Formal Languages
An automata-theoretic introduction

Günter Hotz
Universität des Saarlandes

Klaus Estenfeld
Universität des Saarlandes

Armin Reichert
Alumnus Universität des Saarlandes

English Translation

Preface

This book is in essence the second edition of the book Hotz/Walter, "Automatentheorie und Formale Sprachen 2, Endliche Automaten". As it has been completely reworked, it really is a new edition.

While in the first edition only the theory of finite automata has been treated, in this edition also an introduction into the theory of context-free languages is given. This was only possible in the available space frame because of an automata-theoretic treatment of the theory.

Such a foundation of the theory has already been proposed by Goldstine in 1977 and has been sketched in various of his lectures. The motivation for developing my lecture, from which this book originates, in this way is nevertheless not based on his proposal. It has almost automatically been arosen from the work of the French school. I want to emphasize here the book by Jean Berstel on transductions. Mr Berstel finally pointed my to the work of Goldstine.

I fully support Goldstine's opinion that it would be worth rethinking the whole theory of formal languages along this automata-theoretic lines.

This book is only an introduction into the theory of formal languages. The interested reader who wants to gets a deeper understanding of the theory or who wants to get a different look into it is pointed to the books by Ginsburg, Harrison or Salomaa. Relations to applications can be found in books on complier design.

Dr. Klaus Estenfeld worked out my lecture "Formal Languages 1" which I held on that topic in 1980/81 to become the foundation of this book and he made a number of additions at some places.

Dipl.-Math. Bernd Becker carefully read the manuscript and contributed with his proposals to the success of this book.

The publisher as well as the editors of the series earn our thanks for their patience of waiting for the second edition.

Saarbrücken, August 1981

Günter Hotz

Introduction

There are several reasons for the interest in the theory of formal languages in computer science. Practical problems as they arise in the context of definition and translation of programming languages find an exact description in the theory of formal languages and thus get accessible to an exact treatment. Generation processes definable by formal languages can be interpreted as non-deterministic automata, that is as a generalized notion of a computer.

These kinds of generalizations in general are easier to understand than deterministic algorithms which contain more details that do not reflect the original problem but the necessity to uniquely define the algorithm. This is part of the reason to prove the correctness of programs in an understandable way. The proof of correctness for grammars or other mechanisms for generating languages on the other side offers the possibility to study correctness proofs at simpler objects.

The theory of formal languages in this respect contains the theory of algorithms but most often only the theory of context-free languages is treated because of her extraordinal simplicity and beauty.

In the spotlight of the theory are standing different methods for defining formal language classes, to study their word and equivalence problems, and to put them into different hierarchical classifications.

The generation processes themselves become objects of interest in the theory because the generation process of a language in case of programming languages relates to the semantics of programs.

Of course in the context of such a pocket book we have to make a strong selection of topics concerning language classes, generation processes as well as basic questions. In doing so, we let us guide by the intention to keep the formal machinery rather small.

Because the theory of finite automata is the foundation for the whole theory of formal languages, we start our book with this topic. In developing this theory we do not consider the technical realization of finite automata by logical circuits and binary storage devices but rather focus on the basic algorithm however it is realized. Our intuitive notion of finite automaton consists of a finite, oriented graph whose edges are labeled with the symbols from the input alphabet of the automaton. Depending on the input word we look for a path in the graph labeled with that word. If the end point of such a path, originating from the dedicated "start point" of the automaton, is a member of the set of "end points", our automaton "accepts" the word and doesn't so otherwise.

We prove the equivalence of this concept with the other known methods of defining finite automata. We prove the usual closure properties of languages defined by finite automata. Additionally we investigate the relation between deterministic and non-deterministic automata and also 2-way automata.

It is possible to generalize this theory in the direction of considering not only the free monoid of strings (words) over a finite alphabet but also arbitrary monoids.

By considering finite automata with output, which means to attach a second label at the graph edges, we get the theory of rational transducers. An extensive treatment of the theory of general transductions can be found in the book by Berstel.

Here, we restrict ourselves to some special generalizations of the free monoid (of words), namely the free group, the H-group (here, the relation $xx^{-1} = 1$ holds for x from the generating system, but not $x^{-1}x = 1$) and the polycyclic monoid (in addition to $xx^{-1} = 1$ it holds $xy^{-1} = 0$ for $x \neq y$ and $0x = x0 = 0$ for x, y from the generating system).

By investigating the transductions from free monoids into the polycyclic monoids one gets a smooth transition from the theory of finite automata into the theory of context-free languages.

The corresponding construction of the theory of context-free languages leads to a simple path to the most important representation theorems. This includes the theorems of Chomsky-Schützenberger, Shamir and Greibach. Also for the transformation into Greibach normal form we get a simple and efficient algorithm.

In the same easy way as for finite automata you can prove the known closure properties for context-free languages.

In the end we also prove the equivalence of this representation with the usual representation of context-free languages using context-free grammars.

Our buildup of the theory is very close to the one repeatedly recommended by Goldstine since 1977, but it originated independently. The difference is that we prove Greibach's representation theorem by making our automaton deterministic, namely by switching from output monoids to monoid rings. Doing that you get the theorem of Shamir in a natural way and from this the theorem of Greibach.

From the theorem of Shamir you can get quite easily the algorithm of Valiant for deciding the word problem of context-free languages. Because of lack of space this could not be included into this book, the same holds for the treatment of the deterministic languages.

We want to emphasize another advantage of this buildup of the theory: As known, the exact formalization of the notion of "derivation" when using grammars brings some difficulties. In our theory the "derivation tree" corresponds to a path in our graph.

Maybe the usage of non-free monoids at first is a problem for readers not used to it. But it seems to be the case that defining context-free languages that way supports the intuition. For example, the usage of "syntax diagrams" for the definition of programming languages gives some evidence for this.

Because we judge the former as rather important, we want to explain it on a specific example, namely the so-called **Dyck language**.

The **Dyck language** $D(X_k)$ contains the correctly nested bracket sequences over k different pairs of brackets, where $k \in \mathbb{N}$.

A formal definition of $D(X_k)$ is as follows:

Let $X_k = \{x_1, \ldots, x_k\}$ be an alphabet of k elements, for X_k there is $\bar{X}_k = \{\bar{x}_1, \ldots, \bar{x}_k\}$ such that $\bar{x}_i$ is regarded as the corresponding bracket for x_i.

Then it holds:

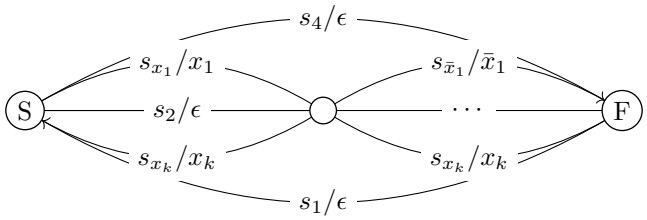
(1) $\epsilon \in D(X_k)$

(2) $u, v \in D(X_k) \Rightarrow u \cdot v \in D(X_k)$

(3) $u \in D(X_k) \Rightarrow x_i \cdot u \cdot \bar{x}_i \in D(X_k), \quad i = 1, \ldots, k$

(4) $D(X_k)$ is minimal with (1), (2) and (3).

For $D(X_k)$ we get the following syntax diagram:



If we consider all labelings of paths from S to F we get of course also words not contained in $D(X_k)$, for example $x_1x_2\bar{x}_k$ or $x_1\bar{x}_1\bar{x}_2$ etc.

We have to guarantee that we get Dyck words only. To do that, we define a homomorphism from the path category of the graph into the polycyclic monoid over $X_k \cup \bar{X}_k$, such that the homomorphic images of paths from S to F have a special form, for example they have to be equal to the unit of the polycyclic monoid.

Let us consider the word

$$x_1x_2\bar{x}_2\bar{x}_1x_2\bar{x}_2 \in D(X_2),$$

then we have paths

$$s_{x_1}s_2s_{x_2}s_{\bar{x}_2}s_3s_{\bar{x}_1}s_1s_{x_2}s_{\bar{x}_2}$$

and

$$s_{x_1}s_2s_{x_2}s_{\bar{x}_2}s_3s_{\bar{x}_1}s_3s_2s_{x_2}s_{\bar{x}_2}$$

which both have this word as their labeling and we can easily define a homomorphism in the sense above.

We get different paths in our graph leading to acceptance of the same word.

The problem to construct a graph such that for each word in the accepted language exactly one path exists, leads to the existence of the deterministic finite automaton with storage.

CHAPTER 1

Mathematical Foundations

1. Notations, basic notions

In this first section we want to define the elementary notions that are used throughout the whole book. We use the usual notions

$$\mathbb{N} = \{0, 1, 2, \ldots\} \text{ for the natural numbers}$$

$$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\} \text{ for the integer numbers}$$

$$\mathbb{Q} = \{\frac{a}{b} \mid a, b \in \mathbb{Z}, b \neq 0\} \text{ for the rational numbers}$$

For the set operations we use $\cup$ for the union and $\cap$ for the intersection. Also $A \subset B$, $a \in A$, $a \notin A$, $\bar{A}$, $A - B$, $A \times B$ and $\emptyset$ have their usual meaning. For the power set of a set A we write 2^A or $Pot(A)$. $Card(A)$ denotes the cardinality of A. Logical implication is denoted by $\Rightarrow$.

Mappings are denoted as $f : A \to B$, in that case f is a total mapping. We write $Q(f) = A, Z(f) = B$. Here Q stands for "Quelle" (source) and Z for "Ziel" (target).

If $f : A \to B, g : B \to C$ are mappings, then $f \circ g : A \to C$ is the composed mapping that one gets by applying f first and then g, i.e. $(f \circ g)(a) = g(f(a))$. If $f : A \to B$ and $C \subset A$, then $f(C) = \{f(c) \mid c \in C\}$.

A subset $R \subset A \times B$ is called a **relation** between A and B. $R_f \subset A \times B$ with $R_f = \{(a, b) \mid f(a) = b\}$ is the relation **induced by** the mapping f or the **graph** of f.

Let $f : A \to B$ be a mapping, $A_1 \subset A$ and $g : A_1 \to B$ a mapping. f is called the **continuation** of g if $f(a_1) = g(a_1), a_1 \in A_1$. In this case we also write $f \mid_{A_1} = g$ (in words: f restricted to A_1).

A **semi-group** consists of a set M and an associative operation on that set, usually denoted as a multiplication. If a semi-group is commutative, we also use "+" instead of "$\cdot$".

A semi-group is a **monoid** if M contains a neutral element. We often denote it with 1_M or shortly 1. In the commutative case we often write 0 instead of 1.

For $A, B \subset M$ we denote by $A \cdot B = \{ab \mid a \in A, b \in B\}$ the **complex product** of A and B.

$A \subset M$ is a **submonoid** of M if the following holds: $1_M \in A$ and A is closed under the operation of M.

For a set A, the set A^* defined as follows, is the smallest submonoid of M which contains A. More specific,

$$A^* = \bigcap_{M' \in M(A)} M'$$

where $M(A) = \{M' \subset M \mid M' \text{ is a submonoid of } M, A \subset M'\}$.

It is easy to see that

$$A^* = \bigcup_{n \geq 0} A^n \text{ with } A^0 = \{1\} \text{ and } A^{n+1} = A^n \cdot A$$

In the same sense the notion $A^+ = A^* - \{1\}$ is defined for semi-groups. A is called the **generation system** of A^* and A^+ resp.

A special meaning for us is assigned to the set of "words" (string) over a fixed alphabet A. We understand as words the finite sequences of elements from the alphabet A as for example (a, b, d, a, c) for alphabet $A = \{a, b, c, d\}$.

We define

$$WORD(A) := \{\epsilon\} \cup A \cup (A \times A) \cup (A \times A \times A) \cup \dots$$

as the set of words (strings) over A. The symbol ϵ denotes the **empty word** over A, that is $A^0 = \{\epsilon\}$.

If $w, v \in WORD(A)$ then $w \cdot v$ is the word over A which you get by concatenating w and v, more formally:

$$\text{If } w = (a_1, \dots, a_k), v = (a_{k+1}, \dots, a_n) \text{ then } w \cdot v = (a_1, \dots, a_n)$$

With this operation $WORD(A)$ becomes a monoid which is usually also denoted with A^*. This is slightly inconsistent because for the first definition of the $*$-operator it holds $(A^*)^* = A^*$ but for the second usage of the $*$-operator it holds $(A*)^* \neq A^*$.

The following example should clarify that:

Let $A = \{a, b, c\}$ and let (a, b, a) and $(b, a) \in A^*$.

$$(a, b, a) \cdot (b, a) = (a, b, a, b, a) \in A^*, \text{ but}$$

$$((a, b, a), (b, a)) \in (A^*)^* \text{ but } \notin A^*$$

Instead of (a) we write just a. In this sense it holds $A \subset A^*$. This also holds in the sense of the first definition of A^*.

If $w = (w_1, \dots, w_n)$ we call $|w| := n$ the **length** of w. Obviously it holds: $|w \cdot v| = |w| + |v|$ and $|\epsilon| = 0$.

The **mirror word** w^R of a word $w = (w_1, \dots, w_n)$ is the word $(w_n, \dots, w_1)$. It holds: $(w \cdot v)^R = v^R \cdot w^R$ and $\epsilon^R = \epsilon$.

In A^* the reduction rules hold, i.e.

(1) $w \cdot x = w \cdot y \Rightarrow x = y$

(2) $x \cdot w = y \cdot w \Rightarrow x = y$

We define **left** and **right quotient** for sets of words X, Y:

$$X^{-1} \cdot Y = \{w \mid \exists x \in X, y \in Y \text{ with } x \cdot w = y\}$$

and

$$X \cdot Y^{-1} = \{w \mid \exists x \in X, y \in Y \text{ with } w \cdot y = x\}$$

Because of the reduction rules it holds:

$\{w\}^{-1} \cdot \{v\}$ and $\{w\}^{-1} \cdot \{v\}$ are either empty or contain a single element.

If $\{w\}^{-1} \cdot \{v\}$ is not empty we call w a **prefix** of v. If $\{w\} \cdot \{v\}^{-1} \neq \emptyset$, we call v a **suffix** of w.

In the future we will always write just w instead of $\{w\}$ and also w is prefix of v if $w^{-1} \cdot v \neq \emptyset$.

2. Monoid homomorphisms and congruence relations

DEFINITION 2.1. *A* **monoid homomorphism** *(short: homomorphism) from a monoid M to a monoid S is a mapping $\Phi : M \to S$ with the following properties:*

(1) $\Phi(m_1 \cdot m_2) = \Phi(m_1) \cdot \Phi(m_2), \quad m_1, m_2 \in M$

(2) $\Phi(1_M) = 1_S$

It can be easily shown: if $M_1 \subset M$ is a submonoid of M, then $\Phi(M_1)$ is a submonoid of S. If S_1 is a submonoid of S, then $\Phi^{-1}(S_1)$ is a submonoid of M.

A homomorphism $\Phi : M \to S,\ M, S$ monoids, is called

monomorphism: if Φ is injective

epimorphism: if Φ is surjective

isomorphism: if Φ is bijective

Homomorphisms $\Phi : M \to M$ are called **endomorphisms**, isomorphisms $\Phi : M \to M$ are called **automorphisms**.

Monoids M and S are called *isomorphic*, if there exists an isomorphism between M and S.

Of course, a homomorphism cannot be defined arbitrarily on a monoid M. Thus the following two questions arise:

(1) If $M_1 \subset M$ is a submonoid and $\Phi_1 : M_1 \to S$ is an arbitrary mapping. When is it possible to extend Φ_1 to a homomorphism $\Phi : M \to S$?

(2) If Φ_1, Φ_2 both are homomorphisms from M to S which coincide on $M_1 \subset M$. In which way can Φ_1 and Φ_2 be different?

The answer to this question of course depends on the structure of M_1. If $M_1 = \{1_M\}$ then Φ is determined uniquely on M_1 but there is little information on the relation between Φ_1 and $\Phi 2$.

The following two simple theorems which can be found in introductory algebra books are holding:

(1) If M_1 is a generating system of M and $\Phi_1, \Phi_2 : M \to S$ both are monoid homomorphisms which coincide on M_1, then $\Phi 1 = \Phi_2$.

(2) If A is a set and $M = A^*$, and $\Phi_1 : A \to S$ is an arbitrary mapping, then there exists exactly one continuation Φ from Φ_1 which is a monoid homomorphism from A^* to S.

DEFINITION 2.2. *A subset $A \subset M$ is called a* **free generating system** *of M, if each mapping $\Phi_1 : A \to S$, where S is an arbitrary monoid, can be continued to a monoid homomorphims in a unique way.*

A monoid with a free generating system is called a **free monoid**.

A^* therefore is a free monoid and A is a free generating system of A^*.

It holds also: If A is a free generating system of M and A^* is the monoid of words (string) over A, then A^* and M are isomorphic.

A free monoid has at most one free generating system. From that we can see that the length $|w|$ of a word $w \in A^*$ can be defined in a unique way for any free monoid.

The length mapping L is an example for a monoid homomorphism $L : A^* \to \mathbb{N}$.

If $\Phi : M \to S$ is a monoid homomorphism, then the sets

$$\{\Phi^{-1}(s) \mid s \in S\} \subset Pot(M)$$

form a monoid isomorphic to $\Phi(M)$.

We want to handle now the following question: Let M be a monoid, $L \subset M$ be any subset of M. Does there exist a monoid S and a homomorphism $\Phi : M \to S$ with the following property: There exists an $s \in S$ with $L \subset \Phi^{-1}(s)$?

Of course, there always exists such an S: Choose $S = \{1\}$ and $\Phi(M) = \{1\}$.

Therefore we strengthen our task: Find S and Φ such that $L \subset \Phi^{-1}(S)$ and for each other homomorphism Ψ with that property holds: $L \subset \Psi^{-1}(S') \Rightarrow \Phi^{-1}(S) \subset \Psi^{-1}(S')$.

We want to describe L as close as possible by a monoid homomorphism.

Such an S and Φ exists for each $L \subset M$ (see Algebra text), it is named $synt_M(L)$ an is constructed as follows:

Definition 2.3 (syntactic congruence). *Let M be a monoid and $L \subset M$. For $a, b \in M$ we define*

$$a \equiv b \ (L) \Leftrightarrow \textit{for all } u, v \in M : u \cdot a \cdot v \in L \Leftrightarrow u \cdot b \cdot v \in L$$

$\equiv (L)$ is a congruence relation, it holds:

(1) Let $[a]_L = \{b \in M \mid a \equiv b \ (L)\}$ then $b \in [a]_L \Rightarrow [a]_L = [b]_L$

(2) If we define $[a]_L \cdot [b]_L := [ab]_L$ (complex product), then

$$synt_M(L) = \{[a]_L \mid a \in M\}$$

becomes a monoid and the mapping

$$\Psi_L : M \to synt_M(L), \ \Psi_L(a) = [a]_L$$

is a monoidepimorphism.

We call $\equiv (L)$ the **syntactic congruence** of L and $synt_M(L)$ the **syntactic monoid** of L wrt. M.

To motivate the name "syntactic monoid" we give an example from German language. Let A be the alphabet of German and L the set of sentences in German. One can denote two words w_1 and w_2 as congruent if they can always be exchanged in each german sentence. There exist words that cannot always be exchanged. In the sentence "Apfel ist eine Kernfrucht" the word "Apfel" can be exchanged by "Birne" but this is not possible in the sentence "Apfel schreibt sich A p f e l".

The difficulty is of semantic nature. If you don't consider semantic correctness of sentences you get a classification of words wrt. their syntactic meaning.

The important notion of "syntactic congruence" has been introduced by M. P. Schützenberger in the context of coding problems.

3. Special monoids and the free group

We have just learned about the syntactic monoid as an example for a monoid. Further information on the theory of syntactic monoids can be found in [**?**] and [**?**].

Let's have a look at more special monoids which we will need again later. To do so, we introduce the notion of **generated congruence relation**.

Let A be an alphabet and $R = \{u_i = v_i \mid i = 1, \ldots, n,\ u_i, v_i \in A^*\}$ a set of equations.

Then by the following conditions an congruence relation $\bar{R}$ is uniquely determined:

(1) $\{(u_i, v_i) \mid u_i = v_i \in R\} \subset \bar{R}$

(2) $\bar{R}$ is a congruence relation

(3) $\bar{R} \subset R'$ for all R' fulfilling conditions 1) and 2).

$\bar{R}$ is called the **congruence relation generated by** R over A^*.

The factor monoid $A^*/\bar{R}$ is named also simply A^*/R.

It holds: Words $u, v \in A^*$ are congruent wrt. $\bar{R}$ (Notation: $u \equiv v(\bar{R})$) iff there exists $n \in \mathbb{N}, u_i \in A^*$ with $u_i = u_{i,1} \cdot u_{i,2} \cdot u_{i,3}$ such that for $i = 1, \ldots, n$ it holds:

(1) $u = u_1, v = u_n$

(2) $u_{i,1} = u_{i+1,1}, u_{i,3} = u_{i+1,3}, (u_{i,2} = u_{i+1,2}) \in R$ for all $i = 1, \ldots, n-1$.

We say: v is constructed from u by applying the equations from R.

The congruence classes of $u \in A^*$ in A^*/R are denoted by $[u]_{A^*/R}$ or just $[u]$.

Definition 3.1. *Let X be an alphabet. Define $X^{-1} := \{x^{-1} \mid x \in X\}$ as the set of formal inverses.*

We can think of x and x^{-1} as corresponding pairs of brackets as we did in the definition of the Dyck languages in the introduction.

We will now consider different partitionings of $(X \cup X^{-1})^*$ wrt. to different congruence relations and investigate the corresponding factor monoids.

Definition 3.2.

$$X^{[*]} := (X \cup X^{-1})^*/\{xx^{-1} = 1 \mid x \in X\}$$

is called the **H-group**. *(The name (H = "half") shall remember of semi-group).*

Now we introduce a special absorbing element 0 by defining:

Definition 3.3.

$X^{(*)} := (X \cup X^{-1} \cup \{0\})^*/\{xx^{-1} = 1, xy^{-1} = 0, 0z = z0 = 0 \mid x, y \in X, z \in X \cup X^{-1}\ \{0\}\}$

is called the **polycyclic monoid**.

Using the naming of the previous section we get:

$$X^{(*)} = synt_{X^*}(D(X))$$

which means: the polycyclic monoid is the syntactic monoid of the Dyck language.

Definition 3.4.

$$F(X) := (X \cup X^{-1})^* / \{xx^{-1} = x^{-1}x = 1 \mid x \in X\}$$

is the **free group** over X.

Remark: It holds $D(X) = [1]_{X^{(*)}}$ and $D(X) = [1]_{X^{[*]}}$, which means the Dyck language is the set of words from $(X \cup X^{-1})^*$ which can be reduced to the empty word.

In the following we will mainly consider the H-group over X.

For $w \in (X \cup X^{-1})^*$ we define the reduced word $|w|$ as follows: If w does not contain a subword of the form xx^{-1} then $|w| = w$. Otherwise, replace the leftmost occurence of xx^{-1} by the empty word 1.

This process is called **reduction** and the result is denoted by $\rho(w)$. One can easily prove:

Lemma 3.1. *There exists a minmal number $k \in \mathbb{N}$ with $\rho^k(w) = |w|$. The number k is called the* **reduction length** *of w. It holds: $\rho(|w|) = |w|$.*

Lemma 3.2.

$$[w] = [w'] \in X^{[*]} \Leftrightarrow |w| = |w'|.$$

Proof:

"$\Leftarrow$":

It holds $w \equiv |w| = |w'| \equiv w' \Rightarrow [w] = [w']$.

"$\Rightarrow$":

Let $[w] = [w']$. We may assume that w' is created from w by application of an equation $xx^{-1} = 1$. Let $w = w_1xx^{-1}w_2$ and $w' = w_1w_2$.

We show: If k is the reduction length of w_1 then $\rho^{k+1}(w) = \rho^k(w')$ (thus the reduced words are equal).

Proof by induction over k:

$k = 0$: w_1 is already reduced, so $\rho(w) = w_1w_2 = w'$.

$k > 0$: It holds $\rho(w) = \rho(w_1xx^{-1}w_2), \quad \rho(w') = \rho(w_1w_2)$. The reduction length of $\rho(w)$ by induction proposition is $k - 1$ and $\rho^k\rho(w) = \rho^{k-1}\rho(w') \Rightarrow$ the reduced word of w and w' is the same so $|w| = |w'|$.

Remark: Using the same argument one can show that the creation of the reduced word doen not depend on the order of the reductions.

Therefore the reduced word for a representant of an element of $X^{[}*]$ is unique, so we can just speak of "the" reduced word in the following.

Remark: These results have been used in [**?**] to obtain a space-optimal algorithm for the analysis of the Dyck language.

Similar results also hold for the free group $F(X)$, see [**?**].

4. Graphs, categories and functors

Before defining graphs formally, we want to describe what we mean by a graph. A graph consists of points and edges. Each edge connects two points which are not necessarily different. You can imagine a graph as streetmap, the cities are the points and the streets are the edges of the graph. The edges may be oriented such that they have a one-way direction. Paths in graphs are sequences of edges that you could drive for example with a car without violating the traffic rules.

One can show that every graph as we will formally define has, with a certain restriction, a faithful(?) image in $\mathbb{R}^3$, see [**?**]. The points of the graph are here the points in $\mathbb{R}^3$, the edges are lines in $\mathbb{R}^3$ which do not intersect pair-wise.

The mentioned restriction is that the graph must not have more points than the cardinality of $\mathbb{R}^3$. The restriction concerning the edges is more severe: It say that there is at most one edge between two points and that the graph has no loops. Loops are edges with just a single point.

From what has been said we see that we may use a concrete geometric picture of a graph without getting our intuition mistaken. The following definition of a graph nevertheless doe not contain any geometry.

DEFINITION 4.1 (graph). *A* **graph** $G = (V, E)$ *consists of a non-empty set* V *of points (also called vertices) and a set* E *of edges and a mapping* $\rho : E- > Pot(V)$ *with* $card(\rho(e)) <= 2$ *for* $e \in E$. $\rho(e)$ *is the set of* border points *of* e.

Border points of an edge do not need to be different. If $card(\rho(e)) = 2$ we call e a **line**, if $card(\rho(e)) = 1$ we call it a **loop**.

DEFINITION 4.2 (loop-free). *A graph is called* **loop-free** *if it does not contain a loop.*

We introduce an orientation for the edges.

DEFINITION 4.3 (oriented graph). *A graph* $G = (V, E)$ *is called an* **oriented graph** *if there are two mappings* $Q : E \to V$ *and* $Z : E \to V$ *with* $\rho(e) = Q(e), Z(e)$ *for all* $e \in E$.

$Q(e)$ is called the **source** and $Z(e)$ the target point of e. The notions of loop and line are naturally transferred to oriented graphs.

For each graph one can assign the corresponding oriented graph $\hat{G}$ by defining two edges (P_1, e, P_2) and (P_2, e, P_1) for every edge e with border points P_1 and P_2 and defining $Q((P_1, e, P_2)) = P_1 = Z((P_2, e, P_1))$ and $Q((P_2, e, P_1)) = P_2 = Z((P_1, e, P_2))$.

DEFINITION 4.4 (connected graph). *A graph* $G = (V, E)$ *is called* **connected** *if in the corresponding oriented graph* $\hat{G}$ *for each points* P *and* P' *there exist edge sequences* $e_1, \ldots, e_k$ *with* $Q(e_1) = P, Z(e_k) = P'$ *and* $Z(e_i) = Q(e_{i+1})$ *for all* $i = 1, \ldots, k-1$.

DEFINITION 4.5 (ordered graph). *A loop-free graph* $G = (V, E)$ *is called* **ordered** *if for each point* $P \in V$ *holds: There exists a unique (up-to cyclic permutation) ordering on the set* $\{e \in E \mid P \in \rho(e)\}$.

Notation: $\{e \in E \mid P \in \rho(e)\}$ is called the **cycle** belonging to P ($cycle(P)$).

Explanation: Image each point and its adjacent edges to be stuck on a litte circle as in the following figure:

FIGURE

DEFINITION 4.6 (oriented graph). *A loop-free,* **oriented** *graph G is called* **ordered** *if for all points $P \in V$ it holds: There exists an ordering $e_1, \ldots, e_k, e'_m, \ldots, e'_1$ such that $\{e_1, \ldots, e_k\} = \{e \in E \mid Z(e) = P\}$ and $\{e'_1, \ldots, e'_m\} = \{e \in E \mid Q(e) = P\}$.*

$e_1, \ldots, e_k$ is called the **ordering** of the incoming edges of P and $e'_1, \ldots, e'_m$ the **ordering** of the outgoing edges of P.

Example:

FIGURE

DEFINITION 4.7 (path). *A* **path** *in an oriented graph G is a sequence $w = (Q_1, e_1, \ldots, e_k, Z_k)$ with $k \geq 1$ and $e_1, \ldots, e_k \in E, Q(e_1) = Q_1, Z(e_k) = Z_k$ and $Q(e_{i+1}) = Z(e_i)$ for all $i = 1, \ldots, k-1$.*

We extends the mappings Q and Z onto paths by defining $Q(w) := Q_1$ and $Z(w) := Z_k$. Q_1 is called the **start point** and Z_k the **end point** of path w.

k is the **length** of w, written as $L(w) = k$. For $k = 0$ we declare for all points $P \in V$ that $w = (P, P)$ is the path of length 0 from P to P.

Paths in arbitrary graphs are defined by switching to the oriented graph $\hat{G}$.

DEFINITION 4.8 (subpath). *Let $w = (Q, e_1, \ldots, e_k, Z_k)$ be a path. A path $w' = (Q'_1, e'_1, \ldots, e'_m, Z'_m)$ is called a* **subpath** *of w, if it holds: $\exists i, i \leq i \leq k$ such that $e'_j = e'_{i+j-1},\ j = 1, \ldots, m$ and $i + m - 1 \leq k$.*

A path is called **closed** if $Q(w) = Z(w)$, it is called a **circle** if it is closed and does not contain any closed subpath w' with $L(w') > 0$.

DEFINITION 4.9 (circle-free graph). *A graph $G = (V, E)$ is called* **circle-free** *if there are no circles in G.*

For our purposes we will only consider oriented graphs. For these graphs the following definition reflects a special connectivity property.

DEFINITION 4.10 (star, center). *Let $G = (V, E)$ be an oriented graph and $P \in V$. G is called a* **star around P** *if for each $P' \in V$ there exists a path $w_{P'}$ with $Q(w_{P'}) = P$ and $Z(w_{P'}) = P'$. P is called the* **center** *of G.*

We want to introduce now a special kind of graph that plays a central role in the theory of formal languages.

DEFINITION 4.11 (tree). *A* **tree** *is a circle-free star where for all $P \in V$ it holds $card(\{e \in E \mid Z(e) = P\}) \leq 1$.*

The following lemma holds:

LEMMA 4.1. *A tree has exactly one center which is called the* **root**.

Historical remark: Leonard Euler (1735) at a walk in Königsberg asked himself if he could traverse each of the seven bridges over the Memel in such a way that he would traverse each bridge exactly once. In the figure below you can see a graph describing the situation. Euler gave a simple criterion for the existance of paths that traverse each edge of a graph exactly once (the so called Euler paths).

FIGURE

Now we want to concatenate paths or mathematically, define a product operation on paths. We define:

$$\begin{aligned}&(Q_1, e_1, \ldots, e_k, Z_k) \cdot (Q_{k+1}, e_{k+1}, \ldots, e_n, Z_n)\\ &:= (Q1, e_1, \ldots, e_n, Z_n) \text{ if } Q_{k+1} = Z_k\end{aligned}$$

That means you can concatenate two paths if the end point of the first is the start point of the second path. Obviously it holds:

(1) The product of paths is associative (if defined)

(2) For each point P of a graph G there exists exactly one path $1_P := (P, P)$ such that for each path w it holds:

(3)

$$\begin{aligned}w \cdot 1_P &= w, \text{ if } Z(w) = P\\ 1_P \cdot w &= w, \text{ if } Q(w) = P\end{aligned}$$

We denote the set of paths of a graph G with $\mathcal{W}(G)$.

$\mathcal{W}(G)$ is called the **path category** of G and G in this context is also called **schema**. $\mathcal{W}(G)$ is an important special case of a category.

Notation:

$$\mathcal{W}(G)(P, P') := \{w \in \mathcal{W}(G) \mid Q(w) = p, Z(w) = P'\}$$

Categories are algebraic structures with a *partial* operation.

DEFINITION 4.12 (category). *$C = (O, M, Q, Z, \circ)$ is called a* **category** *if the axioms (K1) to (K4) are fulfilled:*

(K1) *O and M are sets and $Q : M \to O$ and $Z : M \to O$ are mappings. $Q(f)$ is the source of f and $Z(f)$ is the target of f, O is the set of* **objects** *and M the set of* **morphisms** *of the category C.*

(K2) *For $f, g \in M$ the operation $\circ$ is defined if $Q(g) = Z(f)$. In this case it holds $f \circ g \in M, Q(f \circ g) = Q(f), Z(f \circ g) = Z(g)$.*

(K3) *The associative law $(f \circ (g \circ h)) = (f \circ g) \circ h$ holds in the sense that each of both sides is defined if one of both is.*

(K4) *For each object $w \in O$ there exists a unit morphism $1_w \in M$ with $Q(1_w) = Z(1_w) = w$ and for all morphisms $f, g \in M$ with $Q(f) = Z(g) = w$: $1_w \circ f = f$ and $g \circ 1_w = g$. It can be easily shown that there exists exactly one unit morphism for each object w.*

Notations: $Obj(C) := O$ is the **set of objects** and $Mor(C) := M$ the **set of morphisms** of the category C.

Historical remark: Euler was already interested in graphs and paths in graphs. The path category has already been used before the notion of category even existed. The axiomatic formulation of categories and its importance for many areas of mathematics has been elaborated by S. Eilenberg and S. MacLane in 1945 [**?**]. Their work has stimulated a broad, very abstract theory of categories. We will only use the notations for structures which are categories and some elementary concepts which also in the theory of formal languages lead to fruitful questions.

We explain the notion of category on a number of examples:

(1) **The category of relations**
Define $REL(O) = (O, M, Q, Z, \circ)$ by:

- Let O be a set of sets ($O \notin O$).
- $M = \{(A, B, R) \mid A \in O, B \in O, R \subset A \times B\}$
- $Q(A, B, R) = A, \quad Z(A, B, R) = B$
- $(A, B, R_1) \circ (B, C, R_1) = (A, C, R')$
  where $R' = \{(a, c) \mid \exists b \in B : (a, b) \in R_1 \text{ and } (b, c) \in R_2\}$

With these definitions $REL(O)$ becomes a category.

(2) **The category of matrices**
Let $MAT(\mathbb{Q}) = (O.M, Q, Z, \circ)$ with

- $O = \mathbb{N}$
- $M =$ the set of $k \times n$ matrices, $k, n \in \mathbb{N}$, with entries from .
- For a $k \times n$ matrix $A_{k,n}$ define source and target mappings by

  $Q(A_{k,n} = k)$ the number of rows

  $Z(A_{k,n} = n)$ the number of columns

With the matrix multiplikation as category operation $\circ$ the set $MAT(\mathbb{Q})$ becomes a category. Units in this category are the $n \times n$ unit matrices.

Analogously to the monoid homomorphisms we introduce structure-preserving mappings between categories, named **functors**.

Definition 4.13 (functor). *Let $C_i = (O_i, M_i, Q_i, Z_i, \circ_i), i = 1, 2$ be two categories and $\phi_1 : O_1 \to O_2$ and $\phi_2 : M_1 \to M_2$ be mappings.*

$\phi = (C_1, C_2, \phi_1, \phi_2)$ is called a **functor** *from C_1 to C_2 if the axioms (F1) to (F3) hold:*

(F1) *The diagram*

$$\begin{array}{ccccc} O_1 & \xleftarrow{Q_1} & M_1 & \xrightarrow{Z_1} & O_1 \\ \downarrow{\scriptstyle \phi_1} & & \downarrow{\scriptstyle \phi_2} & & \downarrow{\scriptstyle \phi_1} \\ O_2 & \xleftarrow{Q_2} & M_2 & \xrightarrow{Z_2} & O_2 \end{array}$$

is commutative.

(F2) $\phi_2(f \circ_1 g) = \phi_2(f) \circ_2 \phi_2(g)$ *for all* $f, g \in M_1$ *with* $Z(f) = Q(g)$.

(F3) $\phi_2(1_w) = 1_{\phi_1(w)}$ *for all* $w \in O_1$.

A functor ϕ is called injective (surjective, bijective) if ϕ_1 and ϕ_2 are injective (surjective, bijective).

Let's look at some examples:

Example 1: Consider the following oriented graphs G_1 and G_2:

FIGURE

$G1 = (V_1, E_1)$ represents an infinite binary tree. From each point of the tree two edges go out which are labeled with f and g.

$G_2 = (V_2, E_2)$ consists of a single point P_0 and two loops labeled with f and g respectively.

Consider the path categories $\mathcal{W}(G_1)$ and $\mathcal{W}(G_2)$. For $P \in V_1$ define $\phi_1(P) := P_0$ and $\phi_2(1_P) := 1_{P_0}$.

For an edge $e \in E_1$ we define:

$$\phi'(e) = \begin{cases} f & \text{if } e \text{ is marked with } f \\ g & \text{if } e \text{ is marked with } g \end{cases}$$

Now we define for $(P, e_1, \ldots, e_n, P') \in \mathcal{W}(G_1)$:

$\phi_2((P, e_1, \ldots, e_n, P')) = (P_0, \phi_2'(e_1), \ldots, \phi_2'(e_n), P_0)$.

Obviously $\phi = (\mathcal{W}(G_1), \mathcal{W}(G_2), \phi 1, \phi_2)$ is a functor.

It is a special functor because

(1) ϕ is surjective

(2) If P_1 is a point in G_1 and $\bar{w}$ is a path in G_2, then there exists exactly one path w in G_1 with $Q(w) = P_1$ such that $\phi_2(w) = \bar{w}$.

Example 2: Let graphs $G1, G_2$ be given as follows:

FIGURE

Then there exists a surjective functor from $\mathcal{W}(G_1)$ to $\mathcal{W}(G_2)$.

It is possible to construct surjective functors which fulfill (2) from example 1 and other surjective functors which don't.

Example 3: Let G_1 and G_2 be given as:

FIGURE

We define:

$$\begin{aligned} \phi_1(P_1) &= Q_1 \\ \phi_1(P_2) &= Q_2 \\ \phi_1(P_3) &= Q_2 \\ \phi_1(P_4) &= Q_3 \\ \phi_2((P_1, s, P_2)) &= (Q_1, f, Q_2) \\ \phi_2((P_3, r, P_4)) &= (Q_2, g, Q_3) \end{aligned}$$

For the units, the defintion of ϕ_2 is clear.

One can see that $\phi = (\mathcal{W}(G_1), \mathcal{W}(G_2), \phi_1, \phi_2)$ is a functor.

It is remarkable that $\phi_2(\mathcal{W}(G_1))$ is not a category because this set is not closed under the $\circ$ operation.

Example 4: Let G_1 and G_2 be given as follows:

FIGURE

We define:

$$\begin{array}{rcl} \phi_1(1) & = & 1' \\ \phi_1(2) & = & 2' \\ \phi_1(3) & = & 3' \\ \phi_1(4) & = & 3' \\ \phi_1(5) & = & 3' \\ \phi_2(a) & = & a' \\ \phi_2(b) & = & b' \\ \phi_2(c) & = & c' \\ \phi_2(d) & = & d' \\ \phi_2(e) & = & 1_3 \\ \phi_2(f) & = & 1_3 \\ \phi_2(g) & = & 1_3 \end{array}$$

$\phi = (\mathcal{W}(G_1), \mathcal{W}(G_2), \phi_1, \phi_2)$ is a functor.

Example 5: The graph G shall be defined by

FIGURE

Additionally, the following matrices are given:

$$a' = \begin{pmatrix} 1 & 0 & 2 & 1 \\ 0 & 1 & 2 & 5 \\ 1 & 1 & 2 & 1 \end{pmatrix} \qquad b' = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 2 & 2 & 2 \\ 1 & 5 & 1 \end{pmatrix} \qquad c' = \begin{pmatrix} 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}$$

$$d' = \begin{pmatrix} 7 & 4 \\ 5 & 3 \\ 3 & 5 \\ 4 & 7 \end{pmatrix} \qquad e' = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 1 & 0 & 0 & 0 \end{pmatrix} \qquad f' = \begin{pmatrix} 1 & 2 \\ 2 & 0 \end{pmatrix}$$

We consider $\mathcal{W}(G)$ and $MAT(\mathbb{N})$, the category of matrices over $\mathbb{N}$.

We define $\phi_1(i) = i$ for $i = 2, 3, 4$ and $\phi_2'(x) = x'$ for $x \in \{a, b, c, d, e, f\}$.

ϕ_2' can be extended in a unique way to a mapping $\phi_2 : \mathcal{W}(G) \to MAT(\mathbb{N})$ such that $\phi = (\mathcal{W}(G), MAT(\mathbb{N}), \phi_1, \phi_2)$ is a functor.

We want to define now some special properties of functors.

DEFINITION 4.14. *Let G_1, G_2 be ordered graphs, $\phi = (\mathcal{W}(G_1), \mathcal{W}(G_2), \phi_1, \phi_2)$ a functor.*

ϕ is called **ordered** *or* **order preserving** *if it holds:*

Let $\phi_1(P) = P' \in V_2$ for any $P \in V_1$, then for the ordering $e_1, \ldots, e_k, e'_m, \ldots, e'_1$ which belongs to P it holds:

$\phi_2(e_1), \ldots, \phi_2(e_k), \phi_2(e'_m), \ldots, \phi_2(e'_1)$ is contained in the ordering that belongs to P' in the given order.

It is possible that lines coincide which are counted only once in that case.

Let's give an example for this definition:

Let $P \in V_1$ be a point with ordering $e_1, e_2, e_3, e'_4, e'_3, e'_2, e'_1$ and $P' \in V_2$ be a point with ordering $r_1, r_2, r'_5, r'_4, r'_3, r'_2, r'_1$ as shown in the following figure:

FIGURE

Define ϕ by $\phi_1(P) = P'$ and

$$\phi_2(e_1) = r_1, \phi_2(e_2) = r_2, \phi_2(e_3) = r_2$$
$$\phi_2(e'_1) = r'_1, \phi_2(e'_2) = r'_3, \phi_2(e'_3) = r'_4, \phi_2(e'_4) = r'_5$$

Then ϕ respects the ordering in point P.

DEFINITION 4.15. *Let G_1, G_2 be oriented graphs and $\phi = (\mathcal{W}(G_1), \mathcal{W}(G_2), \phi_1, \phi_2)$ be a functor.*

ϕ is called **regular** *$\Leftrightarrow$ the restriction of ϕ_2 to the set $\{e \in E_1 \mid Q(e) = P\}$ and $\{e' \in E_2 \mid Q(e') = \phi_1(P)\}$ and to $\{e \in E_1 \mid Z(e) = P\}$ and $\{e' \in E_2 \mid Z(e') = \phi_1(P)\}$ for $P \in V_1$ is bijective.*

To each incoming / outgoing edge of a point $P \in V_1$ corresponds exactly one incoming / outgoing edge of $\phi_1(P) \in V_2$.

In our example, ϕ was not regular.

We slightly weaken the definition of a regular functor by only postulating regularity on the outgoing edges.

DEFINITION 4.16. *Let G_1, G_2 be oriented graphs and $\phi = (\mathcal{W}(G_1), \mathcal{W}(G_2), \phi_1, \phi_2)$ be a functor.*

ϕ is called **out-regular** *$\Leftrightarrow$ the restriction of ϕ_2 to the set $\{e \in E_1 \mid Q(e) = P\}$ and $\{e' \in E_2 \mid Q(e') = \phi_1(P)\}$ for $P \in V_1$ is bijective.*

The following lemma holds:

LEMMA 4.2. *If $\phi = (\mathcal{W}(G_1), \mathcal{W}(G_2), \phi_1, \phi_2)$ is an out-regular functor, then $\phi(\mathcal{W}(G_1))$ is a category.*

Our next lemma describes a well-known fact from graph theory that has found many applications.

LEMMA 4.3. *To each circle-free star $G = (V, E)$ relative to a point P there exists a tree B and an out-regular functor $(\mathcal{W}(B), \mathcal{W}(G), \phi_1, \phi_2)$ mapping the root of the tree B to the point P. B is determined up to isomorphisms.*

5. Subcategory, generating system

DEFINITION 5.1. *Let*

$$U = (Obj(U), Mor(U), Q_U, Z_U, \circ_U)$$

and

$$C = (Obj(C), Mor(C), Q_C, Z_C, \circ_C)$$

be categories.

U is called a **subcategory** *of $C \Leftrightarrow$*

(1) $Obj(U) \subset Obj(C)$ *and* $Mor(U) \subset Mor(C)$

(2) $Q_U = Q_C|_{Mor(U)}$ *and* $Z_U = Z_C|_{Mor(U)}$

(3) $\circ_U = \circ_C|_{Mor(U)\times Mor(U)}$

(4) *For* $w \in Obj(U) \Rightarrow 1_w \in Mor(U)$

U is called **full subcategory** of $C \Leftrightarrow$

$$\forall w_1, w_2 \in Obj(U), f : w_1 \to w_2 \in Mor(C) \Rightarrow f \in Mor(U)$$

This means, all morphisms in C between objects in U are also morphisms in U. $f : w_1 \to w_2$ stands for $Q(f) = w_1 \wedge Z(f) = w_2$.

We want to explain this fact at some examples:

Example 1:

Let $A = \{x, y, z, a, b, c\}$ and $f, g, h : A^* \to A^*$ be mappings defined as follows:

$$f(u_1 \cdot x \cdot u_2 \cdot x \cdots x \cdot u_k \cdot x) = u_1 \cdot ax \cdot u_2 \cdot ax \cdots ax \cdot u_k \cdot ax)$$

where $u_i \in (A - \{x\})^*$,

$$g(u_1 \cdot y \cdot u_2 \cdot y \cdots y \cdot u_k \cdot y) = u_1 \cdot by \cdot u_2 \cdot by \cdots by \cdot u_k \cdot by)$$

where $u_i \in (A - y\})^*$,

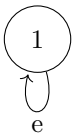
$$h(u_1 \cdot z \cdot u_2 \cdot z \cdots z \cdot u_k \cdot z) = u_1 \cdot cz \cdot u_2 \cdot cz \cdots cz \cdot u_k \cdot cz)$$

where $u_i \in (A - y\})^*$.

Let M be the monoid of mappings generated by $f, g, h : A^* \to A^*$.

Then $C = (A^*, M, Q, Z, \circ)$ is a category, if $\circ$ denotes the monoid operation in M.

Let G be the graph defined by the following figure:



We define a functor $\phi = (\mathcal{W}(G), C, \phi_1, \phi_2)$ by $\phi_1(1) := A^*$ and $\phi_2(e) := f \circ g \circ h$.

$\phi_2(\mathcal{W}(G))$ is a subcategory of C and it holds:

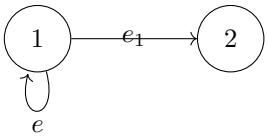
$$\phi_2(\mathcal{W}(G))(xyz) = \{(a^n x b^n y c^n z) \mid n \in \mathbb{N}\}$$

Example 2:

We expand on example 1. In addition to f, g, h we have three monoid homomorphisms f_1, g_1, h_1 defined by:

$$\begin{aligned} f_1(x) = \epsilon, f_1(u) = u \quad \forall u \in A - \{x\} \\ g_1(y) = \epsilon, g_1(u) = u \quad \forall u \in A - y\} \\ h_1(z) = \epsilon, h_1(u) = u \quad \forall u \in A - z\} \end{aligned}$$

We extend the graph G as follows to a graph G_1:



Consider $\mathcal{W}(G_1)(1,2)$. Then $\mathcal{W}(G_1)(1,2) \cup \{1_1, 1_2\}$ is a subcategory of $\mathcal{W}(G_1)$. In addition, $\mathcal{W}(G)$ is a subcategory of $\mathcal{W}(G_1)$.

We extend the functor ϕ from example 1 onto $\mathcal{W}(G_1)$ by defining:

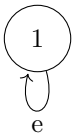
$$\phi_2(e_1) := f_1 \circ g_1 \circ h_1$$

We get:

$$\phi_2(\mathcal{W}(G_1)(1,2))(xyz) = \{(a^n b^n c^n \mid n \in \mathbb{N}\}.$$

Example 3:

Let G be defined as follows:



The full subcategory of $\mathcal{W}(G)$ generated by $\{1', 2', 3'\}$ is the path category $\mathcal{W}(G')$ with the following graph G':

FIGURE

As an exercise, one can show: The mapping ϕ_1 defined by

$$\begin{aligned}\phi_1(1) &= 1'\\ \phi_1(4) &= 1'\\ \phi_1(2) &= 2'\\ \phi_1(5) &= 2'\\ \phi_1(3) &= 3'\\ \phi_1(6) &= 3'\end{aligned}$$

can be extended to a functor $\phi = (\mathcal{W}(G), \mathcal{W}'(G'), \phi_1, \phi_2)$ where ϕ_2 has to be chosen in a suitable way.

Remark: The preimage of a closed path does not have to be closed.

We prove now the following

LEMMA 5.1. *Let*

$$C_i = (O_i, M_i, Q, Z, \circ),\ i = 1, 2, 3$$

be categories and C_1 and C_2 be subcategories of C_3. Then $C_1 \cap C_2$ is a category.

Proof: It holds

(1) $w \in O_1 \cap O_2 \Rightarrow 1_w \in M_1 \cap M_2$

(2) $f, g \in M_1 \cap M_2 \Rightarrow f \circ g \in M_1 \cap M_2$, if $Z(f) = Q(g)$

It follows that $C_1 \cap C_2$ is a category.

LEMMA 5.2. *Let $C_i = (O_i, M_i, Q, Z, \circ),\ i \in I$, where I is an arbitrary index set, be categories. If $C_I, i \in I$, are subcategories of a category C, then*

$$\tilde{C} := \bigcap_{i \in I} C_i$$

is a category.

The proof is similar as the one of the previous lemma.

DEFINITION 5.2 (generated subcategory). *Let $C = (O, M, Q, Z, \circ)$ be a category and $O_1 \subset O, M_1 \subset M$ and*

$$\mathcal{U}_C(O_1, M_1) := \{C' \mid C' \text{ is subcategory of } C, O_1 \subset O', M_1 \subset M'\}.$$

Then

$$<O_1, M_1> := \bigcap_{C' \in \mathcal{U}_C(O_1, M_1)} C'$$

is called the subcategory of C generated by (O_1, M_1) and (O_1, M_1) is called the generating system of $<(O_1, M_1)>$.

Obviously for each category $C = (O, M, Q, Z, \circ)$ it holds: $C = <O, M>$.

We say M_1 "generates" $<O_1, M_1>$, if

$$O_1 = \{Q(m) \mid m \in M_1\} \cup \{Z(M) \mid m \in M_1\}.$$

We have already seen an example for a nontrivial generating system.

Let $G = (V, E)$ be a graph, then E is a generating system of $\mathcal{W}(G)$ which means $\mathcal{W}(G) = <E(G)>$. The path category of a graph has a special property namely that $E(G)$ is a **free** generating system of $\mathcal{W}(G)$.

DEFINITION 5.3 (free generating system). *Let $C = (O, M, Q, Z, \circ)$ be a category and $E \subset M$. E is called a* **free generating system** *of C, if the following holds:*

If $C' = (O', M', Q, Z, \circ)$ is an arbitrary category and $\phi_1 : O \to O'$ and $\phi_s : E \to M'$ are mappings which fulfill the following diagram:

$$\begin{array}{ccccc} O & \xleftarrow{Q} & E & \xrightarrow{Z} & O \\ \downarrow{\scriptstyle \phi_1} & & \downarrow{\scriptstyle \phi_2'} & & \downarrow{\scriptstyle \phi_1} \\ O' & \xleftarrow{Q} & M' & \xrightarrow{Z} & O' \end{array}$$

Then there exists a unique continuation of ϕ_2' to $\phi_2 : M \to M'$ such that $\phi = (C, C', \phi_1, \phi_2)$ is a functor.

DEFINITION 5.4 (free category). *A category C is called* **free** *if there exists a free generating system E of C.*

We formulate now our observation above as a theorem:

THEOREM 5.1. *Let $G = (V, E)$ be a graph. Then E is a free generating system of $\mathcal{W}(G)$.*

Proof: Let $G = (V, E)$ be a graph and C an arbitrary category. Let $\phi_1 : E \to O$, $\phi_2' : E \to M$ be mappings and the following diagram commute:

$$\begin{array}{ccccc} V & \xleftarrow{Q} & E & \xrightarrow{Z} & V \\ \downarrow{\scriptstyle \phi_1} & & \downarrow{\scriptstyle \phi_2'} & & \downarrow{\scriptstyle \phi_1} \\ O & \xleftarrow{Q} & M & \xrightarrow{Z} & O \end{array}$$

We define

$$\phi_2(P, P) = 1_{\phi_1(P)}, P \in V$$

and

$$\phi_2(e) = \phi_2'(e), e \in E$$

Let $\phi_2(w)$ be defined for all $w \in \mathcal{W}(G)$ with $|w| \leq n, n \geq 1$ and ϕ_2 be compatible with Q and Z for all these paths w.

Further let ϕ_2 be uniquely determined for these paths w and it holds:

$$\phi_2(w \cdot v) = \phi_2(w) \cdot \phi_2(u)$$

for all w, v with $|w \cdot v| \leq n$.

Now let $v = (P, s_1, \ldots, s_{n+1}, P') \in \mathcal{W}(G)$. We split v into

$$v = (P, \underbrace{s_1, \ldots, s_n}_{v_1}, P'') \cdot (P'', \underbrace{s_{n+1}}_{v_2}, P')$$

.

By induction hypothesis, $\phi_2(v_1)$ and $\phi_2(v_2)$ are defined.

For ϕ_2 to become a functor, necessarily $\phi_2(v) = \phi_2(v1) \cdot \phi_2(v_2)$ must hold.

By assumption, ϕ_2 is compatible with source and target mappings Q and Z for v_1 and v_2. Therefore $Z(\phi_2(v_1)) = Q(\phi_2(v_2))$ and $\phi_2(v)$ is defined.

Let $v = u_1 \cdot u_2$ be any partition of v, so

$$v = (P, \underbrace{s_1, \ldots, s_j}_{u_1}, \bar{P}) \cdot (\bar{P}, \underbrace{s_{j+1}, \ldots, s_{n+1}}_{u_2}, P').$$

By induction hypothesis it holds:

(1) $Z(\phi_2(u_1)) = Q(\phi_2(u_2))$, so $\phi_2(u1) \cdot \phi_2(u_2)$ is defined.

(2) With $u_2' = (\bar{P}, s_{j+1}, \ldots, s_n, P'')$ we have

$$\begin{aligned} \phi_2(u_1) \cdot \phi_2(u_2) &= \phi_2(u_1) \cdot (\phi_2(u_2') \cdot \phi_2(v_2)) \\ &= (\phi_2(u_1) \cdot \phi_2(u_2')) \cdot \phi_2(v_2) \\ &= \phi_2(v_1) \cdot \phi_2(v_2) \\ &= \phi_2(v) \end{aligned}$$

It remains to show

$$\phi_1(Q(v)) = Q(\phi_2(v)), \quad \phi_1(Z(v)) = Z(\phi_2(v))$$

This follows directly from $Q(v) = Q(v_1)$ and $Z(v) = Z(v_2)$ by induction hypothesis.

THEOREM 5.2. *To each category C there exists a free category F and a surjective functor $\phi = (F, C, \phi_1, \phi_2)$.*

Proof: For the category C, create the oriented graph $G_C = (V, E)$ with $V = Obj(C)$ and $E = \{f \mid Q(f) = O_1, Z(f) = O_2, f : O_1 \to O_2 \in M\}$.

That means: The objects of the category become the points of the graph and each morphism becomes an edge between the corresponding source and target objects. Because $\mathcal{W}(G_C)$ is a free category by theorem 1 we can choose F to be exactly this category.

Let $\phi_1 : V \to O$ with $\phi_1(w) = w$ and $\phi_2' : E \to M$ with $\phi_2'(f) = f$ be mappings and $\phi_2 : \mathcal{W}(G_C) \to M$ be the continuation of ϕ_2' such that $\phi = (\mathcal{W}(G_C), C, \phi_2, \phi_2)$ becomes a functor.

By construction, ϕ is surjective.

The following theorem tells about the uniqueness of free generating systems.

THEOREM 5.3. *If E and E' are free generating systems of a category F, then $E = E'$.*

The proof is similar to the one of the corresponding theorem for free monoids.

6. Grammars and derivations

In the introduction of this book we already learned about formal languages. The wish to describe these in general infinite sets of words by a finite generating system leads to the notion of a **grammar**.

DEFINITION 6.1 (Chomsky grammar). *$G = (N, T, P, s)$ is a* **Chomsky grammar**, *if*

(1) *N is a finite, nonempty set of* **nonterminal symbols**
(2) *T is a finite, nonempty set of* **terminal symbols** *with $N \cap T = \emptyset$*
(3) *$P \subset N^+ \times (N \cup T)^*$ is a finite set of* **productions**
(4) *$S \in N$ is the* **axiom** *or* **start symbol**

Notation: For $p = (u, v) \in P$ we also write $u \xrightarrow{p} v$ and $Q(p) = u, Z(p) = v$ denote the source and target of a production.

Examples:

(1) $G_1 = (N, T, P, S)$ with $N = \{S\}, T = \{x, x'\}$,
$P = \{S \to SS, S \to xSx', S \to \epsilon\}$
(2) $G_2 = (N, T, P, S)$ with $N = \{S, X\}, T = \{x, x'\}$
$P = \{S \to xSX, S \to xX, X \to x'S, X \to x'\}$

To use a grammar for generating the words of a language, starting with the axiom there are intermediate words generated by application of the productions, until the produced word will contain terminal symbols only. This leads to the notation of a "derivation" which we will now define formally.

DEFINITION 6.2 (directly derivable, derivable). *Let $G = (N, T, P, S)$ be a grammar and let $w, w' \in (N \cup T)^*$.*

w' is **directly derivable** *from w in G, notation: $w \Rightarrow_G w'$, if there are segmentations $w = w_1 \cdot u \cdot w_2$ and $w' = w_1 \cdot v \cdot w_1$ and a production $(u, v) \in P$.*

w' is **derivable** *from w, notation: $w \overset{*}{\Rightarrow}_G w'$, if there exists a sequence of words*

$$w = w_0, \ldots, w_n = w', \quad n \in \mathbb{N}, w_i \in (N \cup T)^*$$

such that for each $0 \leq i \leq n : w_i \Rightarrow_G w_{i+1}$.

Such a sequence is called a **derivation** of length n. Q and Z can be extended in a natural way to derivations.

A derivation is called **canonic** or **leftmost**, if in each step $w \Rightarrow_G w'$ it holds:

If $w = w_1 \cdot u \cdot w_2$ and $w' = w_1 \cdot v \cdot w_2$ are the segmentations and $(u, v) \in P$ the applied production, then $w_1 \in T^*$ which means that always the leftmost nonterminal is replaced.

If the grammar is known we omit the index G from the symbols $\Rightarrow_G$ and $\overset{*}{\Rightarrow}_G$.

Let us consider some properties of the relation $\overset{*}{\Rightarrow}$:

LEMMA 6.1. *Let G be a grammar. Then the following holds:*

(1) $(u, v) \in P \Rightarrow u \overset{*}{\Rightarrow}_G v$

(2) $w \overset{*}{\Rightarrow}_G w$ *(reflexivity)*

(3) $w \overset{*}{\Rightarrow}_G w' \wedge w' \overset{*}{\Rightarrow}_G w'' \Rightarrow w \overset{*}{\Rightarrow}_G w''$ *(transitivity)*

(4) $w_1 \overset{*}{\Rightarrow}_G w_1' \wedge w_2 \overset{*}{\Rightarrow}_G w_2' \Rightarrow w_1 \cdot w_2 \overset{*}{\Rightarrow}_G w_1' \cdot w_2'$ *(compatibility with monoid operation)*

Here $w, w', w'', w_1, w_2, w_1', w_2' \in (N \cup T)^*$.

Proof:

(1) follows from the definition of $\overset{*}{\Rightarrow}_G$.

(2) clear with $n = 0$ in the definition of $\overset{*}{\Rightarrow}_G$.

(3) There exist sequences $w = w_0, \ldots, w_n = w'$, $w' = w_0', \ldots, w_m' = w''$ with $w_i \overset{*}{\Rightarrow}_G w_{i+1}$ and $w_j' \overset{*}{\Rightarrow}_G w_{j+1}'$. Because $w_n = w' = w_0'$, the composed sequence $w = w_0, \cdots, w_n, w_1', \cdots, w_n' = w''$ is a derivation from w to w''.

(4) Exercise for the reader

Notation: An intermediate word that is generated by a derivation starting with the axiom is called a **sentence form** of G. We define:

DEFINITION 6.3.

$$SF(G) := \{w \in (N \cup T)^* \mid S \overset{*}{\Rightarrow}_G w\}$$

is the set of **sentence forms** *of G.*

Now we are able to define the formal language generated by a grammar.

DEFINITION 6.4. *Let $G = (N, T, P, S)$ be a grammar.*

$$L(G) := \{w \in T^* \mid S \overset{*}{\Rightarrow}_G w\}$$

is the **language generated by** *G.*

Note: $L(G) = SF(G) \cap T^*$.

Examples:

(1) One can see that for the grammar G_1 given in the previous example 1 it holds: $L(G_1)$ is the Dyck language over the alphabet $\{x, x'\}$.

(2) Let $G = (N, T, P, S)$ with $N = \{S\}, T = \{a, b\}, P = \{S \to aSb, S \to \epsilon\}$. Then $L(G) = \{a^n b^n \mid n \in \mathbb{N}$.

The simple proof is left to the reader.

Grammars are compared with relation to the languages they generate. We define:

DEFINITION 6.5 (weak grammar equivalence). *G is* **weakly equivalent** *to $G' \Leftrightarrow L(G) = L(G')$.*

Remark: The reader should convince himself that the grammars G_1 and G_2 from the first example generate the same language.

Of course you can define infinitely many different grammars for each language.

We now can define different classes of grammars (and languages generated by these) depending on certain restrictions of their production system. In the next chapter we will meet the so called *right-linear* grammars and their languages.

Special importance, also from a practical point of view, have the so called *contextfree* grammars.

DEFINITION 6.6 (context-free grammar). *A grammar* $G = (N, T, P, S)$ *is called* **context-free** *if* $P \subset N \times (N \cup T)^*$.

The term "context-free" describes the fact that in a sentence-form a nonterminal may be replaced by the right-hand side of a production without need to respect the "context" to the left and right around that nonterminal symbol.

In chapter 4 we will treat context-free grammars in depth.

CHAPTER 2

Finite Automata

1. The finite automaton, regular sets in X^*, $REG(X^*)$

Let $G = (V, E)$ be a finite, oriented graph, X a finite set and $\alpha = (\mathcal{W}(G), X^*, \alpha_1, \alpha_2)$ a functor with $\alpha_1 : V \to \{X^*\},\ \alpha_2 : \mathcal{W}(G) \to X^*$.

(Remark by the translator: Here the free monoid X^* is regarded as a category $X^* = (\{X^*\}, X^*, Q, Z, \cdot\})$ where $\cdot$ is the monoid operation (word concatenation). Words are treated as morphisms with source and target X^*. Klingt komisch, is aber so.)

DEFINITION 1.1 (nondeterministic finite automaton). $\mathcal{A} = (G, X^*, \alpha)$ *is called a* **nondeterministic finite automaton**.

If $S, F \in V$ are points of the graph G, we call $\mathcal{A} = (G, X^*, S, F, \alpha)$ a finite automaton with start and final states or shortly a **finite acceptor**.

In the following we will use the terms acceptor and automaton as synonyms.

If the finite automaton works over the free monoid X^* we write shorter just X instead of X^*, otherwise we specify the monoid explicitly.

DEFINITION 1.2 (accepted set). *If we define*

$$\mathcal{W}(G)(S, F) := \{w \in \mathcal{W}(G) \mid Q(w) \in S \wedge Z(w) \in F\},$$

then

$$L_{\mathcal{A}} := \alpha_2(\mathcal{W}(G)(S, F))$$

is called the set **accepted by** *the automaton.*

We will also write shortly α instead of α_2 and $\mathcal{W}()(S, F)$ instead of $\mathcal{W}(G)(S, F)$.

DEFINITION 1.3 (regular language over free monoid). *Let X be an alphabet.*

$$REG(X^*) := \{L \subset X^* \mid \textit{there exists a finite automaton } \mathcal{A} \textit{ with } L = L_{\mathcal{A}}\}$$

$REG(X^*)$ is the set of **regular languages** *over the free monoid X^*.*

Remark: We defined here the finite automaton via its "state graph". Most often, the definition is given using the "next state relation" as follows:

$$\delta = \{(a, P_1, P_2) \in X \times V \times V) \mid$$
$$\text{there exists an edge } e \text{ with } Q(e) = P_1, Z(e) = P_2 \text{ and } \alpha(e) = a \in X\}.$$

δ my be regarded as a relation between $X \times V$ and V where X is the input alphabet and V the state set of the automaton

$$\mathcal{B} = (X, V, \delta, S, F)$$

The elements of V denote the current state of the automaton $\mathcal{B}$.

If the automaton $\mathcal{B}$ is in state $z \in V$ and reads the symbol $x \in X$ then it changes into state $z' \in V$ where $(x, z, z') \in \delta$. If there doesn't exist such a z' the automaton halts.

This interpretation can be visualized as follows:

FIGURE

Let's return to out definition of the finite automaton. We explain its working based on our definition:

The automaton $\mathcal{A} = (G, X, S, F, \alpha)$ may be interpreted as a nondeterministic algorithm. The points of graph G define the possible states of the algorithm, the elements of X are the input alphabet.

The nondeterministic automaton $\mathcal{A}$ which reads a symbol $x \in X$ while in state $P \in V$ changes into state P' if there exists an edge e from P to P' with label $\alpha(e) = x$. If the graph has no such edge originating in P the automaton is set "out of service".

A finite acceptor accepts a word $w \in X^*$ if there exists a path from a point in S to a point in F which is labeld with w.

Let's consider some examples for finite automata:

Example 1: Let $X = \{a, b\}$ and $L = \{(ab)^{2n} \mid n \in \mathbb{N}\}$. It holds: $L \in REG(\{a, b\}^*)$.

The following acceptor accepts L (exercise):

$\mathcal{A} = (G, \{a, b\}, 1, 1, \alpha)$.

FIGURE

Example 2: Lexical analysis, check for special characters.

In every programming language there exist special character combinations (reserved words) that mark certain program actions. These have to be identified during the lexical analysis. We give a finite acceptor which realizes such a check for a selection of reserved words:

Let the set of reserved words be { 'BEGIN', 'END', 'ELSE', 'IF', FI', 'FOR', 'INTEGER', 'THEN', 'LOOP', 'POOL', 'PROCEDURE' }.

The following acceptor accepts this set:

FIGURE

The images of the edges under the mapping α are shown as edge labels. The points of the graph are the ovals with their labels. Start and final states are given by $S =$ { START } and $F =$ { STOP }.

The labels of the points are chosen such that one can see the information stored by the automaton.

Now we want to prove some properties of $REG(X^*)$. To do that, we need some basic properties for finite automata.

LEMMA 1.1. *Let $\mathcal{A} = (G, X, S, F, \alpha)$ be a finite automaton. Then there exists a finite automaton $\mathcal{A}' = (G', X, S', F', \alpha')$ such that $card(S') = card(F') = 1$ and $L_{\mathcal{A}} = L_{\mathcal{A}'}$.*

An automaton with a single start state is called **initial**.

Proof: If $card(S) = card(F) = 1$ we are done.

(Comment by translator: The proof in the book is completely unreadable because of all these tildes, primes, indices etc. Therefore it is reformulated here.)

Let $card(S) > 1$ or $card(F) > 1$.

1. Add new edges leaving the new start state S':

Define the set of all edges leaving an old start state by

$$OUT := \{e \in E \mid Q(e) \in S\}$$

Add the following new edges to the graph:

$$OUT' := \{e' = (S', Z(e)) \mid e \in OUT,\ \alpha'(e') := \alpha(e)\}$$

2. Add new edges reaching the new final state F':

Define the set of all edges reaching an old final state by

$$IN := \{e \in E \mid Z(e) \in F\}$$

Add the following new edges to the graph:

$$IN' := \{e' = (Q(e), F') \mid e \in IN,\ \alpha'(e') := \alpha(e)\}$$

To each new edge we assign the same label as the edge from which it has been derived.

The new automaton $\mathcal{A}' = (G', X, \{S'\}, \{F'\}, \alpha')$ is defined by the graph $G' = (V \cup \{S', F'\}, E \cup OUT' \cup IN')$ and the new labeling α' which is identical to α for all existing edges and is defined as shown above for the new edges.

It is easily shown that $L_{\mathcal{A}'} = L_{\mathcal{A}}$.

LEMMA 1.2. *Let $\mathcal{A} = (G, X, S, F, \alpha)$ be a finite automaton. Then there exists an automaton $\mathcal{A}' = (G', X, S', F', \alpha')$ with $\alpha'(e) \in X\ \forall e \in E(G)$ and $L_{\mathcal{A}} = L_{\mathcal{A}'}$.*

Proof: We "split" all edges according to their labels.

(1) Let $e \in E$ with $\alpha_2(e) = x_1 \cdots x_k,\ k > 1, x_i \in X$.
Remove edge e and add new edges $e'_1, \ldots, e'_k$ and new points $P'_1, \ldots, P'_{k-1}$ such that $(Q(e), e'_1, \ldots, e'_k, Z(e)) \in \mathcal{W}(G')$ and define a new graph $G' = (V', E')$. The labeling of the new edges is defined by $\alpha'(e'_i) := x_i$ for $i = 1, \ldots, k$.

Then $\alpha_2(e) = \alpha'_2(e'_1) \cdots \alpha'_2(e'_k)$.

(2) Let $e \in E$ be an edge labeled with ϵ.

(a) (Remove ϵ-loops)
If $Q(e) = Z(e) : E' := E - e$

(b) (Remove ϵ-edges which cannot be continued to a longer path)
If there is not $e' \in E$ with $Q(e') = Z(e') : E' := E - e$

(c) (Skip ϵ-edges that can be continued and remove the ϵ-edge)
If there exists an edge $e' \in E$ with $Q(e') = Z(e) : E'' := E - e$. Add new edges: $E' := E'' \cup \{\tilde{e} \mid Q(\tilde{e}) = Q(e), Z(\tilde{e}) = Z(e'),\ alpha'(\tilde{e}) := \alpha(e')\}$

If in step (b) or (c) the target of the edge is a final state, then add the source of the edge to the set of final states.

Continue this algorithm inductively until no more ϵ-edges remain in the graph. The algorithm terminates because the point and edge sets are finite. For the new automaton $\mathcal{A}'$ that results from this algorithm holds: $L_{\mathcal{A}'} = L_{\mathcal{A}}$.

If we apply this algorithm to our automaton from example 1, we obtain:

FIGURE

Now we want to prove some closure properties of $REG(X^*)$.

THEOREM 1.1. *Let $L, L' \in REG(X^*)$. Then the union $L \cup L'$ and the intersection $L \cap L'$ both are regular.*

Proof: Let $L = L_{\mathcal{A}}$ and $L' = L_{\mathcal{B}}$ with automata

$$\mathcal{A} = (G_A, X, S_A, F_A, \alpha)$$

and

$$\mathcal{B} = (G_B, X, S_B, F_B, \beta).$$

We may assume that the edge and point sets of both automata graphs are disjoint.

(1) Closure under union: Define

$$\gamma_2(e) := \begin{cases} \alpha_2(e), & e \in E(G_A) \\ \beta_2(e), & e \in E(G_B) \end{cases}$$

Then the automaton $\mathcal{C} = (G_A \cup G_B, X, S_A \cup S_B, F_A \cup F_B, \gamma)$ accepts the language $L_{\mathcal{A}} \cup L_{\mathcal{B}}$.

(2) Closure under intersection: Define $G' = (V', E')$ where

$$V' = V_A \times V_B$$

$$E' = \{(e_A, e_B) \in E_A \times E_B \mid \alpha_2(e_A) = \beta_2(e_B)\}.$$

By lemma 2 we may assume that the edge labels are all single symbols from X.

We define the new labeling δ_2 by

$$\delta_2 : E' \to X, \ \delta_2((e_A, e_B) = \alpha_2(e_A)).$$

For the automaton $\mathcal{A}' = (G', X, S_A \times S_B, F_A \times F_B, \delta)$ then holds: $L_{\mathcal{G}'} = L_{\mathcal{A}} \cap L_{\mathcal{B}}$ and this automaton is called the **cartesian product** of $\mathcal{A}$ and $\mathcal{B}$.

THEOREM 1.2. *Let $L \in REG(X^*) \Rightarrow L^R \in REG(X^*)$ where L^R is the mirror language of L.*

Proof: Let $\mathcal{A} = (G, X, S_{\mathcal{A}}, F_{\mathcal{A}}, \alpha)$ be a finite acceptor for L.

Create the graph $G' = (V(G), E')$ where E' is a set with the same cardinality as E. There exists a bijection from E to E' mapping edges as follows:

$$e : P \to P' \in E \Leftrightarrow e' : P' \to P \in E'$$

which means we reverse the orientation of the edges.

For the resulting automaton $\mathcal{A}' = (G', X, S_{\mathcal{A}}, F_{\mathcal{A}}, \alpha')$ with $\alpha'(e') = \alpha(e)$ for all edges of G' it holds: $L_{\mathcal{A}'} = {L_{\mathcal{A}}}^R = L^R \in REG(X^*)$.

DEFINITION 1.4 (deterministic and complete automaton). *Let $\mathcal{A} = (G, X, S, F, \alpha)$ be a finite automaton with $\alpha(E(G)) \subset X$ (each edge is labeled with a single symbol).*

$\mathcal{A}$ is called **deterministic** $\Leftrightarrow$ *for all $e, e' \in E(G)$ with $Q(e) = Q(e')$ and $\alpha(e) = \alpha(e')$ it holds: $e = e'$.*

$\mathcal{A}$ is called **complete** $\Leftrightarrow$ *for each $P \in V(G)$ and $x \in X$ there exists an edge $e \in E(G)$ with $Q(e) = P$ and $\alpha(e) = x$.*

THEOREM 1.3. *If $\mathcal{A}$ is a finite automaton, then there exists a complete, deterministic automaton $\mathcal{A}'$ which accepts the same language.*

Proof: From our lemmata we may assume that $card(S_{\mathcal{A}}) = 1$ and $\alpha_2(e) \in X$ for all edges e in the graph of $\mathcal{A}$.

We construct an automaton $\mathcal{A}'$ as follows ("subset construction"):

Instead of the points of the graph G our new graph has the power set $Pot(V(G))$ of $V(G)$ as its point set.

For $\tilde{P} \in Pot(V(G))$ we define

$$N(x, \tilde{P}) := \{P \in V(G) \mid \text{there exists an edge } e \in E(G) \text{ with } Q(e) \in \tilde{P}, Z(e) = \tilde{P} \text{ and } \alpha(e) = x\}$$

The empty set $\emptyset$ is also an element of the power set such that it will also become a point of the new graph.

Let $\tilde{P}, \tilde{R} \in Pot(V(G))$. These points are connected by an edge e with label $\alpha'_s(e) = x \Leftrightarrow \tilde{R} = N(x, \tilde{P})$.

This defines our new graph $G' = (V', E')$.

The start and final states are defined as follows:

$$S_{\mathcal{A}'} = \{S_{\mathcal{A}}\}$$

$$F_{\mathcal{A}'} = \{\tilde{R} \in Pot(V(G)) \mid \tilde{R} \cap F_{\mathcal{A}} \neq \emptyset\}$$

This completes the definition of automaton $\mathcal{A}' = (G', X, S_{\mathcal{A}'}, F_{\mathcal{A}'}, \alpha')$.

We first prove that $\mathcal{A}'$ is complete and deterministic:

$\mathcal{A}'$ has a single start state $S_{\mathcal{A}'} = \{S_{\mathcal{A}}\}$, the edge labels are all single symbols and

$$card(\{e \in E' \mid Q(e) = \tilde{P}, \alpha'_2(e) = x\}) = 1$$

for all $\tilde{P} \in V(G')$.

Now we prove that the accepted languages are equal:

"$\subset$": We use diagrams of the following form:

$$\begin{array}{c} P \xrightarrow{x} R \\ \tilde{P} \xrightarrow{x} \tilde{R} \end{array}$$

which are to be understood as follows: For $P \xrightarrow{x} R \in E(G)$ there exists by construction $\tilde{P} \xrightarrow{x} \tilde{R} \in E(G')$ with $P \in \tilde{P}$ and $R \in \tilde{R}$.

Starting with $S_{\mathcal{A}'} = \{S_{\mathcal{A}}\} = \{\{P_0\}\}$ and concatenating these diagrams, we get for $x_1 \cdots x_k \in L_{\mathcal{A}}$:

$$\begin{array}{l} P_0 \xrightarrow{x_1} P_1 \xrightarrow{x_2} P_2 \longrightarrow \ldots \longrightarrow P_{k-1} \xrightarrow{x_k} P_k \in F_{\mathcal{A}} \\ \tilde{P}_0 \xrightarrow{x_1} \tilde{P}_1 \xrightarrow{x_2} \tilde{P}_2 \longrightarrow \ldots \longrightarrow \tilde{P}_{k-1} \xrightarrow{x_k} \tilde{P}_k \end{array}$$

That means $\tilde{P}_k \cap F_{\mathcal{A}} \neq \emptyset \Rightarrow \tilde{P}_k \ \in F_{\mathcal{A}'} \Rightarrow w \in L_{\mathcal{A}'}$.

$$\Rightarrow L_{\mathcal{A}} \subset L_{\mathcal{A}'}$$

"$\supset$": Here we use diagrams

$$\begin{array}{c} \tilde{P} \xrightarrow{x} \tilde{R} \\ P \xrightarrow{x} R \end{array}$$

which are to be understood as follows: For $\tilde{P} \xrightarrow{x} \tilde{R} \in E(G')$ and $R \in \tilde{R}$ there exists $P \in \tilde{P}$ with $P \xrightarrow{x} R \in E(G)$.

For $x_1 \cdots x_K \in L_{\mathcal{A}'}$ we start with $F_{\mathcal{A}'}$ and continue the diagram from right to left. For $P_k \in \tilde{P}_k \cap F_{\mathcal{A}}$ we get

$$\begin{array}{rl} S_{\mathcal{A}'} \ni & \tilde{P}_0 \xrightarrow{x_1} \tilde{P}_1 \xrightarrow{x_2} \tilde{P}_2 \longrightarrow \ldots \longrightarrow \tilde{P}_{k-1} \xrightarrow{x_k} \tilde{P}_k \in F_{\mathcal{A}'} \\ & P_0 \xrightarrow{x_1} P_1 \xrightarrow{x_2} P_2 \longrightarrow \ldots \longrightarrow P_{k-1} \xrightarrow{x_k} P_k \in F_{\mathcal{A}} \end{array}$$

Because of $S_{\mathcal{A}'} = \{S_{\mathcal{A}}\} = \{\{P_0\}\}$ it holds: $x_1 \cdots x_k \in L_{\mathcal{A}} \Rightarrow L_{\mathcal{A}'} \subset L_{\mathcal{A}}$.

$$\Rightarrow L_{\mathcal{A}'} \subset L_{\mathcal{A}}$$

From both inclusions we get $L_{\mathcal{A}} = L_{\mathcal{A}'}$.

Remark: The state set grows exponentially when the automaton is made deterministic. The following example will clarify this fact [**?**].

Example: Let $X = \{a, b\}$. Define for arbitrary, but fixed $n \in \mathbb{N}$

$$L_n := \{w \mid w = w_1 \cdot w_2 \text{ with } w1 \neq w_2, |w_1| = |w_2| = n, w_1, w_2 \in X^*\}$$

$L_n \in REG(X^*)$ because there exists a nondeterministic finite acceptor $\mathcal{A}_n$ with $L_n = L_{\mathcal{A}_n}$.

We want to give the automaton for $n = 3$:

$\mathcal{A}_3 = (G_3, \{a, b\}, \{1\}, \{22\}, \alpha)$ with the following graph G_3:

FIGURE

The labeling α is shown at the edges.

Exercise: Show that this automaton accepts L_3.

If we construct for L_3 from the nondeterministic acceptor $\mathcal{A}_3$ the deterministic acceptor $\mathcal{A}'_3$, the graph G'_3 looks as follows:

FIGURE

The edges pointing upwards are labeled with a and the edges pointing downwards with b.

G'_3 is the graph of the complete (up to loops in the final state and state $\emptyset$), deterministic acceptor $\mathcal{A}'_3$ and accepts the language L_3 (exercise).

In [**?**] it is proved that this automaton is minimal in the number of states.

Making a state graph deterministic brings advantages as well as disadvantages. A program simulating finite automata based on the given state graph is fast for a deterministic graph. However it needs more space because of the possibly very large representation of the deterministic graph.

If the program uses the nondeterministic graph, it can follow all alternatives in parallel similar to our construction of the deterministic automaton. It needs less memory but the computing time can grow linear for each simulation step.

We want to state an important consequence from our last theorem.

THEOREM 1.4. *Let $L \in REG(X^*) \Rightarrow \bar{L} \in REG(X^*)$ where $\bar{L}$ denotes the complement of L.*

Proof: $L \in REG(X^*) \Rightarrow$ there exists a complete, deterministic finite acceptor $\mathcal{A}$ with $L_{\mathcal{A}} = L$.

Define $\mathcal{A}' = (G, X, S_{\mathcal{A}}, F_{\mathcal{A}}, \alpha)$ where $F_{\mathcal{A}} := V(G) - F_{\mathcal{A}}$.

Because $\mathcal{A}$ is complete and deterministic, every word $w \in X^*$ determines a unique path starting in $S_{\mathcal{A}}$.

This $w \in X^*$ uniquely determines a point $P \in V(G)$ with $w \in \alpha(\mathcal{W}(W)(S_{\mathcal{A}}, P))$.

It is either $P \in F_{\mathcal{A}}$ or $P \in F_{\mathcal{A}'}$ and from $F_{\mathcal{A}} \cap F_{\mathcal{A}'} = \emptyset$ follows $w \in L_{\mathcal{A}} \Leftrightarrow w \notin L_{\mathcal{A}'}$ and therefore $L_{\mathcal{A}'} = \bar{L}_{\mathcal{A}}$.

THEOREM 1.5. *Let $L_1, L_2 \in REG(X^*) \Rightarrow L_1 \cdot L_2 \in REG(X^*)$.*

Proof: $L_1, L_2 \in REG(X^*) \Rightarrow$ there exist acceptors $\mathcal{A}_1$ and $\mathcal{A}_2$ with $L_1 = L_{\mathcal{A}1}$ and $L_2 = L_{\mathcal{A}2}$.

We define a new acceptor $\mathcal{A} = (G, X, S, F, \alpha)$ as follows:

$$V(G) = V(G_{\mathcal{A}_1}) \cup V(G_{\mathcal{A}_2}) \text{ where } V(G_{\mathcal{A}_1}) \cap V(G_{\mathcal{A}_2}) = \emptyset$$

$$E(G) = E(G_{\mathcal{A}_1}) \cup E(G_{\mathcal{A}_2}) \cup \text{ the following edges:}$$

For $P \xrightarrow{e} P' \in E(G_{\mathcal{A}_1}), R \in F_{\mathcal{A}_1}, \alpha(e) = x, P' \in S_{\mathcal{A}_2}$ there is an edge $P \xrightarrow{e'} P'$ with $\alpha(e') = x$ added to $E(G)$.

FIGURE

We set $S = S_{\mathcal{A}_1}$ and $F = F_{\mathcal{A}_2}$.

We show now that the automaton defined in this way accepts the language $L_{\mathcal{A}1} \cdot L_{\mathcal{A}2}$.