\to bC \\ C &\to \varepsilon \mid aB. \end{aligned}$$

Then G_3 is a regular grammar since the left hand side of each production consists of a single variable and each right hand side is either empty a terminal followed by a variable. Hence the language $L(G_3)$ generated by G_3 is given by

$$L(G_3) = \{ab, abab, ababab, \dots\}, \tag{1.15}$$

i.e., $L(G_3) = L_{\mathrm{C}3}$. In particular, $L_{\mathrm{C}3} \in \mathscr{L}_3$, that is, $L_{\mathrm{C}3}$ is regular. ♢

Example 1.13. Let $G_2 = (\{S\}, \{a, b\}, P, S)$ be a grammar, with the productions P given by

$$S \to aSb \mid ab.$$

Then G_2 is a context-free grammar according to Definition 1.8. Hence the language $L(G_2)$ generated by G_2 is given by

$$L(G_2) = \{ab, aabb, aaabbb, \dots\}, \tag{1.16}$$

i.e., $L(G_2) = L_{\mathrm{C}2}$. In particular, $L_{\mathrm{C}2} \in \mathscr{L}_2$, that is, $L_{\mathrm{C}2}$ is context-free. ♢

Example 1.14. Let $G_1 = (\{A,\ B,\ C,\ S\},\ \{a,\ b,\ c\},\ P,\ S)$ be a grammar, with the productions P given by

$$\begin{aligned} S &\to SABC \mid ABC, \\ BA &\to AB,\ \ CB \to BC,\ \ CA \to AC, \\ AB &\to ab,\ \ BC \to bc, \\ Aa &\to aa,\ \ Ab \to ab,\ \ bB \to bb,\ \ cC \to cc. \end{aligned}$$

Then G_1 is a context-sensitive grammar according to Definition 1.8. Hence the language $L(G_1)$ generated by G_1 is given by

$$L(G_1) = \{abc, aabbcc, aaabbbccc, \dots\}, \tag{1.17}$$

i.e., $L(G_1) = L_{\mathrm{C}1}$. In particular, $L_{\mathrm{C}1} \in \mathscr{L}_1$, that is, $L_{\mathrm{C}1}$ is context-sensitive. ♢

Example 1.15. For the halting problem we have $L(G_0) \in \mathscr{L}_0$. For the proof we first have to show that there exists a Turing machine accepting the halting problem, namely the machine $\tilde{U}$ working on input $\langle M, x\rangle$ where M is a Turing machine and x a string, defined as

$$\tilde{U}(\langle M, x\rangle) = \textit{accept} \text{ if } M \text{ halts on } x. \tag{1.18}$$

($\tilde{U}$ is a slight modification of a universal Turing machine). Next we have to construct a grammar which simulates the actions of $\tilde{U}$ and generates a terminal string if $\tilde{U}$ accepts; such a grammar is given, for instance, in [12, Satz 9.4]. $\lozenge$

Lemma 1.16. *The diagonal language L_d in (1.13) is not recursively enumerable.*

Proof. [11, §9.1.4] According to the definition, we have to prove that there exists no Turing machine which accepts L_d. One key property of Turing machines is that we can enumerate them such that M_1, M_2, ... exhaust the set of all possible Turing machines, and assign a number string $w_i \in \{0,1\}^*$ to M_i for $i \in \mathbb{N}$, the *Gödel code*, [10, §6.1.5.6], [11, §9.1.2]. Thus L_d is the set of words w_i such that w_i is not in the language $L(M_i)$ of all strings accepted by M_i.

Assume that there exists a Turing machine M by which the diagonal language is accepted, i.e., $L_\mathrm{d} = L(M)$. Then there exists a number $i \in \mathbb{N}$ such that $M_i = M$ and a Gödel code w_i coding M_i. Now, is then $w_i \in L_\mathrm{d}$? (i) If $w_i \in L_\mathrm{d}$ then M_i excepts w_i; but then, by definition of L_d, w_i is not in L_d because L_d only contains those w_j such that M_j does *not* accept w_j. (ii) However, if $w_i \notin L_\mathrm{d}$, then M_i does not accept w_i and thus, by definition of L_d, w_i *is* in L_d. Since w_i thus can neither be in L_d nor fail to be in L_d, we conclude that there is a contradiction of our assumption that M exists. Hence L_d is not recursively enumerable. $\square$

1.3 Regular languages

Example 1.17. Let be $G_{abc} = (\{S, B, C\}, \{a, b, c\}, P, S)$ with the production P given by

$$\begin{aligned} S &\to aS \mid aB \\ B &\to bB \mid bC \\ C &\to cC \mid c. \end{aligned} \tag{1.19}$$

Then G_{abc} is a regular grammar. The language $L_{abc} := L(G_{abc})$ generated by it is

$$L_{abc} = \{a^i b^j c^k \mid i, j, k \in \mathbb{N}\}. \tag{1.20}$$

It is thus a regular language. $\lozenge$

The following theorem presents a technique for proving that a given language is *not* regular. The "pumping lemma" states that every regular language has a "pumping length" such that all longer strings in the language can be pumped by periodically repeating substrings.

Theorem 1.18 (Pumping lemma for regular languages). *A regular language L has a number $p \in \mathbb{N}$ where, if $s \in L$ is any word in L of length at least p, then it may be divided into three pieces*

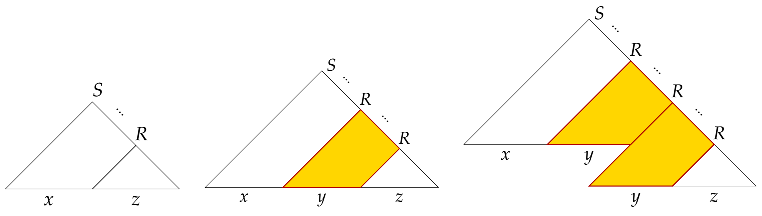
$$s = xyz \tag{1.21}$$

satisfying the conditions

$$xy^iz \in L \quad \text{for each } i \in \mathbb{N}_0, \tag{1.22}$$
$$|y| > 0, \tag{1.23}$$
$$|xy| \leqq p. \tag{1.24}$$

The number p is called pumping length *and is given as $p = |V| + 1$, where $|V|$ is the number of variables of a grammar G generating L.*

Proof. Regular languages have a certain word structure. If the number of derivation steps exceeds the limit p, then by the "pigeonhole principle"[1] at least one variable is repeated, say R. Then given the derivation chain $S \Rightarrow \ldots \Rightarrow R \Rightarrow \ldots \Rightarrow R \Rightarrow \ldots$, the derivation block between the two R's may be repeated arbitrarily often, as depicted in the following diagram for the words xz, xyz, xy^2z:



Therefore we may cut s into three pieces xyz and repeat the second piece and obtain a word still in the language, i.e., $xy^iz \in L$ for any $i = 0, 1, 2, \ldots$. For more details on the proof see [10, §4.3.2]. □

Corollary 1.19. *The language L_{C2} in Equation (1.10) is not regular.*

Proof. The proof is by contradiction. Assume that L_{C2} is regular. Then there exists a pumping length p according to the pumping lemma 1.18. Select the word $s = a^pb^p$. Clearly $s \in L_{C2}$ and $|s| > p$, and we can apply the pumping lemma to s. So we can find three words in L_{C2} such that $a^pb^p = xyz$, where condition (1.23) guarantees that y is nonempty. So there are three possible cases:

1. y consists only of a's: In this case the string $xyyz$ has more a's than b's and therefore is not in L_{C1}, contradiciting (1.22).
2. y consists only of b's: This case gives also a contradiction to (1.22).
3. y consists of both a's and b's: Then xy^2z may contain equal numbers of a's and b's, but not in the correct order since there are some b's before a's. Hence it cannot be in L_{C2}.

Therefore, it is impossible to subdivide s into xyz such that all three conditions (1.22) – (1.24) are satisfied. Hence our introductory assumption must be false, i.e., L_{C2} cannot be regular. □

[1] The *pigeonhole principle* is a fancy name for the rather obvious fact that if p pigeons are placed into fewer than p holes, some hole has to have more than one pigeon in it.

1.3.1 Regular expressions

Regular language can be elegantly described by regular expressions.

Definition 1.20. Given an arbitrary alphabet Σ, the set Regex_Σ of *regular expressions* is defined recursively by the following rules.

- $\emptyset, \varepsilon \in \text{Regex}_\Sigma$, $\Sigma \subset \text{Regex}_\Sigma$,
- If $r \in \text{Regex}_\Sigma$ then $(r) \in \text{Regex}_\Sigma$ and $r^* \in \text{Regex}_\Sigma$.
- If r, $s \in \text{Regex}_\Sigma$ then rs, $(r \mid s) \in \text{Regex}_\Sigma$.

Here the operator $\mid$ is the logical OR operator, the operator * is the "Kleene star" or "zero-or-more" operator, where

$$r^* \;:=\; \varepsilon \mid r \mid rr \mid rrr \mid \ldots, \tag{1.25}$$

and the operation $(\cdot)$ is called "grouping." $\lhd$

Lemma 1.21. *The neutral elements of* Regex_Σ *with respect to the* $\mid$ *operation are* $\emptyset$ *and* ε, *i.e.,*

$$r \mid \emptyset \;=\; \emptyset \mid r \;=\; r \mid \varepsilon \;=\; \varepsilon \mid r \;=\; r. \tag{1.26}$$

Moreover, for r, s, $t \in \text{Regex}_\Sigma$ *we have the law of idempotence,*

$$r \mid r \;=\; r, \tag{1.27}$$

the law of commutativity,

$$r \mid s \;=\; s \mid r, \tag{1.28}$$

and the laws of distributivity,

$$r\,(s \mid t) \;=\; rs \mid rt, \tag{1.29}$$

$$(s \mid t)\,r \;=\; sr \mid tr. \tag{1.30}$$

Definition 1.22. A regular expression $r \in \text{Regex}_\Sigma$ generates a language $L(r) = L_{\text{Regex}_\Sigma}(r)$ by the following recursive definition:

$$\begin{aligned}
L(\emptyset) &= \emptyset, \\
L(\varepsilon) &= \{\varepsilon\}, \\
L(a) &= \{a\} \qquad (a \in \Sigma), \\
L(rs) &= L(r)L(s), \\
L\left((r \mid s)\right) &= L(r) \,\cup\, L(s), \\
L(r^*) &= L(r)^*,
\end{aligned}$$

where r, $s \in \text{Regex}_\Sigma$. $\lhd$

Theorem 1.23. *The language* $L(ab)$ *generated by a regular expression* $ab \in \text{Regex}_{\{a,b\}}$ *over the alphabet* $\{a, b\}$ *is exactly* L_{C3} *in (1.9).*

Proof. $L(a, b) = ab(ab)^*$. $\square$

1.4 Context-free languages

The Dyck language in Example 1.6 and the simple English in Example 1.7 are both context-free languages. By Corollary 1.19, i.e., in the end by the pumping lemma, replacing a by "(" and b by ")" we see that the Dyck language is not regular.

Example 1.24. Let be $G_{\text{alg}} = (\{A, O, S\}, \{x,\ y,\ z,\ +,\ -,\ \cdot,\ /,\ (,\ )\},\ P,\ S)$ with the productions P

$$\begin{aligned} S &\to AOA \mid (S) \\ A &\to S \mid x \mid y \mid z, \quad O \to + \mid - \mid \cdot \mid / \end{aligned} \tag{1.31}$$

Then G_{alg} is a context-free grammar. The language $L_{\text{alg}} := L(G_{\text{alg}})$ generated by it is the set of all algebraic formulas (in "infix" notation) made of the three variables x, y, z, of the operations $+$, $-$, $\cdot$ and $/$, and of the round brackets. In particular,

$$(x+y)\cdot x - z\cdot y/(x-y) \in L_{\text{alg}}.$$

L_{alg} is thus a context-free language. $\Diamond$

Example 1.25. *(Simple HTML)* Let be $G_{\text{html}} = (V,\ \text{ASCII},\ P,\ S)$ with the set of variables

$$V = \{S, H, T, B, A\}$$

(where A represents plain text and the other variables "tags") and the production P

$$\begin{aligned} S &\to \texttt{<html>}HB\texttt{</html>} \\ H &\to \varepsilon \mid \texttt{<head>}T\texttt{</head>} \\ B &\to \varepsilon \mid \texttt{<body>}A\texttt{</body>} \\ T &\to \varepsilon \mid \texttt{<title>}A\texttt{</title>} \\ A &\to \varepsilon \mid \texttt{A}A \mid \texttt{B}A \mid \dots \mid \texttt{Z}A \mid \texttt{a}A \mid \texttt{b}A \mid \dots \mid \texttt{z}A \mid \dots \end{aligned}$$

Then G_{html} is a context-free grammar, and the language $L(G_{\text{html}})$ is context-free. For a more detailed study of HTML and XML see [11, §§5.3.3, 5.3.4]. $\Diamond$

Example 1.26. *(Simple Java)* Let be $G_{\text{Java}} = (V, \text{UTF-8}, P, S)$ with the set of variables

$$V = \{S, M, T, I, N, A\}$$

and the production P

$$\begin{aligned} S &\to \texttt{public class}\ N\ \texttt{\{}\ M\ \texttt{\}} \\ M &\to \varepsilon \mid T\ N\texttt{;} \mid \texttt{public}\ T\ N\ \texttt{\{}I\texttt{\}} \mid \texttt{public static}\ T\ N\ \texttt{\{}I\texttt{\}} \\ T &\to \texttt{void} \mid \texttt{boolean} \mid \texttt{int} \mid \texttt{double} \\ I &\to \varepsilon \mid A\texttt{;}\ I \\ N &\to \texttt{A}A \mid \texttt{B}A \mid \dots \mid \texttt{Z}A \mid \texttt{a}A \mid \texttt{b}A \mid \dots \mid \texttt{z}A \\ A &\to \varepsilon \mid \texttt{A}A \mid \dots \mid \texttt{Z}A \mid \texttt{a}A \mid \dots \mid \texttt{z}A \mid \texttt{0}A \mid \texttt{1}A \dots \end{aligned}$$

(Note that "`;`" is a terminal!) Here S represents a class, M a class member (attribute or method), T a data type, I an instruction, N a non-empty string, and A a string. Then G_{Java} is a context-free grammar, and the language $L_{\text{Java}} := L(G_{\text{Java}})$ therefore is context-free, too. $\Diamond$

Similarly to the pumping lemma 1.18 for regular languages, the following theorem supplies a technique for proving that a given language is *not* context-free. The "pumping lemma" states that every context-free language has a "pumping length" such that all longer strings in the language can be pumped with *two* substrings.

Theorem 1.27 (Pumping lemma for context-free languages). *A context-free language L has a number $p \in \mathbb{N}$ where, if $s \in L$ is any word in L of length at least p, then it may be divided into five pieces*

$$s = uvxyz \tag{1.32}$$

satisfying the conditions

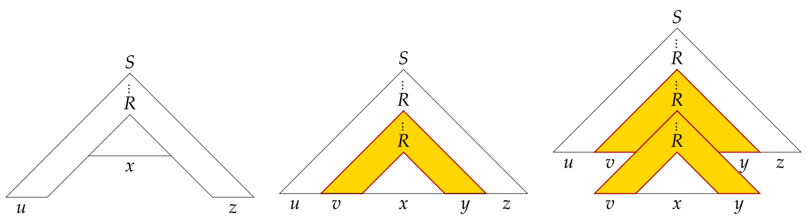
$$uv^i xy^i z \in L \quad \text{for each } i \in \mathbb{N}_0, \tag{1.33}$$
$$|vy| > 0, \tag{1.34}$$
$$|vxy| \leqq p. \tag{1.35}$$

The number p is called pumping length *and is given as*

$$p = k^{|V|+1} \tag{1.36}$$

where $k \geqq 2$ is the maximum number of symbols in the right-hand side of the production rules of a grammar G generating L, and $|V|$ is the number of variables of the grammar.

Proof. The main task of the proof is to show that any string s in L can be pumped and still remains in L. Because s is derivable from the grammar G, it has a parse tree. Since s is rather long, its parse tree is rather tall, that is, it must contain some long path from the from the start variable at its root to one of the terminal symbols at a leaf. On this long path then some variable symbol R must repeat because of the pigeonhole principle. As the following figure shows, this repetition allows us to replace the subtree under the second occurence of R and still get a legal parse tree for the words uxz, $uvxyz$, and uv^2xy^2z:



Therefore we may cut s into five pieces $uvxyz$ and repeat the second and fourth pieces and obtain a word still in the language, i.e., $uv^i xy^i z \in L$ for any $i = 0, 1, 2, \ldots$. For more details on the proof see [19, §2.3]. $\square$

Corollary 1.28. *The language L_{C1} in Equation (1.11) is not context-free.*

Proof. The proof is by contradiction. We assume that L_{C1} is context-free. Then by the pumping lemma 1.27 there exists a pumping length p. Select the word $s = a^p b^p c^p$. Clearly $s \in L_{C1}$ and $|s| > p$, and we can apply the pumping lemma to s. So we can find five words in L_{C1} such that $a^p b^p c^p = uvxyz$. First, condition (1.34) stipulates that either v or y is nonempty. So there are two possible cases:

1. Both v and y contain only one type of alphabet symbol: Then v cannot contain both a's and $b's$ or both b's and c's, and the same holds for y. In this case the string uv^2xy^2z cannot contain equal numbers of a's, b's, and c's, and therefore it is not in L_{C1}, contradiciting (1.33).

2. Either v or y contain more than one type of symbol: Then uv^2xy^2z may contain equal numbers of the three alphabet symbols a, b, c, but not in the correct order. Hence it cannot be in L_{C1}.

Therefore, it is impossible to subdivide s into $uvxyz$ such that all three conditions (1.33) – (1.35) are satisfied. Therefore, our introductory assumption must be false, i.e., L_{C1} cannot be context-free. □

More examples of languages for which the pumping lemma can be applied to prove that they are not context-free are $\{a^i b^j c^k |\ 0 \leqq i \leqq j \leqq k\}$ or $\{ww|\ w \in \{a,b\}^*\}$, as shown in [19, §2.3]. However, the pumping lemma only presents necessary conditions for non-context-free languages, but not sufficient conditions. That is, there may be languages which satisfy the conditions of the pumping lemma but are not context-free. An example is given in [10, §4.4]. However, a sharper necessary condition is known, called Ogden's lemma.

Theorem 1.29 (Ogden's lemma). *A context-free language L has a number $p \in \mathbb{N}$ where, if $s \in L$ is any word in L of length at least p, then it satisfies the following property: marking at least p symbols in s, the word s may be divided into five pieces*

$$s = uvwxy \tag{1.37}$$

satisfying the conditions

- *at least one symbol in vx is marked,*
- *at most p symbols are marked in vwx,*
- *$uv^iwx^iy \in L$.*

Proof. [11] □

Example 1.30. Let $G_{\text{Luk}} = (\{S\}, \{a, b\}, P, S)$ be a grammar, with the productions P given by

$$S \to aSS \mid b.$$

Then G_{Luk} is a comtext-free grammar according to Definition 1.8, but not regular. The language it generates is called the *Łukasiewicz language*. ◊

1.4.1 Linear languages

Definition 1.31. A context-free grammar is called *linear* if the right hand side of each of its productions contains at most one variable. A language is called *linear* if it can be generated by a linear grammar [12, p. 109]. ◁

Example 1.32. *(Palindrome language)* The grammar $G_{\text{pal}} = (\{S\}, \{a, b\}, P, S)$ with the productions

$$S \to aSa \mid bSb \mid a \mid b \mid \varepsilon \tag{1.38}$$

is linear. It generates the palindrome language $L_{\text{pal}} = \{ww^R \mid w \in \{a,b\}^*\}$. ♢

Proposition 1.33. *The palindrome language L_{pal} is not regular.*

Proof. Assume that L_{pal} is regular. Then by the pumping lemma 1.18 above the string $s = a^p b a^p$, with p being the pumping length, can be divided into three pieces $s = xyz$ with $|y| > 0$ and $|xy| \leqq p$. Thus y consists solely of a's. By the pumping lemma the string $s' = xy^2z \in L_{\text{pal}}$, but since s' contains more a's on the left hand side than on the right hand side it cannot be a palindrome. This contradiction proves that our assumption was wrong, i.e., L_{pal} is not regular. □

The linear languages are not closed under the $*$ operation [12, p. 150].

1.5 Context-sensitive languages

Nearly every imaginable language is context-sensitive. The only known proofs of the existence of not context-sensitive languages are based on the undecidability of the halting problem [11, §9.3].

Example 1.34. Let $G_{\text{exp}} = (\{A, B, C, D, E, F, G, S\}, \{a\}, P, S)$ be a grammar, with the productions P given by

$$\begin{gathered} S \to AB, \;\; B \to C \mid D, \;\; aC \to Ca, \;\; AC \to aa, \;\; AD \to AaaB, \\ aD \to EF, \;\; aE \to Ea, \;\; AE \to AaG, \;\; Ga \to aaG, \;\; GF \to aaaB. \end{gathered}$$

Then G_{exp} is a context-sensitive grammar according to Definition 1.8. Hence the language $L(G_{\text{exp}})$ generated by G_0 is given by

$$L(G_{\text{exp}}) = \{aa, aaaa, a^8, a^{16}, \ldots\}, \tag{1.39}$$

i.e., $L(G_{\text{exp}}) \in \mathscr{L}_1$. [12, Ex. 9.4 adapted] ♢

Proof. First we prove by induction over n that we can always derive

$$S \overset{*}{\Rightarrow} Aa^{2^n-2}B \tag{1.40}$$

for $n \in \mathbb{N}$. For $n = 1$ we have clearly $S \to AB$ by the first production rule, and from (1.40) for $n \in \mathbb{N}$ we achieve the respective equation (1.40) for $n+1$ by

$$\begin{aligned} Aa^{2^n-2}B &\Rightarrow Aa^{2^n-2}D \Rightarrow Aa^{2^n-3}EF \overset{*}{\Rightarrow} AEa^{2^n-3}F \\ &\Rightarrow AaGa^{2^n-3}F \overset{*}{\Rightarrow} Aa^{2^{n+1}-5}GF \to Aa^{2^{n+1}-2}B \end{aligned} \tag{1.41}$$

As a second derivation we directly see that

$$Aa^{2^n-2}B \Rightarrow Aa^{2^n-2}C \overset{*}{\Rightarrow} ACa^{2^n-2} \Rightarrow a^{2^n}. \tag{1.42}$$

With (1.40) this gives $S \overset{*}{\Rightarrow} a^{2^n}$ for any $n \in \mathbb{N}$. □

Example 1.35. [10, Abb. 4.26] Let $G'_{\text{exp}} = (\{D, L, S\}, \{a\}, P, S)$ be a grammar, with the productions P given by

$$S \to SD, \;\; SD \to LaD, \;\; aD \to Daa, \;\; LD \to L, \;\; L \to \varepsilon.$$

Then G'_{exp} is a recursively enumerable grammar according to Definition 1.8, but not context-sensitive because of the second to last substitution rule. However, the language $L(G'_{\text{exp}})$ generated by G'_{exp} is given by

$$L(G'_{\text{exp}}) = \{a, aa, aaaa, a^8, a^{16}, \ldots\}, \tag{1.43}$$

i.e., $L(G'_{\text{exp}}) = L_{\text{exp}}$ in Example 1.34. Therefore, a non-context-sensitive grammar can generate a context-sensitive language. ◊

Corollary 1.36. *The language L_{C0} in Equation (1.12) is not context-sensitive.*

Proof. We prove by contradiction that $H := L_{C0}$ is not decidable ("decidable" being the same as "recursive" in [11]). Since every context-sensitive language is decidable [12, Satz 9.7], this proves the assertion. Assume that L_{C0} is decidable. Then in particular the Turing machine $\tilde{U}$ in (1.18) exists, and we can construct the Turing machine D with input $\langle M \rangle$, where M is an arbitrary Turing machine, which runs H as a subroutine with the input $\langle M, \langle M \rangle \rangle$ and does the opposite of $\tilde{U}$, that is, D rejects if M accepts, and accepts if M rejects:

```
D = "On input ⟨M⟩, where M is a Turing machine,
    1. Run Ũ on input ⟨M, ⟨M⟩⟩;
    2. Output the opposite of Ũ."
```

Therefore,

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle, \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle. \end{cases} \tag{1.44}$$

and, applied to itself,

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle, \\ \textit{reject} & \text{if } D \text{ accepts } \langle D \rangle. \end{cases} \tag{1.45}$$

No matter what D does, applied on itself as input it is forced to output the opposite, which is a contradiction. Thus the halting problem L_{C0} cannot be decidable. □

For the solution of the decidability of the halting problem it was essential to apply $\tilde{U}(\langle M, \langle M \rangle \rangle)$, i.e., to run a Turing machine M on its own description, $M(\langle M \rangle)$. This is similar to running a program with itself as input, something which does occasionally occur in practice. For example, a compiler is a program which translates other programs; a compiler for the language Java may itself be written in Java, so it could be running on itself.

1.5.1 Indexed languages

Among the many proposed generalizations of context-free languages the "index languages," introduced by Aho in 1968, seem to be rather well adapted to natural languages [3], [GM, §4-1.6.3]. They use a set of indices working as a recursive stack.

Definition 1.37. [12, §14.3] An *indexed grammar* is a five-tuple (V, Σ, I, P, S) where V is a set of variables with the start variable $S \in V$, Σ is an alphabet, I is a set of indices, and the set P of productions having the form

$$A_{..} \to \alpha_{..}, \qquad A_{..} \to B_{f..}, \qquad A_{f..} \to \alpha_{..} \tag{1.46}$$

where $A, B \in V$, $f \in I$, $.. \in I^*$, and $\alpha \in (V \cup \Sigma)^*$. Whenever a production of the first kind is applied to A, the index string .. is attached to *each* variable in α (but not its terminals). For productions of the second kind, the index is added to the front of the A's index string and attached to B. By productions of the third kind, the index is removed from the index string, and the index reminder is distributed over the variables, as before. A language is called *indexed* if it can be generated by an indexed grammar. ◁

Example 1.38. [12, §14.3] Let $G = (\{S, T, A, B, C\}, \{a, b, c\}, \{f, g\}, P, S\}$ be an indexed grammar with the productions

$$\begin{array}{llll} S \to T_g, & A_f \to aA, & A_g \to a, \\ T \to T_f, & B_f \to bB, & B_g \to b, \\ T \to ABC, & C_f \to cC, & C_g \to c. \end{array}$$

For instance we have

$$\begin{aligned} S &\Rightarrow T_g \Rightarrow T_{fg} \Rightarrow A_{fg}B_{fg}C_{fg} \\ &\Rightarrow aA_gB_{fg}C_{fg} \Rightarrow aaB_{fg}C_{fg} \Rightarrow aabB_gC_{fg} \\ &\Rightarrow aabbC_{fg} \Rightarrow aabbcC_g \Rightarrow aabbcc, \end{aligned}$$

and thus in general

$$S \overset{*}{\Rightarrow} T_{f^ig} \Rightarrow A_{f^ig}B_{f^ig}C_{f^ig} \overset{*}{\Rightarrow} a^{i+1}b^{i+1}c^{i+1} \tag{1.47}$$

for $i \in \mathbb{N}_0$. Therefore, the language $L(G)$ generated by G is exactly L_{C1} in (1.11). According to Example 1.14 and Corollary 1.28, $L(G)$ is context-sensitive but not context-free. ♢

Theorem 1.39. *A context-free grammar is representable as an indexed grammar. In turn, an indexed grammar is context-sensitive.*

Proof. A context-free grammar $A \to \alpha$ directly induces an indexed grammar with an empty index set $I = \emptyset$. The second assertion is proved in [12, Theorem 14.7]. □

Global Index Languages

Global index grammars use the stack of indices as a global control structure during the entire derivation of a word. Global index grammars are a kind of regulated rewriting mechanism with global context and history of the derivation as the main characteristics of its regulating device [3, §1.2.2].

Definition 1.40. [3, §1.2.2] A *global index grammar* is a five-tuple (V, Σ, I, P, S) where V, Σ, I are pairwise disjoint sets, V denoting the set of variables, Σ the set of terminals, I the set of stack indices, $S \in V$ the start symbol, and P the set of productions of the form

$$A \to \alpha, \qquad A \cdot\cdot [..] \to \ a\alpha \cdot\cdot [i..] \quad \textit{(push)}, \qquad A \cdot\cdot [j..] \to \alpha \cdot\cdot [..] \quad \textit{(pop)},$$

with $A \in V$, $\alpha, \cdot\cdot \in (V \cup \Sigma)^*$, $a \in \Sigma$, $i \in I^*$, $j \in I$, and $.. \in I^*$. In contrast to a general indexed grammar as given by Definition 1.37, the global index $[\cdot\cdot]$ is *not* attached to each of its member variables. The second to last production is called a *push operation*, the last one a *pop operation*. The *derivation* $\Rightarrow$ is defined as in Definition 1.4. Then $L(G)$ is the language of G defined as $L(G) = \{w \in \Sigma | \ S[] \overset{*}{\Rightarrow} w[]\}$, analogously to (1.4). The empty stack often is simply omitted, i.e., $\alpha[] = \alpha$. $\lhd$

Example 1.41. *(Copy language)* [3, §1.2.2] Let $G_{\text{copy}} = (\{S, R, A, B, L\}, \{a, b\}, \{i, j\}, P, S)$ with the productions

$$\begin{aligned} S &\to AS \mid BS \mid RS \mid L, \\ R \cdot\cdot [i..] &\to RA \cdot\cdot [..], \quad R \cdot\cdot [j..] \to RB \cdot\cdot [..], \quad R \cdot\cdot [] \to \varepsilon \cdot\cdot [], \\ A \cdot\cdot [..] &\to a \cdot\cdot [i..], \quad B \cdot\cdot [..] \to b \cdot\cdot [j..], \\ L \cdot\cdot [i..] &\to La \cdot\cdot [..], \quad L \cdot\cdot [j..] \to Lb \cdot\cdot [..], \quad L \cdot\cdot [] \to \varepsilon \cdot\cdot []. \end{aligned}$$

For instance,

$$\begin{aligned} S[] &\Rightarrow AS[] \Rightarrow aS[i] \Rightarrow aBS[i] \Rightarrow abS[ji] \Rightarrow abRS[ji] \\ &\Rightarrow abRBS[i] \Rightarrow abRABS[] \Rightarrow abABS[] \Rightarrow abaBS[i] \\ &\Rightarrow ababS[ji] \Rightarrow ababL[ji] \Rightarrow ababLb[i] \Rightarrow ababLab[] \Rightarrow abababab[] \end{aligned}$$

Therefore, $L(G_{\text{copy}}) = \{ww^+ | \ w \in \{a,b\}^*\}$. $L(G_{\text{copy}})$ is context-sensitive, but not context-free as can be seen by the pumping lemma 1.27. The copy language $L(G_{\text{copy}})$ is relevant for coordination in natural languages. $\diamondsuit$

Example 1.42. *(Bach language)* Let $G_{\text{Bach}} = (\{S, D, F, L\}, \{a, b, c\}, \{i, j, k, l, m, n\}, P, S)$ with the productions

$$\begin{aligned} S &\to FS \mid DS \mid LS \mid \varepsilon, \\ F \cdot\cdot [..] &\to c \cdot\cdot [i..], \quad F \cdot\cdot [..] \to b \cdot\cdot [j..], \quad F \cdot\cdot [..] \to a \cdot\cdot [k..], \\ D \cdot\cdot [..] &\to aSb \cdot\cdot [l..] \mid bSa \cdot\cdot [l..], \quad D \cdot\cdot [..] \to aSc \cdot\cdot [m..] \mid cSa \cdot\cdot [m..], \\ D \cdot\cdot [..] &\to bSc \cdot\cdot [n..] \mid cSb \cdot\cdot [n..], \\ D \cdot\cdot [i..] &\to aSb \cdot\cdot [..] \mid bSa \cdot\cdot [..], \quad D \cdot\cdot [j..] \to aSc \cdot\cdot [..] \mid cSa \cdot\cdot [..], \\ D \cdot\cdot [k..] &\to bSc \cdot\cdot [..] \mid cSb \cdot\cdot [..], \\ L \cdot\cdot [l..] &\to c[..], \quad L \cdot\cdot [m..] \to b[..], \quad L \cdot\cdot [n..] \to a[..], \end{aligned}$$

For instance,

$$\begin{aligned}&S \Rightarrow FS \Rightarrow FDS \Rightarrow FDLS \Rightarrow FDL \Rightarrow cDL[i]\\ &\quad \Rightarrow caScL[mi] \Rightarrow caScb[i] \Rightarrow caDScb[i] \Rightarrow caaSbScb \Rightarrow caabcb\end{aligned}$$

In total we have $L(G_{\text{Bach}}) = \{w \mid |w|_a = |w|_b = |w|_c\}$. (Here $|w|_a$ denotes the number of symbols a in the word w.) This is the *Bach language,* or *MIX language,* which is conjectured to be *not* an indexed language [3, §1.2.2]. $\Diamond$

Example 1.43. *(Dependent branches)* [3, §1.2.2] Let $G_{\text{sum}} = (\{S, R, F, L\}, \{a, b, c, d, e, f\}, \{i\}, P, S)$ with the productions

$$\begin{aligned}&S{\cdot\cdot}[..] \to aSf{\cdot\cdot}[i..] \mid R{\cdot\cdot}[..], \quad R \to FL \mid F \mid L,\\ &F{\cdot\cdot}[i..] \to bFc{\cdot\cdot}[..] \mid bc{\cdot\cdot}[..], \quad L{\cdot\cdot}[i..] \to dLe{\cdot\cdot}[..] \mid de{\cdot\cdot}[..],\end{aligned}$$

For instance,

$$\begin{aligned}&S \Rightarrow aSf[i] \Rightarrow aaSff[ii] \Rightarrow aaRff[ii] \Rightarrow aaFLff[ii]\\ &\quad \Rightarrow aabcLff[i] \Rightarrow aabcdeff\end{aligned}$$

Therefore, $L(G_{\text{sum}}) = \{a^n b^m c^m d^l e^l f^n \mid n = m + l \geqq 1\}$. This language cannot be generated by a linear indexed grammar, as introduced in Definition 1.44. $\Diamond$

Linear Indexed Languages

Definition 1.44. A *linear indexed grammar* is a five-tuple (V, Σ, I, P, S) where V is a set of variables, Σ a set of terminals, I a set of indices, $S \in V$ a start variable, and P a set of productions of the form

$$A[..] \to \alpha B[..]\gamma, \qquad A[i..] \to \alpha B[..]\gamma, \qquad A[..] \to \alpha B[i..]\gamma, \tag{1.48}$$

where $A, B \in V$, $\alpha, \gamma \in (V \cup \Sigma)^*$, $i \in I$, and $.. \in I^*$. $\lhd$

Linear indexed grammars are therefore indexed grammars with the additional constraint in the productions that the stack of indices can be transmitted only to at most one variable. As a consequence they are "semilinear."

Example 1.45. [3, §1.2.1] Let $G_{wcw} := \{\{S, R\}, \{a, b, c\}, \{i, j\}, P, S\}$ with the productions P given by

$$\begin{aligned}&S[..] \to aS[i..], \quad S[..] \to bS[j..], \quad S[..] \to cR[..],\\ &R[i..] \to R[..]a, \quad R[j..] \to R[..]b, \quad R[] \to \varepsilon.\end{aligned}$$

Then $L(G_{wcw}) = \{wcw \mid w \in \{a, b\}^*\}$. $\Diamond$

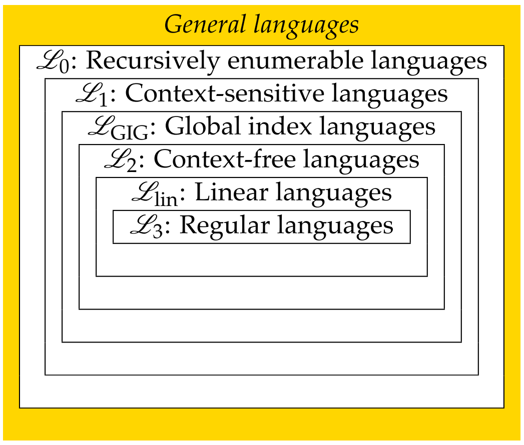

General languages
$\mathscr{L}_0$: Recursively enumerable languages
$\mathscr{L}_1$: Context-sensitive languages
$\mathscr{L}_{\mathrm{GIG}}$: Global index languages
$\mathscr{L}_2$: Context-free languages
$\mathscr{L}_{\mathrm{lin}}$: Linear languages
$\mathscr{L}_3$: Regular languages

Figure 1.2: Hierarchy of languages, including the Chomsky hierarchy Figure 1.1.

1.5.2 The extended language hierarchy

The linear and the indexed languages are included in the Chomsky hierarchy by the language hierarchy in Figure 1.2. Example 1.42 shows that the Bach language is a global index language, and since it is believed that it is not an indexed language it is suggested that global index languages and indexed language do neither contain each other. To date, however, this has not been proved. Both language classes are context-sensitive and include context-free languages. The only known fact is that by 1.43 both language classes contain the class of linear indexed languages.

[4] proved the fact that global index languages are context-sensitive by constructing the respective automaton LR-2PDA accepting global index languages and by implementing it as a linear bounded automaton with the stack naturally represented by the linear bounded tape, see §1.6.

1.6 Languages and machines

There is a close relationship between formal languages and finite automata which "accept" them. In particular, an automaton corresponding to a certain language $\mathscr{L}$ directly solves the *word problem* of $\mathscr{L}$, i.e., decides whether a given word $w \in \Sigma^*$ formed by the underlying alphabet satisfies $w \in \mathscr{L}$. Especially, a machine accepting a (usually context-free) programming language is called a *parser* [19, p. 99]. In this section we will briefly look at the big picture.

Definition 1.46. A *deterministic finite state machine*, often shortly called *machine* or *automaton*, is a five-tuple $(S, \Sigma, \delta, E, s_0)$ consisting of

- a finite set of states S,
- a finite input alphabet Σ,
- a state transition function $\delta : S \times \Sigma \to S$,
- a set of final states $E \subseteq S$,

- an initial state $s_0 \in S$.

For a given input word $w_0 w_1 \dots w_n \in \Sigma^*$ and two states $s, s' \in S$ we write

$$(s, w_0 w_1 \dots w_n) \to (s', w_1 \dots w_n) \tag{1.49}$$

if $s' = \delta(s, w_0)$. $\lhd$

Initially, any automaton is in its initial state s_0. If it gets the input word $w = w_0 w_1 \dots w_n \in \Sigma^*$, it runs through the states $s_0, s_1, \dots, s_{n+1}$ with $s_{i+1} = \delta(s_i, w_i)$. Accordingly, an automaton can be represented by its state diagram where each state is represented by a circle and each state transition $s_i \to s_{i+1}$ by an arrow indicated by w_i and directing from s_i to s_{i+1}, see Figure 1.3. From a given state

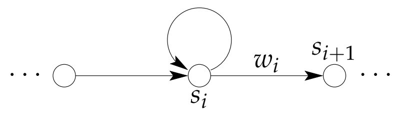


Figure 1.3: State diagram of a finite-state automaton.

there can start several arrows to different states and even back to itself.

Definition 1.47. An automaton $A = (S, \Sigma, \delta, E, s_0)$ is said to *accept* an input word if the final state induced by it is an element of the set E of final states, i.e., if for $w = w_0 w_1 \dots w_n \in \Sigma^*$ we have $s_{n+1} = \delta(s_n, w_n) \in E$. The language $L(A)$ which is *accepted by A* is given by

$$L(A) = \{w_0 \dots w_n \in \Sigma^* | \, \delta(\dots(\delta(s_0, w_0), w_1), \dots), w_n) \in E\},$$

or equivalently

$$L(A) = \{w \in \Sigma^* | \, \exists s_f \in E \text{ with } (s_0, w) \to (s_f, \varepsilon)\}, \tag{1.50}$$

i.e., $L(A)$ contains all the words for which the final state is permissible. $\lhd$

Definition 1.48. A *pushdown automaton* is a five-tuple $(S, \Sigma, \Gamma, \delta, s_0)$ consisting of

- a finite set of states S,
- a finite input alphabet Σ with $\varepsilon \notin \Sigma$,
- a finite *stack alphabet* Γ with $\bot \in \Gamma$,
- a transition relation $\delta : S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathscr{P}(S \times \Gamma^*)$,
- an initial state $s_0 \in S$.

Here $\mathscr{P}(A)$ for an arbitrary set A denotes the *power set* of A, i.e., the set of all possible subsets of A. In particular, $\emptyset$, $A \in \mathscr{P}(A)$, and $|\mathscr{P}(A)| = 2^{|A|}$, [9, 4.19]. For $\gamma \in \Gamma$, the words $w \in \Sigma^*$ and $\kappa \in \Gamma^*$ and two states $s, s' \in S$ we write

$$(s, w, \gamma\kappa) \to (s', w, \kappa' a \kappa) \tag{1.51}$$

if $(s', \kappa') \in \delta(s, \varepsilon, \gamma)$, and

$$(s, \sigma w, \gamma\kappa) \to (s', w, \kappa' a \kappa) \tag{1.52}$$

if $(s', \kappa') \in \delta(s, \sigma, \gamma)$ with $\sigma \in \Sigma$. $\lhd$

Definition 1.49. A pushdown automaton $A = (S, \Sigma, \Gamma, \delta, s_0)$ is said to *accept* an input word if

$$L(A) = \{w \in \Sigma^* \mid (s_0, w, \perp) \to (s, \varepsilon, \varepsilon)\}, \tag{1.53}$$

i.e., $L(A)$ contains all the words for which the final state leaves an empty stack. $\lhd$

Although the definitions of the accepted language of a deterministic machine (1.50) and a pushdown automaton (1.53) formally resemble to one another, the final state in a pushdown automaton is *not* uniquely determined.

Definition 1.50. [4] A *LR-2PDA*, or Left-right two stack pushdown automaton, is a five-tuple $(S, \Sigma, \Gamma, \delta, s_0)$ consisting of

- a finite set of states S,
- a finite input alphabet Σ with $\varepsilon \notin \Sigma$,
- a finite *stack alphabet* Γ with $\perp \in \Gamma$,
- a transition relation $\delta : S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times \Gamma \to \mathscr{P}(S \times \Gamma^* \times \Gamma^*)$,
- an initial state $s_0 \in S$.

The only difference to a pushdown automaton in Definition 1.48 above is therefore the existence of a second stack in the domain and range of the transition function δ. $\lhd$

Definition 1.51. [1, p. 12], [10, pp 267, 274], [19, pp 140, 193] A *(deterministic) Turing machine* M is a seven-tuple $(S, \Sigma, \Gamma, \delta, s_0, \sqcup, E)$ consisting of

- a finite set S of possible states (in which M's "register" can be),
- a finite input alphabet Σ,
- a *tape alphabet* $\Gamma \supseteq \Sigma$ containing a blank symbol $\sqcup \notin \Sigma$,
- a transition function $\delta : S \times \Gamma \to S \times \Gamma \times \{\leftarrow, \rightarrow\}$ describing the rules the machine M uses in performing each step of M.
- an initial state $s_0 \in S$,
- a set of final states $E \subseteq S$.

There are two kinds of transitions, the right translation $\delta(s, \sigma) = (s', \sigma', \rightarrow)$, which we write as

$$(v\rho, s, \sigma w) \to \begin{cases} (v\rho\sigma', s', w) & \text{if } w \neq \varepsilon, \\ (v\rho\sigma', s', \sqcup) & \text{if } w = \varepsilon, \end{cases} \tag{1.54}$$

and the left translation $\delta(s, \sigma) = (s', \sigma', \leftarrow)$, written as

$$(v\rho, s, \sigma w) \to \begin{cases} (v, s', \rho\sigma' w) & \text{if } v \neq \varepsilon, \\ (\sqcup, s', \rho\sigma') & \text{if } v = \varepsilon, \end{cases} \tag{1.55}$$

with the symbols ρ, $\sigma \in \Gamma$, the words v, $w \in \Gamma^+$, and the state $s \in S$. (Note the different meanings of the arrows "$\rightarrow$" as a control symbol and as state transition sign.)

A *nondeterministic Turing machine* M is a seven-tuple $(S, \Sigma, \Gamma, \delta, s_0, \llcorner\!\lrcorner, E)$ consisting of the same elements as a determistic Turing machine, but with a transition relation

$$\delta : S \times \Gamma \rightarrow \mathscr{P}(S \times \Gamma \times \{\leftarrow, \rightarrow\}). \tag{1.56}$$

A Turing machine is called *linear bounded*, or shortly *linear bounded automaton*, if only the symbols of the input word are changed, such that any output sequence cannot be longer than the input sequence. ◁

Theorem 1.52. *Any language accepted by a deterministic finite automaton is regular, and for each regular language there exists a finite automaton accepting it.*

Proof. [10, §5.4] □

With this theorem, the proof idea of the pumping lemma 1.18 for regular languages can be beautifully illustrated by the state diagram of the machine [19, Thm. 1.70], see Figure 1.4. A word thus can be pumped if it induces a closed loop

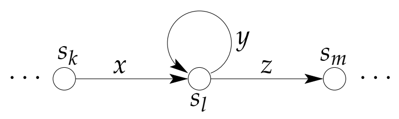


Figure 1.4: The effects of the words x, y, and z in the pumping lemma 1.18 above.

in the state diagram of the machine, which must necessarily exist if the length of xy is longer than the number of states. Moreover, the word problem of a regular language is simply solved by applying the corresponding finite-state automaton to the input word $w \in \Sigma^*$ and looking if it is accepted [10, §5.4.2]. The running time of this algorithm is linear with respect to the word length n, i.e., it is $O(n)$.

Theorem 1.53. *Any language accepted by a pushdown automaton is context-free, and for each context-free language there exists a pushdown automaton accepting it.*

Proof. [10, §5.5.2] □

The word problem of context-free languages is solved by the CYK which has running polynomial time $O(n^3)$ with respect to the input word length n [10, §4.4.4].

Theorem 1.54. *Any language accepted by a LR-2PDA is a global index language, and for each global index language there exists a LR-2PDA accepting it.*

Proof. [2] proves that a modified version of the Earley algorithm □

The word problem of global index languages is solved by Castaño's modified Earley algorithm which has running polynomial time $O(n^6)$ with respect to the input word length n [2, p. 33].

Theorem 1.55. *Any language accepted by a nondeterministic linear bounded Turing machine is context-sensitive, and for each context-sensitive language there exists a nondeterministic linear bounded Turing machine accepting it.*

Proof. [10, Satz 6.15] □

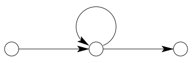
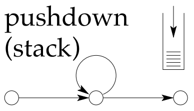
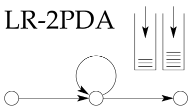
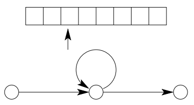
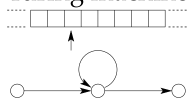
It is a still open question whether any context-sensitive language is accepted by a *deterministic* linear bounded Turing machine [12, p. 239]. This question is known as the "first LBA problem" initially posed by [15], cf. [10, p. 302].[2]

Theorem 1.56. *Any language accepted by a Turing machine is recursively enumerable, and for each recursively enumerable language there exists a Turing machine accepting it.*

Proof. [10, Satz 6.14] □

In summary, the relationships of languages, grammars, automata and word problems are given in Table 1.1. The language classes in the left column of this table consist of exactly those languages that are accepted by the automata and generated by the grammars in the same line. Each line contains all of those above it. The right hand column shows the computational complexity of solving the word problem, i.e., of recognizing whether a given word is an element of the respective language. These relationships have fundamental importance in computer science.

Table 1.1: Languages and automata. Modified from [18]

Language	Automaton	Grammar	Recognition $w \in \mathscr{L}$?
regular	finite-state 	regular $A \to xA$	linear
context-free	pushdown (stack) 	context-free $S \to xSy$	polynomial (CYK, $O(n^3)$)
global index	LR-2PDA 	global index $S[i] \to xSy[ji]$	polynomial ($O(n^6)$)
context-sensitive	linear bounded 	context-sensitive $Ax \to yA$	exponential (?)
recursively enumerable	Turing machine 	unrestricted $Bxx \to A$	undecidable

In general, we can identify problems with languages, words with solution candidates, and algorithms with Turing machines [17, p. 59]. Then the classifications of algorithmic problems correspond to the one of grammars. In particular, the set $\mathfrak{L}_{\text{reg}}$ of all regular languages is precisely $\mathbf{SPACE}(n)$ [17, p. 55], the set $\mathfrak{L}_{\text{cf}}$ of all context-free languages is in $\mathbf{P}$, [17, p. 67], and the set $\mathfrak{L}_{\text{cs}}$ of all context-sensitive languages is precisely $\mathbf{NSPACE}(n)$ [17, p. 67], i.e.,

$$\mathfrak{L}_{\text{reg}} = \mathbf{SPACE}(n), \qquad \mathfrak{L}_{\text{cf}} \subset \mathbf{P}, \qquad \mathfrak{L}_{\text{cs}} = \mathbf{NSPACE}(n). \tag{1.57}$$

[2] Kuroda's "second LBA problem" has been solved in 1987 by the Immerman–Szelepcsényi Theorem [1, §4.3.2], [17, Thm. 7.6].

Here a language L is in the space complexity class $\mathbf{SPACE}(n)$ if there is a Turing machine that decides L and operates within space bound n [17, p. 35], in the time complexity class $\mathbf{P}$ if there is a Turing machine that decides L and operates within time bound n^k for some $k \in \mathbb{N}$, and in the space complexity class $\mathbf{NSPACE}(n)$ if there is a nondeterministic Turing machine that decides L and operates within space bound n [17, p. 141], see also [Zoo]. According to [17, Thm. 7.4 and p. 142],

$$\mathbf{SPACE}(n) \subset \mathbf{P} \qquad \text{and} \qquad \mathbf{NSPACE}(n) \subset \mathbf{NP}. \tag{1.58}$$

Moreover, $\mathbf{SPACE}(n) \neq \mathbf{NP}$ [17, p. 155]. The word problem for a context-sensitive language is probably solvable only by complete enumeration, i.e., by non-polynomial algorithms. If, however, we had $\mathbf{P} = \mathbf{NP}$, then the word problem would be solvable in polynomial time, too, and there would be no essential difference between context-free and context-sensitive languages from the perspective of computational complexity. This indeed would be surprising.

Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt.

L. Wittgenstein (1918), *Tractatus Logico-Philosophicus*, 5.6[3]

[3] *http://books.google.de/books?id=BraohBA1avIC&pg=PA118* [2012-03-17]: The limits of my language are the limits of my world.

Appendix A

Mathematical Foundations

A.1 Notation

Mathematical and logical symbols

$:=$	is defined as
$\emptyset$	the empty set, $\emptyset = \{\}$
$\forall$	for all
$\exists$	there exists
$\exists_1$	there exists exactly one
$\Rightarrow$	implies, only if
$\Leftarrow$	follows from, if
$\Leftrightarrow$	if and only if

Number sets

$\mathbb{N}$	the natural numbers, $\mathbb{N} = \{1, 2, 3, \dots\}$
$\mathbb{N}_0$	the natural numbers with zero, $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$
$\mathbb{Z}$	the integers, $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, 3, \dots\}$
$\mathbb{Z}^\times$	the nonvanishing integers, $\mathbb{Z}^\times = \mathbb{Z} \setminus \{0\}$
$\mathbb{Q}$	the rational numbers, $\mathbb{Q} = \{\frac{m}{n} \mid m \in \mathbb{Z},\ n \in \mathbb{N}\}$
$\mathbb{Q}^\times$	the nonvanishing rational numbers, $\mathbb{Q}^\times = \mathbb{Q} \setminus \{0\}$
$\mathbb{R}$	the real numbers, $\mathbb{R} = (-\infty, +\infty)$
$\mathbb{R}^\times$	the nonvanishing real numbers, $\mathbb{R}^\times = \mathbb{R} \setminus \{0\}$

A.2 Sets

The fundamental notion of mathematics is the set. Intuitive as at seems at first sight, this notion is rather subtle. It was a recognized in the beginning of the 20th century that a collection of objects need not be a set. A set is a mathematical concept which obeys strict axioms, the words "collection", "class", "family", or "system" of objects (which may also be sets) are often loosely used.

In mathematical logic, the notions "set" and "element x of a set S," denoted $x \in S$, are frequently used obeying the eight axioms of Zermelo-Fraenkel [6, §VII.3], [16, §3], [Ra, §2], [21, §4.4.3]:

(i) *Axiom of extensionality:* Two sets are equal if and only if they have the same elements.

(ii) *Axiom of separation:* For every set S and each property $\mathscr{A}$ there exists a set T of those elements of S with the property $\mathscr{A}$, written symbolically

$$T = \{x \in S \mid x \text{ has the property } \mathscr{A}\}. \tag{A.1}$$

(iii) *Axiom of pairing:* For every two sets S and T there exists the set $\{S, T\}$.

(iv) *Axiom of union:* For every set S the union $\bigcup S$ of sets in S is a set.

(v) *Axiom of power set:* For every set S there exists the power set $\mathscr{P}(S)$ of S, written $\mathscr{P}(S) = \{A \mid A \subset S\}$.

(vi) *Axiom of infinity:* There exists a set which contains the infinitely many sets 0 $:= \emptyset$, $1 := \{\emptyset\}$, $2 := \{\emptyset, \{\emptyset\}\}$, $3 := \{\emptyset, \{\emptyset, \{\emptyset\}\}\}$, ... , or in other words, $1 = \{0\}$, $2 = \{0, 1\}$, and in general $n = \{0, 1, ..., n - 1\}$.

(vii) *Axiom of replacement:* If the expression $\varphi(x, y, z_1, \dots, z_n)$ for chosen parameters z_1, ..., z_n defines an association $x \mapsto y = f(x)$, then the image $f(A)$ of a set is a set.

(viii) *Axiom of choice:* For a given finite or infinite index set I and every family $(S_i)_{i \in I}$ of nonempty sets S_i there exists a family $(x_i)_{i \in I}$ of elements $x_i \in S_i$ for every $i \in I$.

This system of axioms is also called *ZFC*, for Zermelo-Fraenkel with axiom of choice. The axiom of choice is the most famous of the ZFC axioms. There are a series of equivalent axioms, for instance Zorn's lemma or Zermelo's well-ordering theorem [16, §8.8], [20, p 5]. In contrast to the other axioms it does not tell constructively when general classes can be regarded as sets, but represents a mere existence postulate: It offers no recipe *how* an element of each set of a system of nonempty sets could be chosen.

Historically perhaps most important is Axiom (ii), the axiom of separation, which excludes Russell's antinomy. In 1901 Russel communicated this paradox to Frege, the leading mathematical logician of the period.[1] It shattered Cantor's (nowadays called naive) set theory. Since Frege at this time was going to publish his second edition of *Grundgesetze der Arithmetik*, which finally was released in 1903 and which heavily based on Cantor's set theory, he draw the bitter conclusion in the epilog:

> *Einem wissenschaftlichen Schriftsteller kann kaum etwas Unerwünschteres begegnen, als daß ihm nach Vollendung einer Arbeit eine der Grundlagen seines Baues erschüttert wird. In diese Lage wurde ich durch einen Brief des Herrn Bertrand Russell versetzt, als der Druck dieses Bandes sich seinem Ende näherte.*

[1] In 1879, Gottlob Frege (1848 – 1925) published his *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens* ("concept notation, a formula language, modelled on that of arithmetic, of pure thought"). The Begriffsschrift was arguably the most important publication in logic since Aristotle founded the subject more than 2000 years before.

Zermelo, who had discovered the paradox even a year before Russell but did not publish it, solved the problem in 1908 by introducing a set theory basing on the axiom of separation.

Theorem A.1 (Exclusion of Russell's antinomy). *There exists no set of all sets. (Or equivalently: The class of all sets is not a set.)*

Proof. Assume there exists a set M of all sets. By the axiom of separation, Axiom (ii), the class $R = \{x \in M \mid x \notin x\}$ then is a set. Hence there are two possibilities: (i) If $R \notin R$, then $R \in R$, by construction of R. (ii) If $R \in R$, then $R \notin R$ by construction of R. Hence we get the contradiction $R \in R \iff R \notin R$, i.e., the assumption must be wrong. □

Theorem A.2. *Given two sets A and B, for $x \in A$ and $y \in B$ the "Kuratowski pair" $(x,y) := \{\{x\},\{x,y\}\}$, also called the "ordered pair" (x,y), satisfies*

$$(x,y) = (x',y') \iff x = x' \text{ and } y = y', \tag{A.2}$$

and the class

$$A \times B := \{(x,y) \mid x \in A \text{ and } y \in B\} \tag{A.3}$$

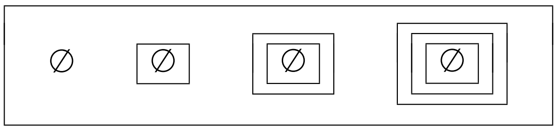
is a set, called the Cartesian product *of A and B.*

Proof. The property (A.2) of ordered pairs follows directly from the ZFC axioms (i) and (iii), the second assertion follows from axioms (ii) and (v). For details see [16, §4.3]. □

Why do we need the axiom of infinity? For instance, the set

$$4 = \{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\{\{\emptyset\}\}\}\}$$

can be represented as boxes in boxes:



Each element of the set leaves us in a situation resembling the birthday child who receives the "disappointing gift" [16, p 180], namely a huge box which is opened to exhibit another box, which can be opened again to show another box in which is another box etc., to eventually present — an empty set.

Up to the end of the 19th century it was commonplace belief among philosophers and mathematicians that the existence of infinite sets could be proved. In particular, one thought that the set of natural numbers could be constructed by logic. We know better now. *"Logic can codify the valid forms of reasoning but it cannot prove the existence of anything"* [16, p 27]. Zermelo was the first to recognize that the axiom of infinity has to be postulated to construct the set of natural numbers.

The axiom system of Zermelo-Fraenkel ZFC works well, that is, virtually all results in current mathematics can be derived by it.[2] However, there remains an uncomfortable property of the system to allow the set

$$\Omega = \{\Omega\}. \tag{A.4}$$

[2] It is a still open question, though, whether ZFC is logically consistent at all [6, pp 120].

We can think of Ω as the "ultimately frustrating gift" [16, p 181] since denoting Ω by a box $\Box$, each box has exactly one box inside, identical with the one just opened, and you can keep opening the boxes without ever finding *anything* …

$$\boxed{\boxed{\boxed{\boxed{\;\cdots\;}}}}$$

Not even an empty set. If such phenomena are desired to be banned, it is common to use the following additional axiom proposed by von Neumann in 1925:

(ix) *Axiom of Regularity:* Every non-empty set S contains an element x which is disjoint from S.

ZFC with the axiom of regularity thus forbids the set $\Omega = \{\Omega\}$, and hence in set theories based on this extension there exists no ultimately frustrating gift.

A.3 Maps

Definition A.3. Let A, B be two sets. A *map*, or a *function*, from A to B, written

$$f : A \to B, \tag{A.5}$$

denotes an association of any element $a \in A$ to an element $b = f(a)$ in B, also written $a \mapsto b$. Then A is called the *domain* of f, and B the *codomain* or *target* of f. For any subset $A' \subset A$ of A, the set

$$f(A') := \{f(a) |\, a \in A'\} \subset B \tag{A.6}$$

is called the *image* of A' under f. In particular, $f(A)$ is shortly called the image of f. For a subset $B' \subset B$ of B, the set

$$f^{-1}(B') := \{a \in A |\, f(a) \in B'\} \subset A \tag{A.7}$$

is called the *preimage* of B' under f. In particular, $f^{-1}(B)$ is shortly called the preimage of f. The map $f : A \to B$ is called …

- *injective* if $f(a) = f(a')$ implies $a = a'$ for all a, $a' \in A$, or equivalently, $|f^{-1}(\{b\})| = 1$ for all $b \in f(A)$;
- *surjective* if $f(A) = B$;
- *bijective* if it is both injective and surjective.

Two maps f: $A \to B$ and g: $C \to D$ are *equal*, if $A = C$, $B = D$, and $f(x) = g(x)$ for all $x \in A$. If for two maps A and B there exists a bijective map, we say that A and B have the same *cardinality* and denote $A \sim B$. $\lhd$

Remark A.4. In set theory, a function is considered as a special binary relation. A *binary relation* R on two sets A and B is a subset $R \subset A \times B$. For instance, let be A the set of parents and B the set of children, given by

$$A = \{\text{Homer, Marge}\}, \qquad B = \{\text{Bart, Lisa, Maggie}\}.$$

Then the "Simpsons relation" $R =$ "is mother of" on A and B implies

$$(\text{Marge, Bart}) \in R, \qquad \text{but} \qquad (\text{Homer, Lisa}) \notin R.$$

A special class of binary relations are those $f \subset A \times B$ assigning to each element of A some element of B, formally

$$\forall x \in A \;\; \exists y \in B \text{ such that } (x, y) \in R. \tag{A.8}$$

Such a relation is called a *multivalued function*, written $f\colon\ A \to \mathscr{P}(B)$. Then a function $f \subset A \times B$ is a special relation assigning to each element of A *exactly one* element of B [16, §4.16],

$$\forall x \in A \;\; \exists_1 y \in B \text{ such that } (x, y) \in R. \tag{A.9}$$

Note that the above defined "Simpsons relation" on A and B is neither a multivalued function nor a function. $\Diamond$

Remark A.5. From the view point of algorithmics, the *computation* of a map is important. In set theory the functions $f, g : \mathbb{R}^2 \to \mathbb{R}$,

$$f(x, y) = (x + y)^2 \qquad g(x, y) = x^2 + 2xy + y^2 \tag{A.10}$$

are equal, but their computation is rather different. Algorithmically, f is more efficient than g (since there are only two arithmetical operations to compute f, but six for g). Whether the intuitive notion of "function as a computation" can be represented faithfully in set theory is an unsolved problem [16, p 41]. $\Diamond$

Example A.6. *(Bijection between* $\mathbb{N}$ *and* $\mathbb{Z}$*)* Define the function $f : \mathbb{N} \to \mathbb{Z}$,

$$f(n) = (-1)^n \, \frac{2n-1}{4} + \frac{1}{4}. \tag{A.11}$$

Then for all $n \in \mathbb{N}$ we have $f(2n) = n$ and $f(2n-1) = 1 - n$. In particular, this implies immediately that $f(m) \neq f(n)$ if $m \neq n$ for $m, n \in \mathbb{N}$, i.e., f is injective. Moreover, f is surjective since for any $y \in \mathbb{Z}$ there exists an element $x \in \mathbb{N}$ such that $f(x) = y$, namely $x = 2y$ if $y > 0$ and $x = 1 - 2y$ if $y \leqq 0$. Thus, f is a bijection and $\mathbb{N} \sim \mathbb{Z}$, cf. [8, p 305]. $\Diamond$

Remark A.7. In a proof that two general sets A and B have the same cardinality, it is usually a tedious task to construct a surjective map. By the Cantor-Bernstein Theorem, often also called Schröder-Bernstein Theorem, it suffices to find two injective maps $f : A \to B$ and $g : B \to A$ [8, §3.5.1]. $\Diamond$

Theorem A.8 (Pigeonhole Principle). *Every map $f : A \to A$ on a finite set A into itself is injective if and only if it is surjective.*

Proof. The "only if" part ("f injective $\Rightarrow f(A) = A$") is shown in [16, 5.25]. Let $I_n = \{0, 1, ..., n-1\}$. The proof bases on the property (A.14) of a map $g : I_n \to I_n$, since A is finite, there exists a bijective permutation $\pi : A \to I_n$, and we can define $g : I_n \to I_n$ by the equation

$$g(i) = \pi(f(\pi^{-1}(i))) \tag{A.12}$$

for $i < n$ so that

$$f(x) = \pi^{-1}(g(\pi(x))) \tag{A.13}$$

for all $x \in A$. Now if f is injective then also g is injective, as a composition of injections (A.12); but then by (A.14) g is bijective, and hence f as a compositions of bijections is bijective. By the same reasoning, if f is surjective, g is surjective as a compositions of surjections, i.e. injective by (A.14), and thus f is injective as a composition of injections. □

Lemma A.9. *Let $I_n = \{0, 1, ..., n-1\}$. For all maps $g : I_n \to I_n$ and an arbitrary natural number n we have*

$$g(x) \neq g(y) \quad \forall x \neq y \quad \iff \quad g(I_n) = I_n, \tag{A.14}$$

with $x \in y \in I_n$.

Proof. The proof is by induction over n. For $n = 0$ we have only one map $g : \emptyset \to \emptyset$, and for $n = 1$ only one function $g : \{1\} \to \{1\}$, and both are bijective. For the induction step $n \to n+1$ we assume (A.14) for *all* maps $h : I_n \to I_n$. Since $I_{n+1} = I_n \cup \{n\}$, for any map $g : I_{n+1} \to I_{n+1}$ and its restriction $h := g|_{I_n}$, defined by $h(k) = g(k)$ for $0 \leqq k < n$, we have one of the following three cases.

Case (1): $g(n) = n$. Then g is clearly injective if and only if h is injective, and g is surjective if and only if h is surjective.

Case (2): $n \notin g(I_{n+1})$. This means that g is not surjective, and since $g(n) = h(k) = g(k)$ for some k, i.e., g then is also not injective. On the other hand, the assumption that g is injective implies that $\nexists k \in [0, k)$ with $h(k) = g(n) \in [0, n)$, i.e., h is not injective, hence g is not injective, which is a contradiction to the assumption.

Case (3): There exist numbers $u, v < n$ such that $g(u) = n$, $g(n) = v$. Defining $h' : I_n \to I_n$,

$$h'(k) = \begin{cases} h(k) & \text{if } k < n \text{ and } k \neq u, \\ v & \text{if } k = u. \end{cases}, \tag{A.15}$$

i.e., a function h' agreeing with g at all arguments except u, we see that h' is injective if and only if g is injective, and that h' is surjective if and only if g is surjective. □

Remark A.10. Theorem A.8 is equivalently stated as [7, p 130]: *If n pigeons are put into n pigeonholes, then there is an empty pigeonhole if and only if there is a hole with more than one pigeon.* Here an empty pigeonhole means "not surjective," and a hole with more pigeons means "not injective." It implies the pigeonhole principle in its usual formulation refering to Dirichlet's *Schubfachprinzip* ("box principle") [8, §2.4]: *If more than n objects are distributed over n containers, then some container must contain more than one object.* Here "contain more than one object" means "is mapped injectively." ◊

Remark A.11. *(Hilbert's Hotel)* Theorem A.8 cannot be generalized for an infinite set A. For instance, the map $f : \mathbb{N} \to \mathbb{N}$,

$$f(n) = n + 1 \tag{A.16}$$

is injective, but not surjective. The fact that the pigeonhole principle is not valid for infinite sets is the reason for the possibility of Hilbert's Hotel, a hotel with infinitely many rooms completely numbered by 1, 2, ..., all of which are occupied by a guest. However, if a new guest arrives and wishes to be accommodated in the hotel, each hotel guest can be moved from his room number n to the next room with number $n + 1$. Thus it is possible to make room for any finite number of new guests. ◊

A.4 Algebra

A main subject of abstract algebra is group theory. It studies possible binary operations on elements of a set, often called *structures*, which in the end generalize the addition and multiplication operations of numbers. This section gives a brief overview to the basic algebraic notions needed in our considerations here. For more details see any standard textbook on algebra, e.g., [13] or [14]. However, differing from most standard treatments the emphasis here is laid on properties of semigroups and monoids which, in the theory of formal languages, play a more important role than groups.

A.4.1 Semigroups, monoids, and groups

Definition A.12. A *semigroup* $(H, \circ)$ is a nonempty set H with the binary operation $\circ : H \times H \to H$ obeying the *law of associativity*

$$(x \circ y) \circ z = x \circ (y \circ z) \tag{A.17}$$

for all $x, y, z \in H$. If all elements $x, y \in H$ satisfy the *law of commutativity*

$$x \circ y = y \circ x, \tag{A.18}$$

then H is called *Abelian* or *commutative*. The number $|H|$ of elements of H is called the *order* of the semigroup. If it is clear from the context, one often speaks of the semigroup H instead of $(H, \circ)$. ◁

Definition A.13. A semigroup $(M, \circ)$ is called a *monoid* if it contains an element $e \in M$ satisfying

$$e \circ x = x \circ e = x \tag{A.19}$$

for all $x \in M$. In this case, e is called *neutral element*. ◁

Lemma A.14. *The neutral element in a monoid is unique.*

Proof. Assume that $e' \in M$ is another neutral element in the monoid. Then by (A.19) we have $e' \circ e = e'$ (since e is neutral), as well as $e' \circ e = e$ (since e' is neutral), i.e., $e' = e$. □

Examples A.15. *(Natural numbers)* The pair $(\mathbb{N}, +)$ is a semigroup but not a monoid, but $(\mathbb{N}_0, +)$ is a monoid with the neutral element 0. Moreover, $(\mathbb{N}, \cdot)$ is a monoid with the neutral element 1. $\Diamond$

Examples A.16. *(Maps)* Let X be a set. Then the set of all maps $X \to X$ is a monoid with respect to composition of maps, with the identity map as neutral element. $\Diamond$

Examples A.17. *(Words)* Let Σ be an alphabet. Then the set Σ^+ of all words over Σ, i.e., all strings of letters from Σ, is a semi-group with respect to concatenation [5, §6.1]. The set Σ^* containing the empty string ε is a monoid with the empty string as neutral element. $\Diamond$

Definition A.18. A *group* $(G, \circ)$ is a nonempty set G with the binary operation $\circ : G \times G \to G$ obeying the three group axioms, namely: the law of associativity

$$(x \circ y) \circ z = x \circ (y \circ z) \tag{A.20}$$

for all $x, y, z \in G$; there exists an element $e \in G$ satisfying

$$e \circ x = x, \tag{A.21}$$

the *neutral element*; for each $x \in G$ there exists an element $x^{-1} \in G$ such that

$$x^{-1} \circ x = e. \tag{A.22}$$

It is called the *inverse* of x (with respect to the operation $\circ$). If all elements $x, y \in G$ satisfy the *law of commutativity*

$$x \circ y = y \circ x, \tag{A.23}$$

then G is called *Abelian* or *commutative*. $\lhd$

Lemma A.19. *Let be G a group and $x \in G$ with the neutral element $e \in G$. Then we have*

$$x \circ e = x, \tag{A.24}$$
$$x \circ x^{-1} = e \tag{A.25}$$

for all $x \in G$. In particular, a group is a monoid. Moreover, for each $x \in G$ the inverse $x^{-1} \in G$ is unique.

Proof. For $x \in G$ there exists an element $y = x^{-1} \in G$ with $y \circ x = e$, and an element $z = y^{-1} \in G$ with $z \circ y = e$. Thus

$$x = e \circ x = (z \circ y) \circ x = z \circ (y \circ x) = z \circ e.$$

Substituting e by $(e \circ e)$ this gives $x = z \circ (e \circ e) = (z \circ e) \circ e = x \circ e$, i.e., (A.24). In addition, $z = z \circ e = x$ and therefore $x \circ y = z \circ y = e$, i.e., (A.25). If $y' \in G$ is another element being an inverse of x, then we have $y' = y' \circ e = y' \circ (x \circ y) = (y' \circ x) \circ y = e \circ y = y$. $\square$

Examples A.20. *(Numbers)* The monoids $(\mathbb{N}_0, +)$ and $(\mathbb{N}, \cdot)$ of Example A.15 are not groups. However, $(\mathbb{Z}, +)$ and $(\mathbb{Q}^\times, \cdot)$ with $\mathbb{Q}^\times = \mathbb{Q} \setminus \{0\}$ are commutative groups, as well as $(\mathbb{R}, +)$ and $(\mathbb{R}^\times, \cdot)$ with $\mathbb{R}^\times = \mathbb{R} \setminus \{0\}$. $(\mathbb{R}, \cdot)$ is an Abelian monoid but not a group, $(\mathbb{R}^\times, +)$ is not even a semigroup, $\Diamond$

Examples A.21. *(Matrices)* For $n \in \mathbb{N}$, let denote $M_n(\mathbb{K})$ the set of all $(n \times n)$ matrices over a field $\mathbb{K}$, for instance $\mathbb{Q}$, $\mathbb{R}$, or $\mathbb{C}$. Then $(M_n(\mathbb{K}), \cdot)$ with the matrix multiplication $\cdot$ and the identity matrix as neutral element. is a monoid, but not a group. However, $(M_n(\mathbb{K}), +)$ with the matrix addition is a commutative group, and the set $GL(n, \mathbb{K})$ of invertible matrices in $M_n(\mathbb{K})$ is a non-commutative group $(GL(n, \mathbb{K}, \cdot)$ with the identity matrix as neutral element. $GL(n, \mathbb{K})$ is called the *general linear group* over $\mathbb{K}$. $\Diamond$

Examples A.22. *(Unit fractions)* Let $F = \{1, \frac{1}{2}, \frac{1}{3}, \ldots\}$ denote the set of unit fractions $\frac{1}{n}$ with $n \in \mathbb{N}$. Then $(F, \cdot)$ and $(F \cup \{0\}, \cdot)$ are monoids, each with neutral element 1. $\Diamond$

Remark A.23. There are two usual ways to write the group operation, the additive notation and the multiplicative notation. In the additive notation we write $x + y$ instead of $x \circ y$, 0 is the neutral element, and the inverse of x is written as $-x$. It usually denotes commutative operations. In the multiplicative notation, xy is written instead of $x \circ y$, 1 is the neutral element, and x^{-1} is the inverse of x. $\Diamond$

Definition A.24. Let G be a semigroup. Then a subsemigroup $H \subset G$ being in G is called *normal* if

$$xH = Hx \qquad \text{for all } x \in G, \tag{A.26}$$

where $xH = \{xh |\ h \in H\}$ and $Hx = \{hx |\ h \in H\}$. Analogously, a submonoid H of a monoid G is called *normal* if (A.26) holds literally, and a subgroup H of a group G is called *normal* if (A.26) holds. $\lhd$

Theorem A.25. *Is H a subgroup of a group G, then the following assertions are equivalent:*

(i) $xH = Hx$ *for all* $x \in G$;

(ii) $x^{-1}Hx = H$ *for all* $x \in G$;

(iii) $x^{-1}hx \in H$ *for all* $x \in G$ *and* $h \in H$.

Proof. The implications (i) ⇒ (ii) ⇒ (iii) are clear. To show (iii) ⇒ (i), let $x \in G$ and $h \in H$ be given; then $xh = (x^{-1})^{-1}hx^{-1}x \in Hx$ and $hx = x(x^{-1}hx) \in xH$. $\square$

Theorem A.26. *All subgroups of an Abelian group are normal.*

Example A.27. *(Matrices)* Let $\mathbb{K}$ be $\mathbb{Q}$, $\mathbb{R}$, or $\mathbb{C}$ (or an arbitrary field), and denote the *special linear group* $SL(n, \mathbb{K})$ the set of all $(n \times n)$ matrices over $\mathbb{K}$ with determinant 1. Then $SL(n, \mathbb{K})$ is a normal subgroup of the general linear group $GL(n, \mathbb{K})$ from Example A.21. $\Diamond$

A.4.2 Homomorphisms

Structure-preserving maps between algebraic structures are called homomorphisms and play an important role in algebra.

Definition A.28. Let G and H be two semigroups. Then a map $f : G \to H$ is called a *homomorphism* if

$$f(xy) = f(x)\,f(y) \tag{A.27}$$

for all x, $y \in G$. The set of all homomorphisms from G to H is denoted by $\mathrm{Hom}(G, H)$. The set

$$\mathrm{im} f := f(G) := \{f(x) |\; x \in G\} \tag{A.28}$$

is called the *image* of f. If G and H are monoids and $e' \in H$ is the neutral element in H, the set

$$\ker f := f^{-1}(\{e'\}) := \{x \in G |\; f(x) = e'\} \tag{A.29}$$

is called the *kernel* of f. ◁

Lemma A.29. *If $f \in \mathrm{Hom}(G, H)$ for two monoids G and H with neutral elements $e \in G$ and $e' \in H$, then $f(e) = e'$.*

Proof. We have $f(e) = f(ee) = f(e)f(e)$, i.e., $f(e) = e'$. □

Theorem A.30. *If $f \in \mathrm{Hom}(G, H)$ for two groups G and H, then*

$$f(x^n) = f(x)^n \tag{A.30}$$

for all $n \in \mathbb{Z}$. If G and H are monoids, the equation holds for all $n \in \mathbb{N}_0$, and if they are semigroups, it holds for all $n \in \mathbb{N}$.

Proof. For $n \in \mathbb{N}$ Equation (A.30) follows by induction from (A.27). In case of monoids, $x^0 = e$ and $f(x^0) = f(x)^0 = e'$ by Lemma A.29. Finally, if G and H are groups, $e' = f(e) = f(xx^{-1}) = f(x)f(x^{-1})$, hence $f(x)^{-1} = f(x^{-1})$. □

Corollary A.31. *Let $f \in \mathrm{Hom}(G, H)$, with G and H specified as follows.*

(i) If G and H are semigroups then $\mathrm{im} f$ *is a subsemigroup in H.*

(ii) If G and H are monoids or groups then $\mathrm{im} f$ *is a submonoid of H,* $\ker f$ *is a normal submonoid of G. $f(\ker f y) = f(yx)$ if x*

(iii) If G and H are groups then $\mathrm{im} f$ *is a subgroup of H and* $\ker f$ *is a normal subgroup of G.*

Proof. The assertions for the image set $\mathrm{im} f$ follow directly from Equation (A.27) and Theorem A.29. In case (ii) and (iii) we have for any $y \in G$, $x \in K := \ker f$ and the neutral element $e' \in H$ that $f(xy) = f(x)f(y) = e'f(y) = f(y)e' = f(y)f(x) = f(yx)$, i.e., $f(Ky) = f(yK)$, and hence $Ky = f^{-1}(f(Ky)) = f^{-1}(f(yK)) = yK$. □

An important special class of homomorphisms of a semigroup G are are those which are bijective, and in particular those mapping G bijectively into G itself.

Definition A.32. For semigroups G and H, a bijective homomorphisms from G to H is called an *isomorphism*. The set of isomorphisms from G to G is given by

$$\mathrm{Aut}(G) := \mathrm{Hom}(G, G) \tag{A.31}$$

and is called the set of *automorphisms* of G. $\triangleleft$

Lemma A.33. *For a semigroup G and the map composition $\circ$: $\mathrm{Aut}(G) \times \mathrm{Aut}(G) \to \mathrm{Aut}(G)$, the pair $(\mathrm{Aut}(G), \circ)$ is a group, with the identity map Id_G as neutral element.*

Proof. By definition of the composition $\circ$ the law of associativity (A.20) is satisfied. For any $f \in \mathrm{Aut}(G)$ and $x \in G$ we have $(\mathrm{Id}_G \circ f)(x) = \mathrm{Id}_G(f(x))) = f(x)$, i.e., $\mathrm{Id}_G \circ f = f$ as required by (A.21). Let $f \in \mathrm{Aut}(G)$ and $x, y \in G$ such that $y = f(x)$. Since $\mathrm{Aut}(G)$ consists of *all* homomorphisms from G to G, there exists one that maps y to x, i.e., $f^{-1} \in \mathrm{Aut}(G)$, satisfying (A.22). $\square$

Definition A.34. Let G be a group and $g \in G$ an invertible element. Then the map

$$C_g : G \to G, \qquad x \mapsto gxg^{-1} \tag{A.32}$$

is called the *conjugation* with g. $\triangleleft$

Lemma A.35. *For a semigroup G and an invertible element $g \in G$, the conjugation C_g with g is an automorphism, i.e., $C_g \in \mathrm{Aut}(G)$.*

Proof. For $x, y \in G$ we have $C_g(xy) = gxyg^1 = (gxg^1)(gyg^1) = C_g(x)\, C_g(y)$, and $C_e = \mathrm{Id}_G$ for the neutral element $e \in G$. Hence C_g is a homomorphism. Since for two elements $x, y \in G$ the equality $gxg^{-1} = gyg^{-1}$ implies $x = y$ after multiplying with g^{-1} from the left and with g from the right, C_g is injective. Since then the order of G is the same then the order of $C_g(G)$, i.e., $|C_g(G)| = |G|$, C_g is also surjective. $\square$

Theorem A.36. *There are only five non-isomorphic semigroups of order two, given by the following multiplication tables ("Cayley tables"):*

$$\begin{array}{cc}
\underbrace{\begin{array}{ccc}
O_2 & LO_2 & RO_2 \\
\begin{array}{c|cc} \cdot_0 & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 0 \end{array} &
\begin{array}{c|cc} \cdot_l & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 1 & 1 \end{array} &
\begin{array}{c|cc} \cdot_r & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 0 & 1 \end{array}
\end{array}}_{\textit{semigroups}} &
\begin{array}{cc}
(\{0,1\},\wedge) & \mathbb{Z}_2 \\
\begin{array}{c|cc} \wedge & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array} &
\begin{array}{c|cc} \oplus & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array} \\
\textit{monoid} & \textit{group}
\end{array}
\end{array} \tag{A.33}$$

Proof. A complete enumeration of all $2^{2^2} = 16$ possible binary operations on the set {0, 1}, i.e., all combinations of filling the Cayley table

$$\begin{array}{c|cc} \circ & 0 & 1 \\ \hline 0 & a_{00} & a_{01} \\ 1 & a_{10} & a_{11} \end{array}$$

with $a_{ij} \in \{0, 1\}$. Regarding isomorphims yields the assertion.[3] O_2, LO_2 and RO_2 each are no monoids since they do not have a neutral element. Instead, in O_2 the element 0 is an "absorbing" element, in LO_2 both 0 and 1 are left-absorbing, and in RO_2 both are right-absorbing. The semigroup $(\{0, 1\}, \wedge)$ is a monoid with the neutral element 1, and $\mathbb{Z}_2 = (\{0, 1\}, \oplus)$ is a group with neutral element 0 and 1 being its own inverse. □

Remark A.37. The semigroup O_2 is called the *null semigroup*, LO_2 the *left zero semigroup*, and RO_2 the *right zero semigroup*. RO_2 and LO_2 are not isomorphic but antiisomorphic, and hence "equivalent." Moreover, $(\{0, 1\}, \wedge)$ is isomorphic to $(\{0, 1\}, \vee)$, but here the neutral element is 0. An isomorphic matrix representation of $(\{0, 1\}, \wedge)$ is $(\{I, A\}, \cdot)$ with

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \tag{A.34}$$

and the operation $\cdot$ being the usual matrix multiplication. In $\mathbb{Z}_2 = (\{0, 1\}, \oplus)$ the binary operation $\oplus$ is a logical XOR. ◊

Remark A.38. In general there are n^{n^2} possible binary operations on a set with n elements. Under *oeis.org/A027851* the On-Line Encyclopedia of Integer Sequences, lists the number of nonisomorphic semigroups with n elements, and under *oeis.org/A001423* the number of nonequivalent semigroups. ◊

[3] *http://en.wikipedia.org/wiki/Semigroup_with_two_elements* [2011-05-17]

Bibliography

[1] S. Arora and B. Barak. *Computational Complexity. A Modern Approach*. Cambridge University Press, Cambridge, 2009.

[2] J. M. Castaño. 'GIGs: restricted context-sensitive descriptive power in bounded polynomial-time'. In A. Gelbukh, editor, *Proceedings of the 4th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing'03, pages 22–35, Berlin Heidelberg, 2003. Springer-Verlag. *http://books.google.de/books?id=x4KDdwBOIk4C&pg=PA22*.

[3] J. M. Castaño. 'Global index grammars and descriptive power'. In R. T. Oehrle and J. Rogers, editors, *Proceedings of Mathematics of Languages 8*, pages 1–12. molweb.org, 2003. *http://molweb.org/mol8/papers_mol8/castano.pdf*.

[4] J. M. Castaño. 'LR Parsing for Global Index Languages (GILs)'. In O. Ibarra and Z. Dang, editors, *Implementation and Application of Automata*, volume 2759 of *Lecture Notes in Computer Science*, pages 191–225. Springer, Berlin Heidelberg, 2003. doi: 10.1007/3-540-45089-0_25.

[5] A. M. Cohen, H. Cuypers, and H. Sterk. *Algebra Interactive!* Springer-Verlag, Berlin Heidelberg, 1999.

[6] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Einführung in die mathematische Logik*. Spektrum Akademischer Verlag, Heidelberg Berlin, 1996.

[7] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Upper Saddle River, NJ, 2nd edition, 1994.

[8] J. M. Harris, J. L. Hirst, and M. J. Mossinghoff. *Combinatorics and Graph Theory*. Springer, New York, 2nd edition, 2008.

[9] S. Hedman. *A First Course in Logic. An Introduction to Model Theory, Proof Theory, Computability, and Complexity*. Oxford University Press, Oxford New York, 2004.

[10] D. W. Hoffmann. *Theoretische Informatik*. Carl Hanser Verlag, München, 2009.

[11] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, Boston, 2007.

[12] J. E. Hopcroft and J. D. Ullman. *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Addison-Wesley, Bonn, 1988.

[13] C. Jantzen and J. Schwermer. *Algebra*. Springer-Verlag, Berlin Heidelberg, 2006.

[14] O. Körner. *Algebra*. Akademische Verlagsgesellschaft, Frankfurt, 1974.

[15] S.-Y. Kuroda. 'Classes of languages and linear-bounded automata'. *Information and Control*, 7(2):207–223, 1964.

[16] Y. N. Moschovakis. *Notes on Set Theory*. Springer-Verlag, New York, 1994.

[17] C. M. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

[18] D. B. Searls. 'The language of genes'. *Nature*, 420(6912):211–217, 11 2002. DOI 10.1038/nature01255.

[19] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, Boston, 2006.

[20] B. von Querenburg. *Mengentheoretische Topologie*. Springer-Verlag, Berlin Heidelberg, 3. edition, 2001.

[21] E. Zeidler, editor. *Teubner Taschenbuch der Mathematik. Teil 1*, Leipzig, 1996. B. G. Teubner.

Web Resources

[GM] *http://www.informatics.sussex.ac.uk/research/groups/nlp/gazdar/nlp-in-lisp/* – Gerald Gazdar, Chris Mellish (1996): "Natural Language Processing in Lisp." [last access 2011-04-26]

[Mo] *http://www.staff.ncl.ac.uk/hermann.moisl/ell236/* – Hermann Moisl: "Computational Linguistics." [last access 2011-04-06]

[Ra] *http://page.mi.fu-berlin.de/raut/Mengenlehre/m.pdf* – Wolfgang Rautenberg (2008): "Grundkurs Mengenlehre." Lecture notes of Freie Universität Berlin [last access 2011-05-15]

[Zoo] *http://qwiki.stanford.edu/wiki/Complexity_Zoo* – Complexity Zoo Wiki at Stanford University. [last visited 2012-03-18]

Index