Formal Languages
An automata-theoretic introduction

Original title (in German)
Formale Sprachen
Eine automatentheoretische Einführung
Reihe Informatik, Band 35
Bibliographisches Institut AG, Zürich, 1981

Prof. Dr. Günter Hotz
Universität des Saarlandes

Dr. Klaus Estenfeld
Universität des Saarlandes

English translation by
Armin Reichert
Alumnus Universität des Saarlandes
(Version: March 9, 2024)

Contents

Preface

This book is in principle the second edition of the book Hotz/Walter: "Automatentheorie und Formale Sprachen II, Endliche Automaten" (engl. "Automata theory and formal languages II, finite automata"), Bibliographisches Institut, Mannheim, 1969.

The book however has been reworked so extensively that it really can count as completely new. While in the first edition only the theory of finite automata had 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 by developing the theory in an automata-theoretic way.

Composing the theory in such a way had already been proposed by Goldstine [**Gol77**] in 1977 and sketched by him in various talks. The motivation for developing my lecture from which this book originates in that way is however not based on Goldstine's idea.

The composition here arose almost automatically out of dealing with the works of the French school[1]. I must emphasize here the book [**Ber79**] by Jean Berstel on transductions. Mr. Berstel finally pointed me to the work of Goldstine. I fully support Goldstine's opinion that it would be worth rethinking the whole theory of formal languages along these 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 compiler design.

Dr. Klaus Estenfeld worked out my lecture "Formal Languages I" which I gave on this topic in the winter of 1980/81 as the foundation of this book and made 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 this second edition.

Saarbrücken, August 1981 *Günter Hotz*

[1] "*French school*" refers to the works of Schützenberger, Nivat, Perrot, Berstel, Sakarovitch etc. Berstel's book on transductions which is cited very often in the literature was based on lectures of Berstel in Paris and also in Saarbrücken where he spent some months in 1978 on invitation of 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 the specification 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* which represent conceptual generalizations of a computer.

These generalizations usually are easier to grasp than deterministic algorithms which contain more details that do not reflect the original problem but are determined by the need for unambiguosly defining the algorithm. This is part of the reason why it is difficult to prove the correctness of programs in a clear way. Correctness proofs for *grammars* or for other language generation mechanisms offer a possibility to study such proofs on 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 extraordinary simplicity and beauty.

In the foreground of the theory are standing different methods for defining formal language classes, studying their word and equivalence problems and putting them into different hierarchical classifications.

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

Of course, in the context of such a pocket book one has to make a close selection of topics concerning language classes, generation processes as well as basic questions. In doing that we let us guide by the wish to keep the formal machinery as small as possible.

Because the theory of finite automata is the foundation of the whole theory of formal languages, we start our book with that topic. In developing the 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 will get realized. Our intuitive notion of a finite automaton is a *finite, oriented graph* whose edges are labeled with the symbols from the input alphabet of the automaton. Depending on the input string we look for a path in the graph labelled with that string. If the end point of a path starting from the dedicated *start point* of the automaton is one of the *end points*, our automaton *accepts* the input word and doesn't otherwise.

We prove the equivalence of this concept with the other known methods for defining finite automata. We prove also 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, that is attaching a second label at the graph's edges, one gets the *rational transducers*. A detailed theory of these general transductions can be found in the book by Berstel [**Ber79**].

We restrict ourselves here to special generalizations of the free monoid of words:

- the *free group*
- the *H-group* (the relation $x \cdot x^{-1} = 1$ holds for all x from the generating system, but not $x^{-1} \cdot x = 1$)
- the *polycyclic monoid* (in addition to $x \cdot x^{-1} = 1$ it holds $x \cdot y^{-1} = 0$ for $x \neq y$ and $0 \cdot x = x \cdot 0 = 0$ 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 composition of the theory of context-free languages leads to a simple path to the most important representation theorems including the theorems of **Chomsky-Schützenberger, Shamir and Greibach**. Also for the transformation into *Greibach normal-form* one gets a simple, efficient algorithm.

As easy as in the case of finite automata one can prove the known closure properties for context-free languages.

Finally we also prove the equivalence of this representation to the usual representation of context-free languages by context-free grammars.

Our composition of the theory is very close to the one repeatedly recommended by Goldstine since 1977 but originated independently. The difference is that we prove Greibach's representation theorem by making our automaton deterministic going from output monoids to *monoid rings*. In doing so one gets the theorem of Shamir in a natural way and from this the theorem of Greibach.

From the theorem of Shamir one can quite easily get 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 true for the treatment of the deterministic languages.

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

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

Because we judge the former fact as rather important, we want to explain it on a specific example, the so-called *Dyck language*[2].

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

[2]In the literature, the Dyck language is also defined such that corresponding brackets are symmetric. The language defined here is called *Semi-Dyck language* in that case.

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

Then it holds:

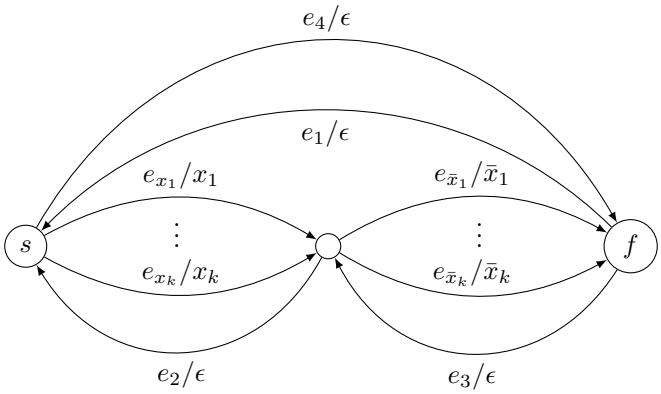
(1) $\epsilon \in D(X_k)$

(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 labellings of paths from vertex s to vertex f we get of course also words not contained in $D(X_k)$ like 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 paths of the graph into the polycyclic monoid over $X_k \cup \bar{X}_k$ such that the homomorphic images of the paths from s to f have a special form, for example being the unit of the polycyclic monoid.

If we consider the word $x_1x_2\bar{x}_2\bar{x}_1x_2\bar{x}_2 \in D(X_2)$ then we have different paths

$$e_{x_1}e_2e_{x_2}e_{\bar{x}_2}e_3e_{\bar{x}_1}e_1e_{x_2}e_{\bar{x}_2}$$

and

$$e_{x_1}e_2e_{x_2}e_{\bar{x}_2}e_3e_{\bar{x}_1}e_3e_2e_{x_2}e_{\bar{x}_2}$$

which both have this word as label and we can easily define a homomorphism in the sense given above fulfilling the condition.

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

The problem of constructing 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*[3].

[3]Finite automata with storage monoids have regained interest in current research on computational algebra and automata theory, see for example [**Kam09**], [**RK09**], [**Zet16**].

CHAPTER 1

Mathematical Foundations

1. Notations, basic terminology

In this first section we want to informally define the basic terminology that is used throughout the whole book. We use the usual notations

$$\begin{aligned} \mathbb{N} &= \{0,1,2,\ldots\} \text{ for the natural numbers} \\ \mathbb{Z} &= \{\ldots,-2,-1,0,1,2,\ldots\} \text{ for the integer numbers} \\ \mathbb{Q} &= \{\tfrac{a}{b} \mid a,b \in \mathbb{Z},\ b \neq 0\} \text{ for the rational numbers} \end{aligned}$$

For the operations on sets we use $\cup$ for the union and $\cap$ for the intersection. Also $A \subset B$, $a \in A$, $a \notin A$, $\bar{A}$, $A \setminus B$, $A \times B$ and $\emptyset$ have their usual meaning.

For the power set of a set A we write $\mathfrak{P}(A)$. $\mathrm{card}(A)$ denotes the cardinality of A.

Logical implication is denoted by $\Rightarrow$.

Mappings are denoted by $f : A \to B$ in which case f is a total mapping. We write $Q(f) = A,\ Z(f) = B$.[1]

If $f : A \to B,\ g : B \to C$ are mappings, then $f \circ g : A \to C$ is the **composition** which one gets by applying f first and then g:

$$(f \circ g)(a) = g(f(a))$$

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.

$$R_f = \{(a,b) \mid b = f(a)\} \subset A \times B$$

is the relation **induced by** the mapping f or also 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 $f\,|_{A_1} = g$, in words: f **restricted to** A_1.

A **semi-group** consists of a set M and an associative operation on that set, usually written 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 the neutral element with 1_M or shortly 1. In the commutative case we often write 0 instead of 1.

For $A, B \subset M$ the set

$$A \cdot B = \{a \cdot b \mid a \in A,\ b \in B\}$$

is the **complex product** of A and B.

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

For an arbitrary subset $A \subset M$, the set A^* is the smallest submonoid of M containing A.

More specific:

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

where

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

[1] The letters Q, Z stem from the German words "Quelle" (source) and "Ziel" (target).

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 $A^+ = A^* \setminus \{1\}$ is defined for semi-groups.

A is called a **generating system** of A^* resp. A^+.

A special meaning for us has the set of **words** (**strings**) over a fixed set (alphabet) $)A$. We define words as the finite sequences of elements of A, for example sequences like (a, b, d, a, c) over the alphabet $A = \{a, b, c, d\}$.

We define

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

as the **set of words (strings) over** A.

The symbol ϵ denotes the **empty word** over A, i.e. $A^0 = \{\epsilon\}$.

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

$$v = (a_1, \dots, a_k),\ w = (a_{k+1}, \dots, a_n) \Rightarrow v \cdot w := (a_1, \dots, a_n)$$

With this operation $\text{WORD}(A)$ becomes a monoid which usually is also named A^*.

This naming 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 this:

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

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

Instead of (a) we just write 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 set $|w| := n$ and call it the **length** of w. Obviously $|w \cdot v| = |w| + |v|$ and $|\epsilon| = 0$.

(Later in this book, the notation $|w|$ is mainly used for reduced words and the string length is then denoted by length(w).)

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

$$(w \cdot v)^R = v^R \cdot w^R$$

In A^* the **cancellation rules** hold:

(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**:

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

and

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

Because of the cancellation 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 sometimes just write w instead of $\{w\}$ and w **is prefix of** v, if $w^{-1} \cdot v \neq \emptyset$.

EXERCISE 1.1. *Let X be an alphabet, $R \subset X^* \times X^*$ a relation on X^*. R is called*

- **reflexive***: $\Leftrightarrow$ for all $x \in X^*$ it holds $(x, x) \in R$*
- **symmetric***: $\Leftrightarrow$ for all $x, y \in X^*$ it holds $(x, y) \in R \Rightarrow (y, x) \in R$*
- **transitive***: $\Leftrightarrow$ for all $x, y, z \in X^*$ it holds $(x, y) \in R$, $(y, z) \in R \Rightarrow (x, z) \in R$*

Given a relation R, construct a relation R' such that

(1) *$R \subset R'$*

(2) *R' is reflexive, symmetric and transitive*

(3) *$R' \subset R''$ for all relations R'' that fulfill 1) and 2).*

R' is called the **equivalence relation** *generated by R.*

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:*

(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 easily be shown: if $M_1 \subset M$ is a submonoid of M, then its image $\phi(M_1)$ is a submonoid of S. If S_1 is a submonoid of S, then its premiage $\phi^{-1}(S_1)$ is a submonoid of M.

A monoid homomorphism $\phi : M \to S$ is called

monomorphism	if ϕ is injective
epimorphism	if ϕ is surjective
isomorphism	if ϕ is bijective
endomorphism	if M=S
automorphism	if ϕ is endo- and isomorphism

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

A homomorphism of course 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. In which case 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 a submonoid $M_1 \subset M$. In which way can ϕ_1 and ϕ_2 be different?

The answers to these questions of course depend on the structure of M_1. If $M_1 = \{1_M\}$ then ϕ is uniquely determined on M_1 but there is little information on the relation between ϕ_1 and ϕ_2.

The following two simple theorems, which can be found in introductory algebra texts, hold:

(1) If M_1 is a generating system of M and $\phi_1, \phi_2 : M \to S$ 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 of 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 in a unique way to a monoid homomorphim .*

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

A free monoid has at most one free generating system (exercise). From this fact one 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 function L is an example for a monoid homomorphism $\text{length} : A^* \to \mathbb{N}$ into the monoid of natural numbers.

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

$$\{\phi^{-1}(s) \mid s \in S\} \subset \mathfrak{P}(M)$$

form a monoid which is isomorphic to $\phi(M)$ (exercise).

We want to investigate now the following problem:

Let M be a monoid and $L \subset M$ 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 any algebra text), it is denoted by $synt_M(L)$ and constructed as follows:

DEFINITION 2.3. *Let M be a monoid and $L \subset M$. For $a, b \in M$ we define*

$$a \equiv b \pmod{L} \iff \textit{for all } u, v \in M \textit{ holds: } u \cdot a \cdot v \in L \Leftrightarrow u \cdot b \cdot v \in L$$

The relation $\equiv (L)$ is a **congruence relation**, i.e. it holds:

(1) Let $[a]_L = \{b \in M \mid a \equiv b \pmod{L}\}$. Then

$$b \in [a]_L \Rightarrow [a]_L = [b]_L$$

(2) If we define

$$[a]_L \cdot [b]_L := [a \cdot b]_L (\text{complex product})$$

then

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

becomes a monoid under this operation and the mapping

$$\phi_L : M \to synt_M(L) \text{ defined by } \phi_L(a) := [a]_L$$

is a monoid epimorphism.

We call $\equiv \pmod{L}$ the **syntactic congruence** of L and $synt_M(L)$ the **syntactic monoid** of L with respect to M.

To motivate the name *syntactic monoid* we give an example.

Let A be the alphabet for English and L the set of English sentences. One can consider two words w_1 and w_2 from L as *congruent* if they can be exchanged in each sentence. There exist words that cannot always be exchanged. In the sentence "An apple is a fruit" the word "apple" can always be exchanged by "pear" but this is not possible in the sentence "apple is spelled a p p l e".

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.

EXERCISE 2.1. *Prove the following theorem:*

THEOREM 2.1. *It is decidable if a monoid homomorphism ϕ is injective.*

Hint: Let $\phi : A^* \to B^*$ be a given monoid homomorphism, $A = \{a_1, \ldots, a_n\}$, $E = \{\phi(a_1), \ldots, \phi(a_n)\}$.

Consider

$$\begin{aligned} A_1 &:= \{u^{-1}v \in B^* \mid u, v \in E,\ u \neq v\} \text{ and then inductively} \\ A_{k+1} &:= {A_k}^{-1} \cdot E \ \cup E \ \cdot A_k \end{aligned}$$

Prove that

(1) *The construction of the A_k stops*
(2) $1 \in \bigcup_{k \geq 1} A_k \iff \phi$ *is not injective.*

COROLLARY 2.1. *For a free monoid M it is decidable if some finite subset is a free generating system of M.*

3. Special monoids and the free group

We have just seen the syntactic monoid as an example of a monoid. For the theory of syntactic monoids see [**Sal73**] and [**Per77**].

We consider now other specific monoids which we will need later on several occasions. To do that, we introduce the notion of a **generated congruence relation**.

Let A be an alphabet and

$$R = \{u_i = v_i \mid i = 1, \ldots, n,\ u_i, v_i \in A^* \text{ for all } i\}$$

a set of equations.

Then the following conditions uniquely define a congruence relation $\bar{R}$ (exercise):

(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 (1) and (2).

$\bar{R}$ is the smallest congruence relation in which all equations from R hold and is called the **congruence relation generated by** R on A^*.

The factor monoid $A^*/\bar{R}$ will also simply be named A^*/R. It holds:

Two words $u, v \in A^*$ are congruent modulo $\bar{R}$, notation: $u \equiv v \pmod{\bar{R}}$, exactly if there exists $n \in \mathbb{N},\ u_i \in A^*$ with $u_i = u_i' \cdot u_i \cdot u_i''$ such that for $i = 1, \ldots, n$:

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

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

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

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

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 introduce now a partition of $(X \cup X^{-1})^*$ wrt. to different congruence relations and look at the resulting factor monoids.

Definition 3.2.

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

is called the **H-group**[2].

We introduce an absorbing element 0 by defining:

[2]In the literature this monoid (which is not a group!) is also called the *free half-group* or the *involutive monoid*.

DEFINITION 3.3.

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

is called the **polycyclic monoid**.

Remark: Using the naming of the previous section it holds:

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

which means:

The polycyclic monoid is the syntactic monoid of the Dyck language (exercise).

DEFINITION 3.4.

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

is the **free group** *over* X.

Remark: It holds

$$D(X) = [1]_{X^{(*)}} \text{ and } D(X) = [1]_{X^{[*]}}$$

i.e. the Dyck language is the set of words from $(X \cup X^{-1})^*$ which can be reduced to the empty word using the equations of the H-group or the equations of the polycyclic monoid.

In the following we will mainly consider $X^{[*]}$, 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, iteratively replace the leftmost occurence of xx^{-1} by 1.

This process is called **reduction** and the result of the reduction process is denoted by $\rho(w)$.

One can easily prove:

LEMMA 3.1. *There exists a minimal number* $k \in \mathbb{N}$ *with* $\rho^k(w) = |w|$. *This number is called the* **reduction length**. *It holds* $\rho(|w|) = |w|$.

LEMMA 3.2. *For* $[w], [w'] \in X^{[*]}$ *holds:*

$$[w] = [w'] \iff |w| = |w'|$$

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 $w = w_1 x x^{-1} w_2$ and $w' = w_1 w_2$.

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

Induction over k:

- $k = 0$: w_1 is already reduced, thus $\rho(w) = w_1 w_2 = w'$.

- $k > 0$: It holds $\rho(w) = \rho(w_1 x x^{-1} w_2),\ \rho(w') = \rho(w_1 w_2)$. The reduction length of $\rho(w)$ by induction assumption 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 does not depend on the order of the reductions.

Therefore the reduced word for a representative of an element of $X^{[*]}$ is uniquely determined and we can speak of *the* reduced word in the following.

Remark: These results have been used in [**HM75**] to obtain a space-optimal algorithm for the analysis of the Dyck language.

Similar results also hold for the free group $F(X)$, see [**CF77**].

4. Graphs, categories and functors

Before defining graphs formally we want to explain informally 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 it, possesses (with a certain restriction) a faithful image in $\mathbb{R}^3$, see e.g. [**Wag70**]. The points of the graph become points in $\mathbb{R}^3$, the edges become connecting lines in $\mathbb{R}^3$ which pairwise do not intersect.

The restriction concerning the points of the graph is that the graph must not have more points than there exist in $\mathbb{R}^3$ (cardinality of the continuum).

The restriction concerning the edges is more essential: It tells that there is at most one edge between two points and that the graph has no loops. Loops are edges with just a single border point.

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

DEFINITION 4.1. *A* **graph** $G = (V, E)$ *consists of a non-empty set* V *of vertices (points), a set* E *of edges and a mapping*

$$\rho : E \to \mathfrak{P}(V) \text{ with } \operatorname{card}(\rho(e)) <= 2 \text{ for } e \in E$$

$\rho(e)$ *is the set of* border points *of* e.

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

DEFINITION 4.2. *A graph is called* **loop-free** *if it does not contain any loops.*

We introduce an orientation for the edges.

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

$$\rho(e) = \{Q(e), Z(e)\} \text{ for all } e \in E$$

$Q(e)$ is called the **source vertex** and $Z(e)$ the **target vertex** of e [3].

The notions of loop and line are naturally transferred to oriented graphs.

To each graph one can assign the corresponding oriented graph $\hat{G}$ by defining two edges

$$(v_1, e, v_2) \text{ and } (v_2, e, v_1)$$

for each edge e with border points v_1 and v_2 and defining

$$Q((v_1, e, v_2)) = v_1 = Z((v_2, e, v_1))$$
$$Q((v_2, e, v_1)) = v_2 = Z((v_1, e, v_2))$$

[3]The letters Q and Z stem from the German words for source ("Quelle") and target ("Ziel").

DEFINITION 4.4. *A graph $G = (V, E)$ is called* **connected** *if in the corresponding oriented graph $\hat{G}$ for each pair of points v and v' there exist edge sequences $e_1, \ldots, e_k$ with*

$$Q(e_1) = v,\ Z(e_k) = v' \text{ and } Z(e_i) = Q(e_{i+1}) \text{ for } i = 1, \ldots, k-1$$

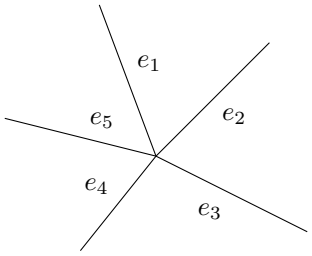
DEFINITION 4.5. *A loop-free graph $G = (V, E)$ is called* **ordered** *if for each point $v \in V$ holds:*

There exists a unique (up-to cyclic permutation) ordering on the set

$$\text{cycle}(v) := \{e \in E \mid v \in \rho(e)\}$$

The set cycle(v) is called the **cycle** that belongs to v.

Explanation: Imagine each point and its adjacent edges to be stuck on a litte disk as shown in the following figure:



In this figure the orderings for $\{e_1, e_2, e_3, e_4, e_5\}$ are:

$$e_1, e_2, e_3, e_4, e_5$$
$$e_2, e_3, e_4, e_5, e_1$$
$$\vdots$$
$$e_5, e_1, e_2, e_3, e_4$$

DEFINITION 4.6. *A loop-free, oriented graph G is called* **ordered** *if for all points $v \in V$ it holds:*

In the cycle of v there exists an ordering

$$e_1, \ldots, e_k, e'_m, \ldots, e'_1$$

such that

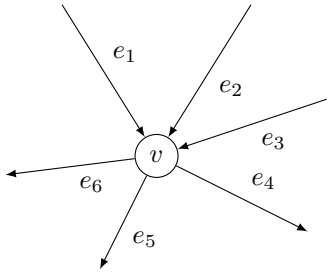
$$\{e_1, \ldots, e_k\} = \{e \in E \mid Z(e) = v\}$$

and

$$\{e'_1, \ldots, e'_m\} = \{e \in E \mid Q(e) = v\}$$

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

Example:



$e_1, e_2, e_3, e_4, e_5, e_6$ is the ordering of vertex v.

e_1, e_2, e_3 is the ordering of the incoming edges.

e_6, e_5, e_4 is the ordering of the outgoing edges.

DEFINITION 4.7. *A* **path** *in an oriented graph G is a sequence*

$$\pi = (q, e_1, \ldots, e_k, z),\ k \geq 1$$

where $e_1, \ldots, e_k \in E$ are edges and

$$\begin{aligned} Q(e_1) &= q \\ Z(e_i) &= Q(e_{i+1}) \text{ for } i = 1, \ldots, k-1 \\ Z(e_k) &= z \end{aligned}$$

We continue the source and target mappings Q and Z onto paths by defining

$$Q(\pi) := q, \quad Z(\pi) := z$$

q is called the **start point** and z the **end point** of π.

length$(\pi) := k$ is called the **length** of the path π. For $k = 0$ we define for all points $v \in V$ that $\pi = (v, v)$ is the path of length 0 from v to v.

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

DEFINITION 4.8. *Let $\pi = (q, e_1, \ldots, e_k, z)$ be a path. A path $\pi' = (q', e'_1, \ldots, e'_m, z')$ is called a* **subpath** *of π if there exists an i with $1 \leq i \leq k$ such that*

$$e'_j = e_{i+j-1},\ j = 1, \ldots, m,\ i + m - 1 \leq k$$

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

DEFINITION 4.9. *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. *Let $G = (V, E)$ be an oriented graph and $c \in V$. G is called a* **star relative to** *c if for each $v \in V$ there exists a path π_v with $Q(\pi_v) = c$ and $Z(\pi_v) = v$. c 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.

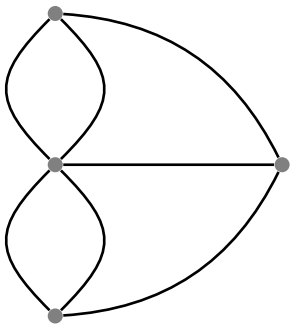
DEFINITION 4.11. *A* **tree** *is a circle-free star where for all* $v \in V$ *it holds*

$$\text{card}(\{e \in E \mid Z(e) = v\}) \leq 1$$

The following lemma holds:

LEMMA 4.1. *A tree has exactly one center (called* **root***).*

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



We want to *concatenate* paths, we define the *product of paths*:

$$(q, e_1, \ldots, e_k, z) \cdot (q', e_{k+1}, \ldots, e_n, z') := (q, e_1, \ldots, e_n, z') \text{ if } z = q'$$

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

Obviously it holds:

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

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

$$\pi \cdot 1_v = \pi \quad \text{if } Z(\pi) = v$$
$$1_v \cdot \pi = \pi \quad \text{if } Q(\pi) = v$$

We denote the set of paths of a graph G with $\mathfrak{W}(G)$. $\mathfrak{W}(G)$[4] is called the **path category** of G and in this context G is also called a **schema**. $\mathfrak{W}(G)$ is an important special case of a *category*.

Notation:

$$\mathfrak{W}(G)(v, v') := \{\pi \in \mathfrak{W}(G) \mid Q(\pi) = v,\ Z(\pi) = v'\}$$

Categories are algebraic structures with a *partial* operation.

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

[4]The letter $\mathfrak{W}$ stems from the German word "Wegekategorie" (path category).

(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 morphisms $f, g \in M$ the operation $\circ$ is defined exactly 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 them 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 $Q(f) = Z(g) = w$ holds:*

$$1_w \circ f = f \text{ and } g \circ 1_w = g$$

It can easily be shown that there exists exactly one unit morphism for each object w.

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

Historical remark: Already Euler was interested in graphs and paths in graphs (see above). The path category of a graph had already been used before the notion of a *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 [**EM45**]. Their work produced a broad, very abstract theory of categories. We will use here only 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 a category with some examples:

(1) **The category of relations**

Let O be a set of sets ($O \notin O$). Define

$$\mathrm{REL}(O) := (O, M, Q, Z, \circ)$$

by:

$$M := \{(A, B, R) \mid A \in O,\ B \in O,\ R \subset A \times B\}$$

$$Q(A, B, R) := A,\ Z(A, B, R) := B$$

$$(A, B, R_1) \circ (B, C, R_2) := (A, C, R')$$
where $R' = \{(a, c) \mid \exists\, b \in B : (a, b) \in R_1 \text{ and } (b, c) \in R_2\}$

With these definitions $\mathrm{REL}(O)$ becomes a category (exercise). The morphisms $(A, A, \{(a, a) \mid a \in A\})$ are the units for each set $A \in O$.

(2) **The category of matrices**

Define

$$\mathrm{MAT}(\mathbb{Q}) := (O.M, Q, Z, \circ)$$

by:

$O := \mathbb{N}$

$M :=$ the set of $k \times n$-matrices with entries from $\mathbb{Q}$ ($k, n \in \mathbb{N}$).

For a $k \times n$-matrix $A_{k,n}$ define source and target mapping by

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

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

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

In analogy to the monoid homomorphisms we want to introduce now structure-preserving mappings between categories called **functors**.

DEFINITION 4.13. *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 $\phi_2 : M_1 \to M_2$ 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* $Z(f) = Q(g)$.

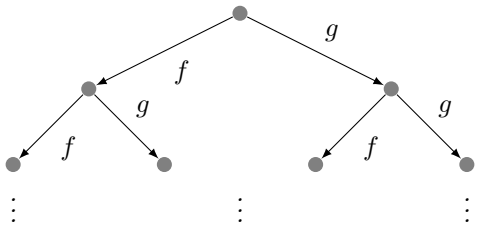
(F3) $\phi_2(1_w) = 1_{\phi_1(w)}$ *for all* $w \in O_1$.

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

Let's look at some examples:

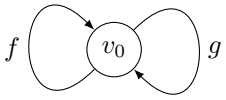
EXAMPLE 4.1. *Consider the following oriented graphs G_1 and G_2:*

G_1:



$G_1 = (V_1, E_1)$ represents an infinite binary tree. From each point of the tree two edges go out which are labelled with f and g.

G_2:



$G_2 = (V_2, E_2)$ consists of a single point v_0 and two loops labeled f and g respectively.

Consider the path categories $\mathfrak{W}(G_1)$ and $\mathfrak{W}(G_2)$. For $v \in V_1$ define $\phi_1(v) := v_0$ and $\phi_2(1_v) := 1_{v_0}$.

For the edges $e \in E_1$ define

$$\phi_2'(e) = \begin{cases} f & \text{if } e \text{ is marked with } f \\ g & \text{if } e \text{ is marked with } g \end{cases}$$

Now we define for the paths $(v, e_1, \ldots, e_n, v') \in \mathfrak{W}(G_1)$:

$$\phi_2((v, e_1, \ldots, e_n, v')) := (v_0, \phi_2'(e_1), \ldots, \phi_2'(e_n), v_0)$$

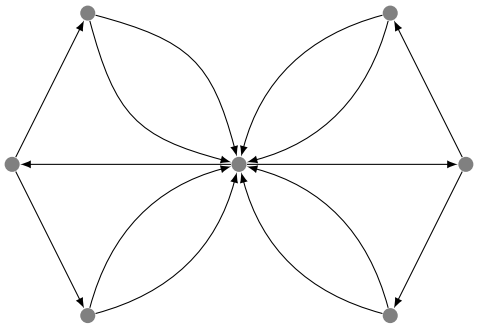
Obviously $\phi = (\mathfrak{W}(G_1), \mathfrak{W}(G_2), \phi 1, \phi_2)$ is a functor. It is even a particular functor because

(1) *ϕ is surjective*

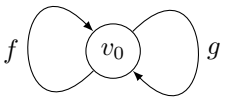
(2) *If v_1 is a point in G_1 and $\bar{\pi}$ a path in G_2 then there exists exactly one path π in G_1 with $Q(\pi) = v_1$ such that $\phi_2(\pi) = \bar{\pi}$.*

EXAMPLE 4.2. *Let graphs G_1, G_2 be given as follows:*

G_1:



G_2 :



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

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

EXAMPLE 4.3. *Let G_1 and G_2 be given as:*

$$G_1 : \; p_1 \xrightarrow{s} p_2 \qquad p_3 \xrightarrow{r} p_4$$

$$G_2 : \; q_1 \xrightarrow{f} q_2 \xrightarrow{g} q_3$$

We define:

$$\phi_1(p_1) = q_1, \;\; \phi_1(p_2) = q_2, \;\; \phi_1(p_3) = q_2, \;\; \phi_1(p_4) = q_3$$

and

$$\phi_2((p_1, s, p_2)) = (q_1, f, q_2), \;\; \phi_2((p_3, r, p_4)) = (q_2, g, q_3)$$

For the units, the definition of ϕ_2 is clear. One can see that

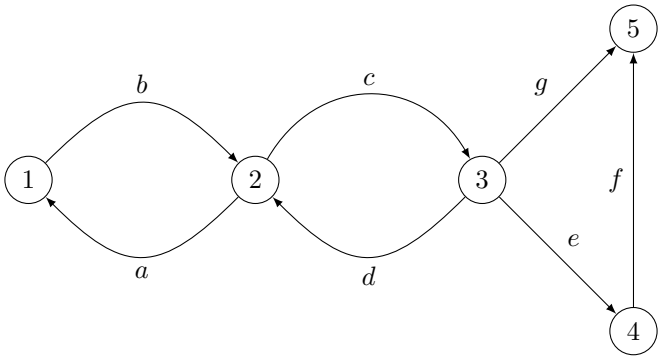
$$\phi = (\mathfrak{W}(G_1), \mathfrak{W}(G_2), \phi_1, \phi_2)$$

is a functor.

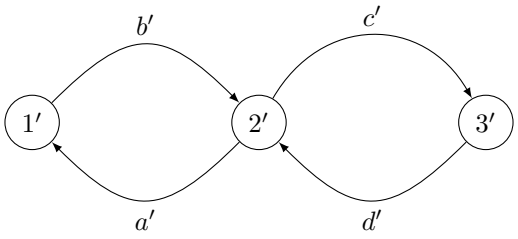
It is remarkable that $\phi_2(\mathfrak{W}(G_1))$ is not a category because this set is not closed under the $\circ$ operation.

EXAMPLE 4.4. *Let G_1 and G_2 be given as follows:*

G_1:



G_2:



We define a functor ϕ by:

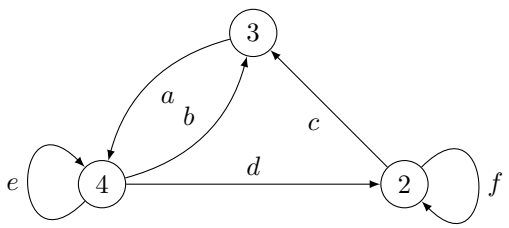
$$\phi_1(1) = 1',\ \phi_1(2) = 2',\ \phi_1(3) = 3',\ \phi_1(4) = 3',\ \phi_1(5) = 3'$$

and

$$\phi_2(a) = a',\ \phi_2(b) = b',\ \phi_2(c) = c',\ \phi_2(d) = d',$$
$$\phi_2(e) = \phi_2(f) = \phi_2(g) = 1_{3'}$$

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

EXAMPLE 4.5. *Let the graph G be defined by*



Additionally, let the following matrices be 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 the categories $\mathfrak{W}(G)$*, the path category of* G*, and* $\mathrm{MAT}(\mathbb{N})$*, the category of matrices over* $\mathbb{N}$*.*

Define

$$\phi_1(i) = i \text{ for } i = 2, 3, 4$$

and

$$\phi_2'(x) = x' \text{ for } x \in \{a, b, c, d, e, f\}$$

Then ϕ_2' *can be extended in a unique way to a mapping* $\phi_2 : \mathfrak{W}(G) \to MAT(\mathbb{N})$ *such that*

$$\phi = (\mathfrak{W}(G), \mathrm{MAT}(\mathbb{N}), \phi_1, \phi_2)$$

is a functor.

We want to define now some special properties of functors.

Definition 4.14. *Let* G_1, G_2 *be ordered graphs,* $\phi = (\mathfrak{W}(G_1), \mathfrak{W}(G_2), \phi_1, \phi_2)$ *a functor.* ϕ *is called* **ordered** *or* **order preserving** *if it holds:*

Let $\phi_1(v) = v' \in V_2$ *for any* $v \in V_1$*, then for the ordering*

$$e_1, \ldots, e_k, e'_m, \ldots, e'_1$$

which belongs to v *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 v' *in the given order.*

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

We explain this definition at the following example:

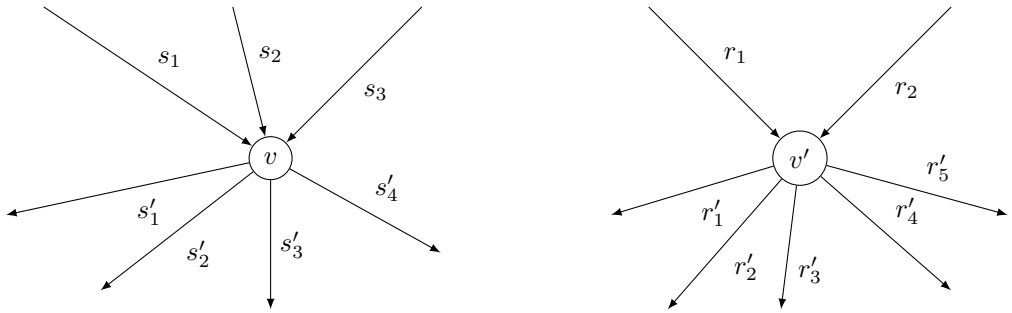
Let $v \in V_1$ be a point with ordering

$$s_1, s_2, s_3, s'_4, s'_3, s'_2, s'_1$$

and $v' \in V_2$ 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:



Define ϕ by $\phi_1(v) = v'$ and

$$\phi_2(s_1) = r_1,\ \phi_2(s_2) = r_2,\ \phi_2(s_3) = r_2$$
$$\phi_2(s'_1) = r'_1,\ \phi_2(s'_2) = r'_3,\ \phi_2(s'_3) = r'_4,\ \phi_2(s'_4) = r'_5$$

Then ϕ respects the ordering in point v.

DEFINITION 4.15. *Let* $G_1 = (V_1, E_1),\ G_2 = (V_2, E_2)$ *be oriented graphs and*

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

ϕ *is called* **regular** $\iff$ *the restriction of* ϕ_2 *to the set*

$$\{e \in E_1 \mid Q(e) = v\}\ \textit{and}\ \{e' \in E_2 \mid Q(e') = \phi_1(v)\}$$

and to

$$\{e \in E_1 \mid Z(e) = v\}\ \textit{and}\ \{e' \in E_2 \mid Z(e') = \phi_1(v)\}$$

for $v \in V_1$ *is bijective.*

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

In our example, ϕ was not regular. We slightly weaken the definition of a regular functor by postulating regularity only for the *outgoing* edges.

DEFINITION 4.16. *Let* $G_1 = (V_1, E_1),\ G_2 = (V_2, E_2)$ *be oriented graphs and*

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

ϕ *is called* **out-regular** $\iff$ *the restriction of* ϕ_2 *to the set*

$$\{e \in E_1 \mid Q(e) = v\}$$

and

$$\{e' \in E_2 \mid Q(e') = \phi_1(v)\}$$

for $v \in V_1$ *is bijective.*

The following lemma holds (exercise):

LEMMA 4.2. *If* $\phi = (\mathfrak{W}(G_1), \mathfrak{W}(G_2), \phi_1, \phi_2)$ *is an out-regular functor, then* $\phi(\mathfrak{W}(G_1))$ *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 $c \in V$ there exists a tree B and an out-regular functor $(\mathfrak{W}(B), \mathfrak{W}(G), \phi_1, \phi_2)$ mapping the root of the tree B to the point c. B is uniquely determined up to isomorphisms.*

PROOF. Exercise. □

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* $\iff$

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

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

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

In a full subcategory, all morphisms of 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 5.1. *Let $A = \{x, y, z, a, b, c\}$ and $f, g, h : A^* \to A^*$ be mappings defined as follows:*

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

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

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

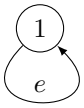
where $u_i \in (A \setminus \{y\})^*$,

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

where $u_i \in (A \setminus \{z\})^*$.

Let M be the monoid of mappings $m : A^* \to A^*$ generated by f, g and h. Then $C = (\{A^*\}, M, Q, Z, \circ)$ is a category where $\circ$ denotes the monoid operation in M.

Let G be the graph defined by the following figure:



We define a functor $\phi = (\mathfrak{W}(G), C, \phi_1, \phi_2)$ by

$$\begin{aligned} \phi_1(1) &:= A^* \\ \phi_2(e) &:= f \circ g \circ h \end{aligned}$$

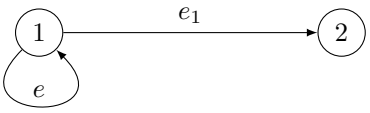
$\phi_2(\mathfrak{W}(G))$ is a subcategory of C (exercise) and it holds:

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

EXAMPLE 5.2. *We continue with example 1. In addition to f, g, h we add three monoid homomorphisms f_1, g_1, h_1 defined by:*

$$\begin{array}{lll} f_1(x) = \epsilon, & f_1(u) = u & \forall u \in A \setminus \{x\} \\ g_1(y) = \epsilon, & g_1(u) = u & \forall u \in A \setminus \{y\} \\ h_1(z) = \epsilon, & h_1(u) = u & \forall u \in A \setminus \{z\} \end{array}$$

We extend the graph G to a graph G_1:



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

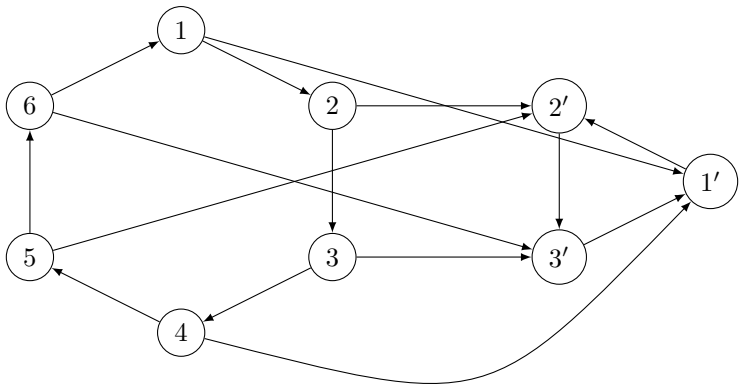
We extend the functor ϕ from example 1 onto $\mathfrak{W}(G_1)$ by defining:

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

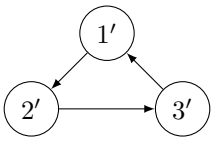
We get:

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

EXAMPLE 5.3. *Let G be defined as:*



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



One can show (exercise): The mapping ϕ_1 defined by

$$\phi_1(1) = \phi_1(4) = 1' \quad \phi_1(2) = \phi_1(5) = 2' \quad \phi_1(3) = \phi_1(6) = 3'$$

can be extended to a functor $\phi = (\mathfrak{W}(G), \mathfrak{W}(G'), \phi_1, \phi_2)$ by defining ϕ_2 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 a simple generalization of the proof for the previous lemma.

DEFINITION 5.2. *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 the* **generating system** (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.

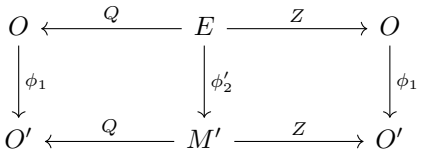
Let $G = (V, E)$ be a graph, then the edge set E is a generating system of the path category $\mathfrak{W}(G)$, so $\mathfrak{W}(G) = <E(G)>$.

The path category of a graph has a special property: the edge set is a **free generating system** of the path category $\mathfrak{W}(G)$.

DEFINITION 5.3. *Let* $C = (O, M, Q, Z, \circ)$ *be a category and* $E \subset M$.

E *is called a* **free generating system** *of* C *if it holds:*

Let $C' = (O', M', Q, Z, \circ)$ *be any category and* $\phi_1 : O \to O'$ *and* $\phi_s : E \to M'$ *be mappings which make the following diagram commute:*



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.

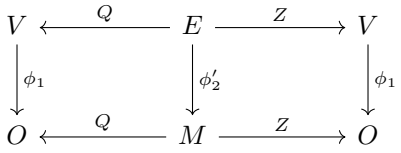
DEFINITION 5.4. *A category C is called* **free** *if there exists a free generating system E of C.*

We formulate now our observation above as a theorem:

THEOREM 5.1. *Let $G = (V, E)$ be a graph. Then*

The edge set E is a free generating system of $\mathfrak{W}(G)$.

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 let the following diagram commute:



We define

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

and

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

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

Further let ϕ_2 be uniquely determined for these paths π and it shall hold:

$$\phi_2(\pi \cdot \psi) = \phi_2(\pi) \cdot \phi_2(\psi)$$

for all paths π, ψ with length $|\pi \cdot \psi| \leq n$.

Now let $\omega = (v, e_1, \ldots, e_{n+1}, v') \in \mathfrak{W}(G)$ be a path from v to v'.

We decompose ω into paths ω_1 and ω_2

$$\omega = (v, \underbrace{e_1, \ldots, e_n}_{\omega_1}, v'') \cdot (v'', \underbrace{e_{n+1}}_{\omega_2}, v')$$

By induction hypothesis $\phi_2(\omega_1)$ and $\phi_2(\omega_2)$ are defined.

For ϕ_2 to become a functor, necessarily

$$\phi_2(\omega) = \phi_2(\omega_1) \cdot \phi_2(\omega_2)$$

must hold.

By assumption, ϕ_2 is compatible with the source and target mappings Q and Z for ω_1 and ω_2. Therefore $Z(\phi_2(\omega_1)) = Q(\phi_2(\omega_2))$ and $\phi_2(\omega)$ are defined.

Let $\omega = \psi_1 \cdot \psi_2$ be any partition of ω, so

$$\omega = (v, \underbrace{e_1, \ldots, e_j}_{\psi_1}, \bar{v}) \cdot (\bar{v}, \underbrace{e_{j+1}, \ldots, e_{n+1}}_{\psi_2}, v')$$

By induction hypothesis it holds:

(1) $Z(\phi_2(\psi_1)) = Q(\phi_2(\psi_2))$, so $\phi_2(\psi_1) \cdot \phi_2(\psi_2)$ is defined.

(2) With $\psi_2' = (\bar{v}, e_{j+1}, \ldots, e_n, v'')$ we have

$$\begin{aligned} \phi_2(\psi_1) \cdot \phi_2(\psi_2) &= \phi_2(\psi_1) \cdot (\phi_2(\psi_2') \cdot \phi_2(\psi_2)) \\ &= (\phi_2(\psi_1) \cdot \phi_2(\psi_2')) \cdot \phi_2(\omega_2) \\ &= \phi_2(\omega_1) \cdot \phi_2(\omega_2) \\ &= \phi_2(\omega) \end{aligned}$$

It remains to show

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

This follows directly from $Q(\omega) = Q(\omega_1)$ and $Z(\omega) = Z(\omega_2)$ by application of the 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 we create an oriented graph $G_C = (V, E)$ with

$$\begin{aligned} V &= Obj(C) \\ E &= \{f \mid Q(f) = O_1,\ Z(f) = O_2,\ f : O_1 \to O_2 \in M\} \end{aligned}$$

i.e the objects of the category become the points (vertices) of the graph and each morphism becomes an edge. Because $\mathfrak{W}(G_C)$ 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 $\phi_2 : \mathfrak{W}(G_C) \to M$ be the continuation of ϕ_2' such that $\phi = (\mathfrak{W}(G_C), C, \phi_1, \phi_2)$ becomes a functor.

By construction, ϕ is surjective. □

The following theorem states 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 proof of the corresponding theorem for free monoids and left to the reader.

6. Grammars and derivations

In the introduction to this book we already learned about examples of 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. *$G = (N, T, P, S)$ is a* **(Chomsky-) grammar**, *if*

(1) *N is a finite non-empty set of* **nonterminal symbols**
(2) *T is a finite non-empty 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 $Q(p) = u,\ Z(p) = v$ (*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 generate the words of a language using a grammar, starting with the axiom intermediate words are generated by multiple application of productions until the produced word contains terminal symbols only. This leads to the notion of a *derivation* that will now be formally defined.

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

w' is **directly derivable** *from w using G ($w \underset{G}{\Rightarrow} w'$) if there are decompositions*

$$w = w_1 \cdot u \cdot w_2 \text{ and } w' = w_1 \cdot v \cdot w_1$$

and there is a production $(u, v) \in P$.

w' is **derivable** *from w ($w \overset{*}{\underset{G}{\Rightarrow}} w'$) 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 $w_i \underset{G}{\Rightarrow} w_{i+1},\ 0 \le i \le n$.

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 $w \underset{G}{\Rightarrow} w'$ the following holds:

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

If the grammar is known, we omit G from the relation symbols $\underset{G}{\Rightarrow}$ and $\overset{*}{\underset{G}{\Rightarrow}}$.

We consider some properties of the relation $\overset{*}{\underset{G}{\Rightarrow}}$:

LEMMA 6.1. *Let G be a grammar. It holds:*

(1) $(u, v) \in P \;\Rightarrow\; u \overset{*}{\underset{G}{\Rightarrow}} v$

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

(3) $w \overset{*}{\underset{G}{\Rightarrow}} w' \wedge w' \overset{*}{\underset{G}{\Rightarrow}} w'' \;\Rightarrow\; w \overset{*}{\underset{G}{\Rightarrow}} w''$ *(transitivity)*

(4) $w_1 \overset{*}{\underset{G}{\Rightarrow}} w_1' \wedge w_2 \overset{*}{\underset{G}{\Rightarrow}} w_2' \;\Rightarrow\; w_1 \cdot w_2 \overset{*}{\underset{G}{\Rightarrow}} w_1' \cdot w_2'$ *(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 $\overset{*}{\underset{G}{\Rightarrow}}$.

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

(3) There exist sequences $w = w_0, \ldots, w_n = w'$, $w' = w_0', \ldots, w_m' = w''$ with $w_i \overset{*}{\underset{G}{\Rightarrow}} w_{i+1}$ and $w_j' \overset{*}{\underset{G}{\Rightarrow}} w_{j+1}'$. 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.

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

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.*

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

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

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

Examples:

(1) One can see that for the grammar G_1 from 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 respect to the languages they generate. We define:

DEFINITION 6.5. *G is* **(weakly) equivalent** *to $G' \iff L(G) = L(G')$.*

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

Of special importance, also from a practical point of view, are the so called *context-free* grammars.

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

The term *context-free* describes the fact that in a sentence-form a nonterminal symbol may be replaced by the right-hand side of a production without having to take the "context" of that symbol, i.e. the symbols to the left or right, into account.

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

CHAPTER 2

Finite Automata

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

DEFINITION 1.1. *Let $G = (V, E)$ be a finite, oriented graph, X a finite set and*

$$\alpha = (\mathfrak{W}(G), X^*, \alpha_1, \alpha_2)$$

a functor with $\alpha_1 : V \to \{X^*\},\ \alpha_2 : \mathfrak{W}(G) \to X^*$.

(The free monoid X^* is considered as the single-object category $(\{X^*\}, X^*, Q, Z, \cdot)$ with string concatenation as category operation and strings as morphisms.)

$$\mathfrak{A} = (G, X^*, \alpha)$$

is called **non-deterministic finite automaton**.

For subsets $S, F \subset V$ we call

$$\mathfrak{A} = (G, X^*, S, F, \alpha)$$

finite automaton with start and final states *or* **finite acceptor**.

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

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

For the set of paths from start to final states

$$\mathfrak{W}(G)(S, F) := \{\pi \in \mathfrak{W}(G) \mid Q(\pi) \in S,\ Z(\pi) \in F\}$$

the set of path labels

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

is called the **set accepted by $\mathfrak{A}$**.

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

DEFINITION 1.2. *Let X be an alphabet. The set*

$$\mathbf{REG}(X^*) := \{L \subset X^* \mid \text{ there exists a finite acceptor } \mathfrak{A} \text{ with } L = L_{\mathfrak{A}}\}$$

is called the **set of regular languages** *over the free monoid X^*.*

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

$$\begin{aligned} \delta \quad = \quad & \{(a, v_1, v_2) \in X \times V \times V) \mid \\ & \text{there exists an edge } e \in E \text{ with } Q(e) = v_1,\ Z(e) = v_2 \text{ and } \alpha(e) = a \in X\} \end{aligned}$$

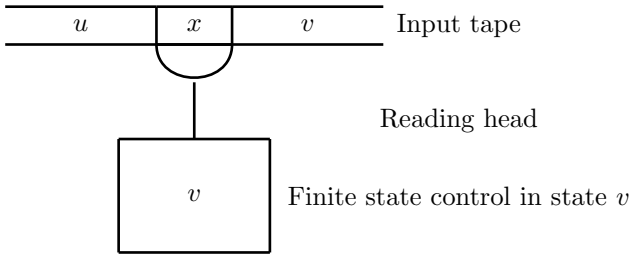
δ my be seen as a relation between $X \times V$ and V. X is the input alphabet and V the set of *states* of the automaton

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

The elements of V contain the possible states of the automaton $\mathfrak{B}$.

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

This interpretation can be visualized as follows:



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

The automaton $\mathfrak{A} = (G, X, S, F, \alpha)$ may be interpreted as a non-deterministic algorithm. The points (vertices) of the graph G define the possible states of the algorithm, the elements of X constitute the input alphabet.

The non-deterministic automaton $\mathfrak{A}$ which reads a symbol $x \in X$ while in state $v \in V$ changes into state v' if there exists an edge e from v to v' with label $\alpha(e) = x$. If the graph has no such edge the automaton goes "out of service".

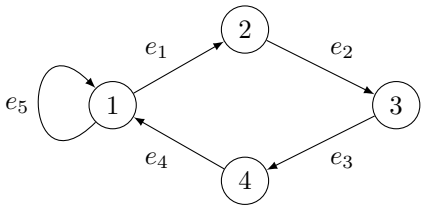
A finite acceptor accepts a word $w \in X^*$ if there exists a path from a vertex in S to a vertex in F which is labelled with w.

We want to look at some examples of finite automata:

EXAMPLE 1.1. *Let* $X = \{a, b\}$ *and* $L = \{(ab)^{2n} \mid n \in \mathbb{N}\}$.

It holds $L \in \mathbf{REG}(\{a, b\}^*)$.

The following acceptor $\mathfrak{A} = (G, X, \{1\}, \{1\}, \alpha)$ *accepts* L *(exercise):*

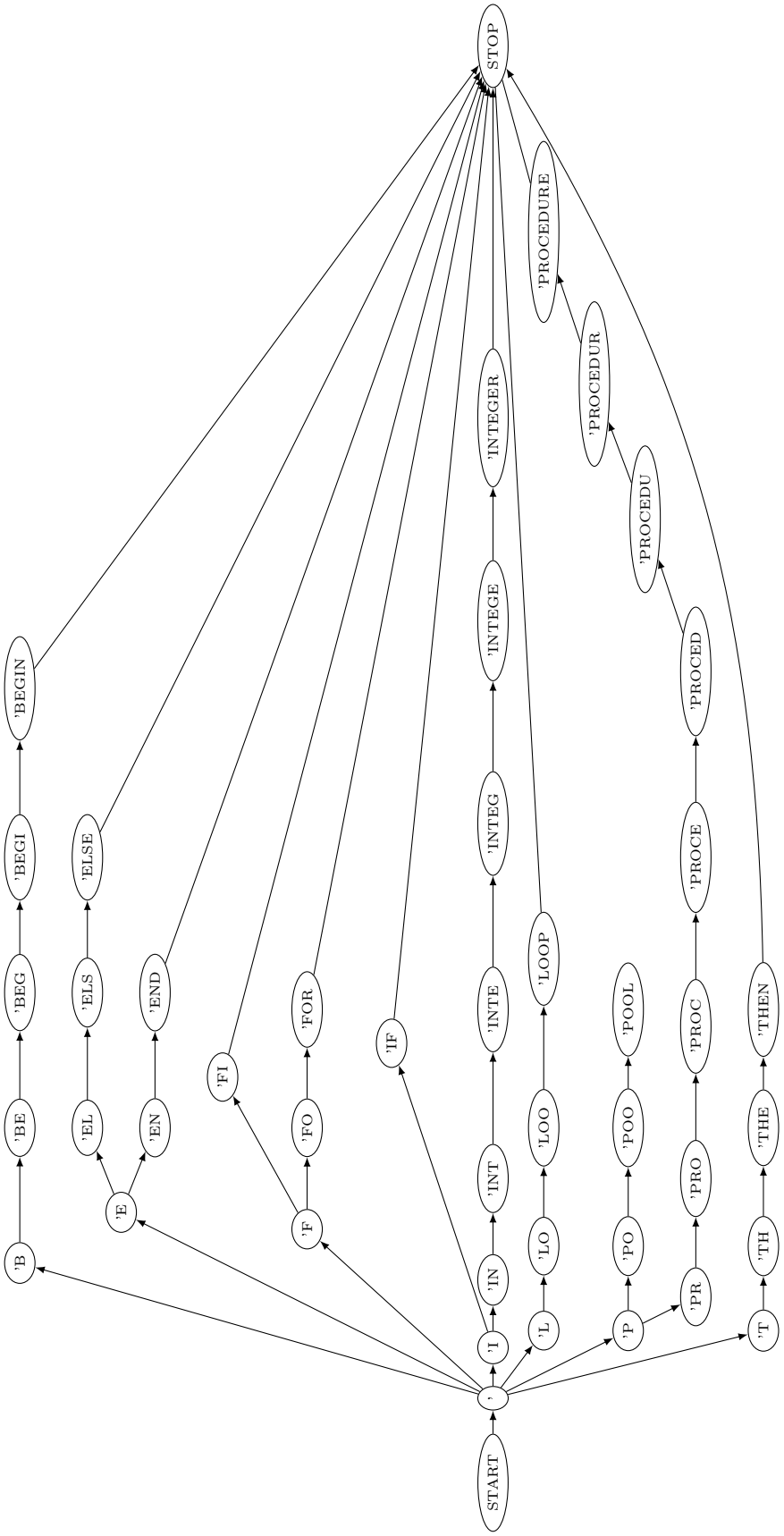


EXAMPLE 1.2. *Lexical analysis, check for special characters.*

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:


START
'
'B
'BE
'BEG
'BEGI
'BEGIN
'E
'EL
'ELS
'ELSE
'EN
'END
'F
'FI
'FO
'FOR
'I
'IF
'IN
'INT
'INTE
'INTEG
'INTEGE
'INTEGER
'L
'LO
'LOO
'LOOP
'P
'PO
'POO
'POOL
'PR
'PRO
'PROC
'PROCE
'PROCED
'PROCEDU
'PROCEDUR
'PROCEDURE
'T
'TH
'THE
'THEN
STOP

The mappings $\alpha(e)$ are shown as edge labels. The vertices of the graph are shown as ovals with labels. Start and final states are given by $S = \{\text{START}\}$ and $F = \{\text{STOP}\}$.

The labels for the vertices are chosen such that one can see the information "stored" by the automaton.

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

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

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

PROOF. (The proof from the original book has been slightly reformulated for better readability.)

If $\text{card}(S) = \text{card}(F) = 1$ we are done.

Let $\text{card}(S) > 1$ or $\text{card}(F) > 1$.

(1) Add a new vertex s' which will become the new start vertex and a new vertex f' which will become the new final vertex.

(2) Let
$$\text{START} := \{e \in E \mid Q(e) \in S\}$$
Add new start edges
$$\text{NEWSTART} := \{e' \mid Q(e') = s',\ Z(e') = Z(e),\ \alpha'(e') = \alpha(e),\ e \in \text{START}\}$$

(3) Let
$$\text{FINAL} := \{e \in E \mid Z(e) \in F\}$$
Add new final edges
$$\text{NEWFINAL} := \{e' \mid Q(e') = Q(e),\ Z(e') = f',\ \alpha'(e') = \alpha(e),\ e \in \text{FINAL}\}$$

The new automaton
$$\mathfrak{A}' = (G', X, \{S'\}, \{F'\}, \alpha')$$
is defined by the graph
$$G' = (V \cup \{S', F'\}, E \cup \text{NEWSTART} \cup \text{NEWFINAL})$$
and the labelling α' which is identical to α for all existing edges and defined as above for the new edges.

It can easily be shown that $L_{\mathfrak{A}'} = L_{\mathfrak{A}}$. □

LEMMA 1.2. *Let $\mathfrak{A} = (G, X, S, F, \alpha)$ be a finite automaton with graph $G = (V, E)$. Then there exists an automaton $\mathfrak{A}' = (G', X, S', F', \alpha')$ with $\alpha'(e) \in X$ for each edge $e \in E$ that accepts the same language.*

PROOF. We "split" the edges according to their labels.

(1) Split edges labelled with a word into edges labelled with single symbols:

Let $e \in E$ with $\alpha_2(e) = x_1 \cdots x_k,\ k > 1,\ x_i \in X$.
Remove the edge e and add new edges $e'_1, \ldots, e'_k$ and new vertices $v'_1, \ldots, v'_{k-1}$ such that $(Q(e), e'_1, \ldots, e'_k, Z(e)) \in \mathfrak{W}(G')$ and define a new graph $G' = (V', E')$. The labelling of the new edges will be 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) Handle edges labelled with the empty word.

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

(a) Remove ϵ-loops:
If for an edge e it holds $Q(e) = Z(e)$, remove e.

(b) Remove ϵ-edges which are dead-ends:
If there is no $e' \in E$ with $Q(e') = Z(e)$, remove e.

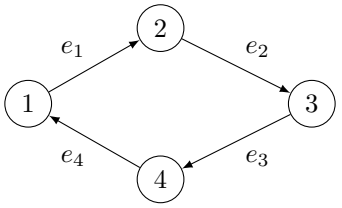
(c) Bridge remaining ϵ-edges and remove them:
If there exists an edge $e' \in E$ with $Q(e') = Z(e)$, remove e and add new edges $\tilde{e}$ with $Q(\tilde{e}) = Q(e),\ Z(\tilde{e}) = Z(e')$ and label $\alpha'(\tilde{e}) = \alpha(e')$.

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 vertex and edge sets are finite. For the new automaton $\mathfrak{A}'$ which results from this algorithm it holds: $L_{\mathfrak{A}'} = L_{\mathfrak{A}}$.

□

If we apply this algorithm to our automaton from example 1, we obtain the following graph where the edge e_5 has disappeared:



We want to prove now some closure properties of $\mathbf{REG}(X^*)$.

THEOREM 1.1 (Closure under union and intersection).

$$L, L' \in \mathbf{REG}(X^*) \Rightarrow L \cup L' \in \mathbf{REG}(X^*) \text{ and } L \cap L' \in \mathbf{REG}(X^*)$$

PROOF. Let $L = L_{\mathfrak{A}}$ and $L' = L_{\mathfrak{B}}$ with automata

$$\mathfrak{A} = (G_{\mathfrak{A}}, X, S_{\mathfrak{A}}, F_{\mathfrak{A}}, \alpha)$$

and

$$\mathfrak{B} = (G_{\mathfrak{B}}, X, S_{\mathfrak{B}}, F_{\mathfrak{B}}, \beta)$$

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

(1) **Closure under union:** Define

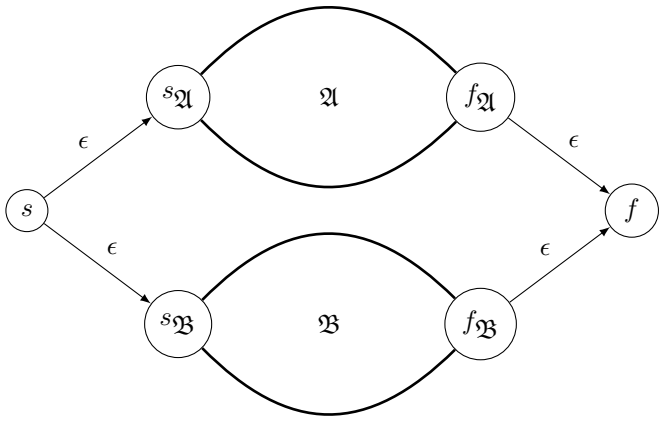
$$\gamma_2(e) := \begin{cases} \alpha_2(e), & e \in E_{\mathfrak{A}} \\ \beta_2(e), & e \in E_{\mathfrak{B}} \end{cases}$$

Then the automaton

$$\mathfrak{C}(G_{\mathfrak{A}} \cup G_{\mathfrak{B}}, X, S_{\mathfrak{A}} \cup S_{\mathfrak{B}}, F_{\mathfrak{A}} \cup F_{\mathfrak{B}}, \gamma)$$

accepts the language $L_{\mathfrak{A}} \cup L_{\mathfrak{B}}$.

If we make $\mathfrak{A}$ and $\mathfrak{B}$ initial, then the following initial automaton also accepts $L_{\mathfrak{A}} \cup L_{\mathfrak{B}}$:



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

$$V' = V_{\mathfrak{A}} \times V_{\mathfrak{B}}$$

$$E' = \{(e_A, e_B) \in E_{\mathfrak{A}} \times E_{\mathfrak{B}} \mid \alpha_2(e_A) = \beta_2(e_B)\}$$

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

We define the new labelling $\delta_2 : E' \to X$ by

$$\delta_2((e_A, e_B)) = \alpha_2(e_A)).$$

For the automaton

$$\mathfrak{A}' = \mathfrak{A} \times \mathfrak{B} := (G', X, S_{\mathfrak{A}} \times S_{\mathfrak{B}}, F_{\mathfrak{A}} \times F_{\mathfrak{B}}, \delta)$$

then holds $L_{\mathfrak{A}'} = L_{\mathfrak{A}} \cap L_{\mathfrak{B}}$. This automaton is called the **cartesian product** of $\mathfrak{A}$ and $\mathfrak{B}$.

□

THEOREM 1.2 (Closure under mirror operation).

$$L \in \mathbf{REG}(X^*) \Rightarrow L^R \in \mathbf{REG}(X^*)$$

PROOF. Let $\mathfrak{A} = (G, X, S_{\mathfrak{A}}, F_{\mathfrak{A}}, \alpha)$ be a finite acceptor for L with graph $G = (V, E)$.

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

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

which means that we reverse the orientation of the edges of G to get the edges of G'.

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

DEFINITION 1.3. *Let $\mathfrak{A} = (G, X, S, F, \alpha)$ be a finite automaton with graph $G = (V, E)$ and labelling $\alpha(E) \subset X$ (each edge is labeled with a single symbol).*

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

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

THEOREM 1.3 (Existence of complete, deterministic acceptor). *If $\mathfrak{A}$ is a finite automaton, then there exists a complete, deterministic automaton $\mathfrak{A}'$ which accepts the same language.*

PROOF. Using our lemmata we may assume that $\mathrm{card}(S_{\mathfrak{A}}) = 1$ and $\alpha_2(e) \in X$ for all edges e in the graph of $\mathfrak{A}$.

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

Let $G = (V, E)$ be the graph of $\mathfrak{A}$.

Instead of the vertices of G our new graph has the power set $\mathfrak{P}(V)$ of V as its vertex set.

For $\tilde{v} \in \mathfrak{P}(V)$ we define

$$\begin{aligned} N(x, \tilde{v}) \quad := \quad & \{v \in V \mid \text{there exists an edge } e \in E \text{ with } Q(e) \in \tilde{v},\ Z(e) = v \\ & \text{and } \alpha(e) = x\} \end{aligned}$$

The empty set $\emptyset \in \mathfrak{P}(V)$ will also become a vertex of the new graph.

Vertices $\tilde{q}, \tilde{r} \in \mathfrak{P}(V)$ will be connected by an edge e with label $\alpha_2'(e) = x \Leftrightarrow \tilde{r} = N(x, \tilde{q})$.

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

The start and final states are defined as follows:

$$\begin{aligned} S_{\mathfrak{A}'} &= \{S_{\mathfrak{A}}\} \\ F_{\mathfrak{A}'} &= \{\tilde{r} \in \mathfrak{P}(V) \mid \tilde{r} \cap F_{\mathfrak{A}} \neq \emptyset\} \end{aligned}$$

This completes the definition of the automaton

$$\mathfrak{A}' = (G', X, S_{\mathfrak{A}'}, F_{\mathfrak{A}'}, \alpha')$$

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

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

$$\operatorname{card}(\{e \in E' \mid Q(e) = \tilde{v}, \alpha_2'(e) = x\}) = 1$$

for all $\tilde{v} \in V'$.

Now we prove that the accepted languages are equal:

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

$$q \xrightarrow{x} r$$
$$\tilde{q} \xrightarrow{x} \tilde{r}$$

which are interpreted as follows: For $q \xrightarrow{x} r \in E$ there exists by construction $\tilde{q} \xrightarrow{x} \tilde{r} \in E'$ with $q \in \tilde{q}$ and $r \in \tilde{r}$.

Starting with $S_{\mathfrak{A}'} = \{S_{\mathfrak{A}}\} = \{\{q_0\}\}$ and concatenating these diagrams, we get for $x_1 \cdots x_k \in L_{\mathfrak{A}}$:

$$q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \longrightarrow \ldots \longrightarrow q_{k-1} \xrightarrow{x_k} q_k \in F_{\mathfrak{A}}$$
$$\tilde{q}_0 \xrightarrow{x_1} \tilde{q}_1 \xrightarrow{x_2} \tilde{q}_2 \longrightarrow \ldots \longrightarrow \tilde{q}_{k-1} \xrightarrow{x_k} \tilde{q}_k$$

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

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

"$\supset$": Here we use diagrams

$$\tilde{q} \xrightarrow{x} \tilde{r}$$
$$q \xrightarrow{x} r$$

which are interpreted as follows: For $\tilde{q} \xrightarrow{x} \tilde{r} \in E'$ and $r \in \tilde{r}$ there exists $q \in \tilde{q}$ with $q \xrightarrow{x} r \in E$.

For $x_1 \cdots x_k \in L_{\mathfrak{A}'}$ we start with $F_{\mathfrak{A}'}$ and continue the diagram from right to left. For $q_k \in \tilde{q}_k \cap F_{\mathfrak{A}}$ we get

$$S_{\mathfrak{A}'} \ni \quad \tilde{q}_0 \xrightarrow{x_1} \tilde{q}_1 \xrightarrow{x_2} \tilde{q}_2 \longrightarrow \ldots \longrightarrow \tilde{q}_{k-1} \xrightarrow{x_k} \tilde{q}_k \in F_{\mathfrak{A}'}$$
$$q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \longrightarrow \ldots \longrightarrow q_{k-1} \xrightarrow{x_k} q_k \in F_{\mathfrak{A}}$$

Here $q_i \in \tilde{q}_i$ for $i = 0, \ldots, k$.

Because of $S_{\mathfrak{A}'} = \{S_{\mathfrak{A}}\} = \{\{q_0\}\}$ it holds $x_1 \cdots x_k \in L_{\mathfrak{A}}$

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

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

Remark: The state set grows exponentially when the automaton is made deterministic. The following example will show this fact [**Com75**].

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

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

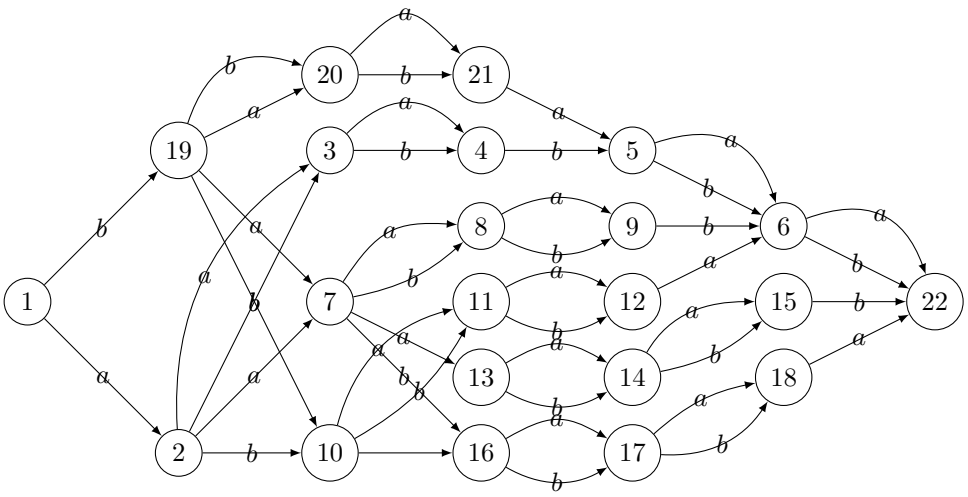
$L_n \in \mathbf{REG}(X^*)$ is regular because there exists a non-deterministic finite acceptor $\mathfrak{A}_n$ with $L_n = L_{\mathfrak{A}_n}$.

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

Let

$$\mathfrak{A}_3 = (G_3, \{a, b\}, \{1\}, \{22\}, \alpha)$$

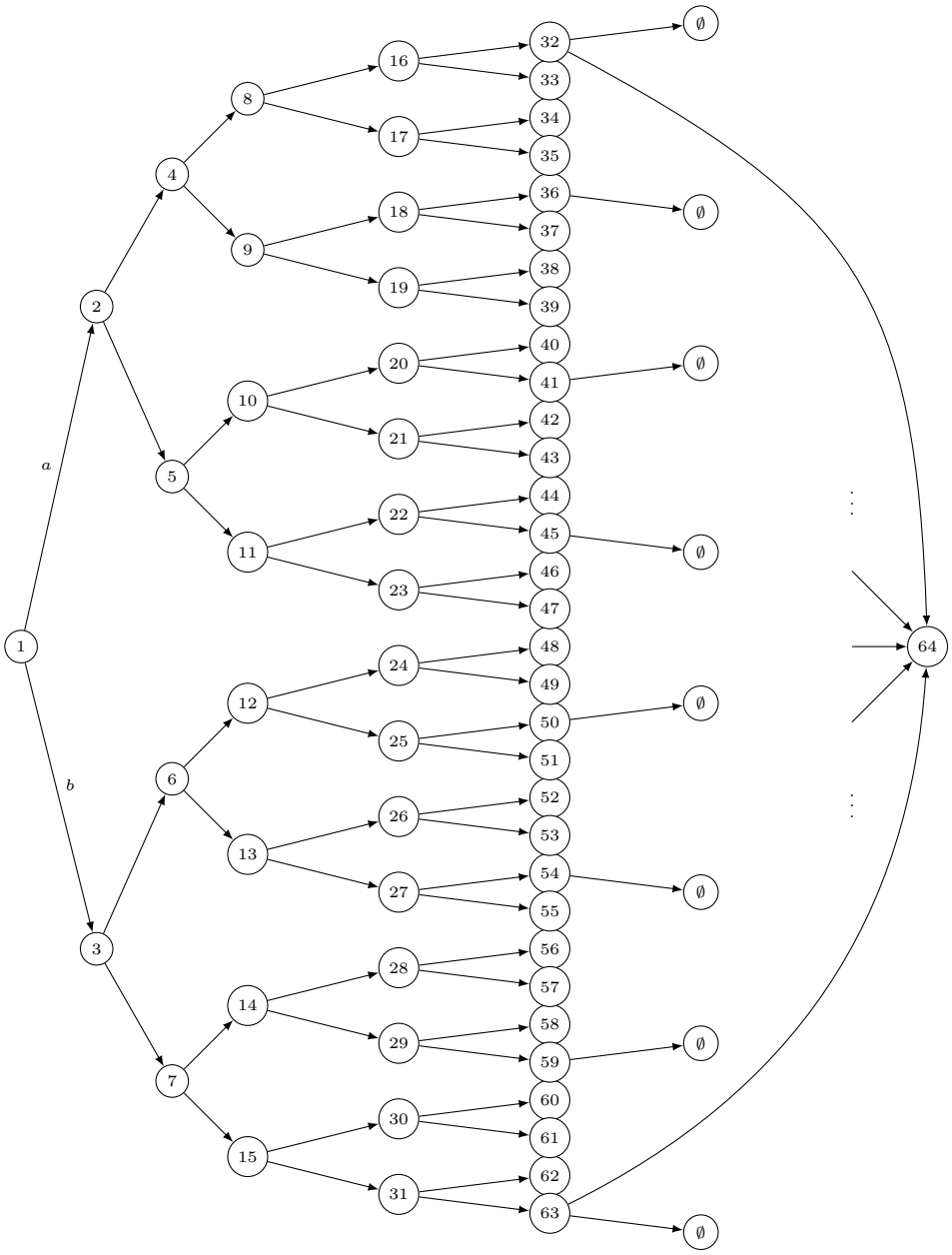
defined by the following graph G_3:



The labelling α is shown at the edges.

EXERCISE 1.1. *Show that this automaton accepts L_3.*

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



The edges pointing upwards are labelled with a and the edges pointing downwards with b. From each of the vertices $32 \ldots 63$ two edges lead to the final state vertex 64 except the edges shown leading to the $\emptyset$-vertices.

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

In [**Com75**] it is proved that this automaton is minimal in the number of states.

Making a state graph deterministic may have 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 non-deterministic graph, it can follow all alternatives in parallel, similarly 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 (Closure under complement).

$$L \in \mathbf{REG}(X^*) \Rightarrow \bar{L} = X^* \setminus L \in \mathbf{REG}(X^*)$$

PROOF. $L \in \mathbf{REG}(X^*)$, then there exists a complete, deterministic finite acceptor $\mathfrak{A}$ with $L_{\mathfrak{A}} = L$.

Define

$$\mathfrak{A}' = (G, X, S_{\mathfrak{A}}, F_{\mathfrak{A}}, \alpha)$$

where $F_{\mathfrak{A}} := V \setminus F_{\mathfrak{A}}$.

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

This $w \in X^*$ uniquely determines a vertex $q \in V$ with $w \in \alpha(\mathfrak{W}(W)(S_{\mathfrak{A}}, q))$.

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

THEOREM 1.5 (Closure under complex product).

$$L_1, L_2 \in \mathbf{REG}(X^*) \Rightarrow L_1 \cdot L_2 \in \mathbf{REG}(X^*)$$

PROOF. For $L_1, L_2 \in \mathbf{REG}(X^*)$ there exist finite acceptors

$$\mathfrak{A}_i = (G_i, X, S_i, F_i, \alpha_i), \quad G_i = (V_i, E_i),\ i = 1, 2$$

such that $L_i = L_{\mathfrak{A}_i}$.

We define an acceptor

$$\mathfrak{A}_1 \circ \mathfrak{A}_2 := (G, X, S, F, \alpha), \quad G = (V, E)$$

with vertex set

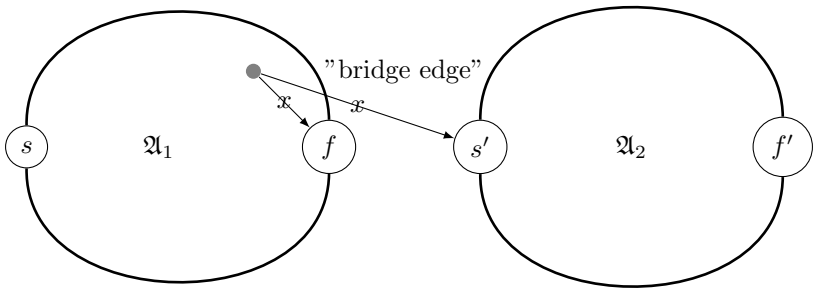
$$V := V_1 \cup V_2 \text{ where without loss of generality } V_1 \cap V_2 = \emptyset$$

and edge set

$$E := E_1 \cup E_2 \cup \underbrace{\{Q(e) \xrightarrow{x} s' \mid \exists e \in E_1,\ e : Q(e) \xrightarrow{x} f,\ f \in F_1,\ s' \in S_2\}}_{\text{bridge edges}}$$

This means for each edge e ending in a final state f of the first automaton, we create a "bridge" from the source of this edge to the start state of the second automaton.

Start and final states of the new automaton are given by $S := S_1$ and $F := F_2$.



We prove that $L_{\mathfrak{A}} = L_{\mathfrak{A}_1} \cdot L_{\mathfrak{A}_2}$.

(1) $L_{\mathfrak{A}} \supset L_{\mathfrak{A}_1} \cdot L_{\mathfrak{A}_2}$

Let $u = u_1 \cdots u_n \in L_{\mathfrak{A}_1}$ and $v = v_1 \cdots v_m \in L_{\mathfrak{A}_2}$. Then there exist accepting paths

$$S_1 \ni p_0 \xrightarrow{u_1} p_1 \xrightarrow{u_2} \ldots \xrightarrow{u_n} p_n \in F_1$$

and

$$S_2 \ni q_0 \xrightarrow{v_1} q_1 \xrightarrow{v_2} \ldots \xrightarrow{v_m} q_m \in F_2$$

By construction of G there exists a path in $\mathfrak{W}(G)(S_1, F_2)$ of the form

$$S_1 \ni p_0 \xrightarrow{u_1} p_1 \xrightarrow{u_2} \ldots \underbrace{p_{n-1} \xrightarrow{u_n} q_0}_{\text{bridge}} \xrightarrow{v_1} q_1 \xrightarrow{v_2} \ldots \xrightarrow{v_m} q_m \in F_2$$

It follows $u = u_1 \cdot u_2 \cdots u_n \cdot v_1 \cdots v_m \in L_{\mathfrak{A}}$ therefore $L_{\mathfrak{A}_1} \cdot L_{\mathfrak{A}_2} \subset L_{\mathfrak{A}}$.

(2) $L_{\mathfrak{A}} \subset L_{\mathfrak{A}_1} \cdot L_{\mathfrak{A}_2}$

Let

$$S \ni p_0 \xrightarrow{w_1} p_1 \xrightarrow{w_2} \ldots \xrightarrow{w_n} p_n \in F$$

be an accepting path in $\mathfrak{W}(G)(S, F)$. Then there exist vertices $p_i \in S_2$ and $p_j \in V_1$ with $j < i$ because $V_1 \cap V_2 = \emptyset$.

Therefore the subpath

$$S_2 \ni p_i \xrightarrow{w_{i+1}} p_{i+1} \longrightarrow \ldots \xrightarrow{w_n}{}_n \in F_2$$

contains only edges from $\mathfrak{A}_2$ and is an accepting path for $w_{i+1} \cdots w_n \in L_{\mathfrak{A}_2}$.

It is $p_{i-1} \in V_1$ and by construction of the graph G there exists an edge labelled w_i which ends in a final state of $\mathfrak{A}_1$. Therefore $w_1 \cdots w_i \in L_{\mathfrak{A}_1}$ and it holds

$$w_1 \cdots w_i \cdots w_n \in L_{\mathfrak{A}_1} \cdot L_{\mathfrak{A}_2} \Rightarrow L_{\mathfrak{A}} \subset L_{\mathfrak{A}_1} \cdot L_{\mathfrak{A}_2}$$

From (1) and (2) it follows $L_{\mathfrak{A}} = L_{\mathfrak{A}_1} \cdot L_{\mathfrak{A}_2}$. □

THEOREM 1.6 (Closure under Kleene-star).

$$L \in \mathbf{REG}(X^*) \Rightarrow L^* \in \mathbf{REG}(X^*)$$

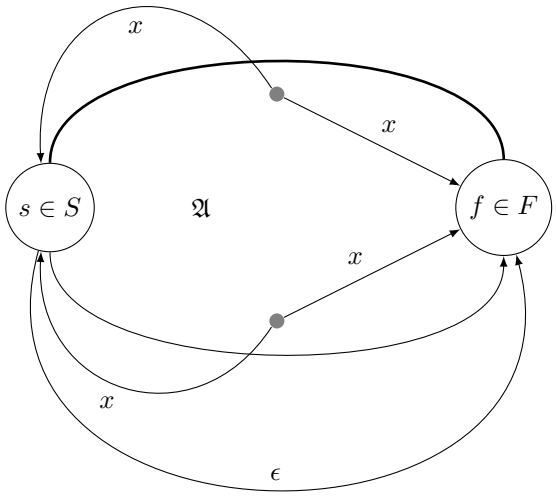
PROOF. Let $\mathfrak{A} = (G, X, S, F, \alpha),\ G = (V, E)$ be a finite acceptor for L. Create the finite acceptor

$$\mathfrak{A}^* = (G^*, X, S, F, \alpha^*),\ G' = (V', E')$$

from G by adding the following edges:

(1) For each edge $e : Q(e) \xrightarrow{x} f \in E,\ f \in F$ ending in a final state and for each start state s, add a backward edge $e' : Q(e) \xrightarrow{x} s$ with label $\alpha'(e') = x$.

(2) For each start state $s \in S$ and each final state $f \in F$, add a forward edge (if not already existing) $e' : s \xrightarrow{\epsilon} f$ with label $\alpha'(e') = \epsilon$.



For the finite acceptor $\mathfrak{A}^*$ then holds (exercise):

$$L_{\mathfrak{A}^*} = L^*$$

□

We have seen (lemma 1) that for non-deterministic automata a single start and a single final state are sufficient, but in the deterministic case a single final state is in general insufficient.

THEOREM 1.7 (Closure under homomorphism).

$$L \in \mathbf{REG}(X^*),\ \phi : X^* \to Y^* \textit{ monoid homomorphism } \Rightarrow$$

$$\phi(L) := \{\phi(w) \mid w \in L\} \in \mathbf{REG}(X^*)$$

PROOF. Let $\mathfrak{A} = (G, X, S, F, \alpha)$ be a finite acceptor for L with graph $G = (V, E)$. Then

$$\mathfrak{B} = (G, Y, S, F, \beta)$$

with $\beta_2(e) := \phi(\alpha_2(e)),\ e \in E(G)$ is an acceptor for $\phi(L)$ because

$$\begin{aligned}\phi(L) &= \phi(\alpha_2(\mathfrak{W}(G)(S,F))) \\ &= \beta_2(\mathfrak{W}(G)(S,F))\end{aligned}$$

Therefore

$$\phi(L) \in \mathbf{REG}(Y^*)$$

□

We prove some more important properties of regular sets.

LEMMA 1.3. *For $x \in X$ it holds*

$$\{x\} \in \mathbf{REG}(X^*) \text{ and } \emptyset \in \mathbf{REG}(X^*)$$

PROOF. For $G = (\{s,f\},\{e\})$ with $Q(e) = s,\ Z(e) = f$, the finite acceptor

$$\mathfrak{A}_x = (G,\{x\},\{s\},\{f\},\alpha)$$

with $\alpha_2(e) = x$ accepts $L_{\mathfrak{A}} = \{x\}$.

For $G = (\{s,f\},\emptyset)$ the finite acceptor

$$\mathfrak{A} = (G,\emptyset,\{s\},\{f\},\emptyset)$$

accepts $L_{\mathfrak{A}} = \emptyset$. □

LEMMA 1.4 (The set of accepting paths is regular). *Let $G = (V,E)$. Then*

$$\mathfrak{W}(G)(S,F) \in \mathbf{REG}(E^*)$$

This lemma states that the paths in a graph leading from a start state to a final state form a regular set over the free monoid of edge sequences.

PROOF. The proof is trivial, just label each edge with itself and you get a finite acceptor for the path set. □

DEFINITION 1.4. *$L \subset X^*$ is called* **local over** *$X \iff$ there exist subsets $S,\ F \subset X$ and a relation $R \subset X \times X$ with*

$$L = \{x_1 \cdots x_k \mid x_1 \in S,\ x_k \in F,\ (x_i, x_{i+1}) \in R, \quad i = 1,\ldots,k-1\}$$

The relation R specifies the "allowed" sequences.

LEMMA 1.5 (The set of accepting paths is a local set). *Let $G = (V,E)$ be a graph. Then $\mathfrak{W}(G)(S,F)$ is local over E.*

PROOF. Let

$$\begin{aligned}S &= \{e \in E \mid Q(e) \in S\} \\ F &= \{e \in E \mid Z(e) \in F\} \\ R &= \{(e,e') \in E \times E \mid Z(e) = Q(e')\}\end{aligned}$$

Then the claim is immediately clear. □

LEMMA 1.6 (Every local set is a regular set).

$$L \text{ local over } X \Rightarrow L \in \mathbf{REG}(X^*)$$

PROOF. Let S and F be the subsets of X from the definition of a local set.

Define a graph $G = (V, E)$ as follows:

$$\begin{aligned} V &:= X \cup \{\bar{S}\} \text{ with } X \cap \{\bar{S}\} = \emptyset \\ E &:= \{e : \{\bar{S}\} \to x \mid x \in S\} \cup \{e : x \to y \mid (x, y) \in R\} \end{aligned}$$

Consider the finite acceptor $\mathfrak{A} = (G, X, \{\bar{X}\}, F, \alpha)$ with labelling α defined by $\alpha_2(e) = Z(e)$ for each edge $e \in E$.

Then it is clear that $L_{\mathfrak{A}} = L$. □

LEMMA 1.7 (Every regular set is the homomorphic image of a local set). *Let $L \in \mathbf{REG}(X^*)$ be a regular set. Then there exists a local set R over X and a monoid homomorphism ϕ such that*

$$\phi(R) = L$$

PROOF. Exercise. □

We want to summarize our results to a number of main theorems, here we organize these results differently.

Let $X_\infty = \{x_1, x_2, \ldots\}$ be an infinite alphabet. Define

$$\mathbf{REG}(X_\infty^*) = \bigcup_{\substack{X \subset X_\infty \\ X finite}} \mathbf{REG}(X^*)$$

MAIN THEOREM 1.1.

(1) $\mathbf{REG}(X_\infty^*)$ *is closed under union, complex product and star-operator.*

(2) *If $\phi : X_\infty^* \to X_\infty^*$ is a monoid homomorphism, then $\mathbf{REG}(X_\infty^*)$ is closed under ϕ.*

(3) *Every $L \in \mathbf{REG}(X_\infty^*)$ is the homomorphic image of a local set over X_∞^*.*

(4) $\mathbf{REG}(X_\infty^*)$ *contains the sets $\{x\}$, $x \in X_\infty^*$ and the empty set $\emptyset$.*

MAIN THEOREM 1.2.

(1) $\mathbf{REG}(X_\infty^*)$ *is a Boolean Algebra with the operations union, intersection and complement.*

(2) *Every language $L \in \mathbf{REG}(X_\infty^*)$ is accepted by some complete, deterministic automaton.*

To prove the first main theorem we didn't need the fact that X^* is a **free** monoid. In the proof of main theorem 2 we needed that fact.

There exist monoids M for which the second main theorem does not hold for $\mathbf{REG}(M)$, in both statements (see exercises).

Of special interest are the monoids $M = F(X)$, where $F(X)$ is the free group generated by X (see chapter 1.3), and $M = X^{\oplus}$, where $X^{\oplus}$ is the free commutative group generated by X.

Languages $L \in \mathbf{REG}(X^{\oplus})$ are also called **semi-linear**.

Exercise 1.2.

(1) *Let* $\phi : (\{x, y\}^*, \cdot) \to (\mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}, +)$ *be given by* $\phi(x) = (0, 1),\ \phi(y) = (1, 0)$.

Construct a finite automaton $\mathfrak{A}$ *with* $L_{\mathfrak{A}} = \phi^{-1}((0, 0))$ *and present* $\phi^{-1}((0, 0))$ *as a rational set.*

(2) *Let* $\mathfrak{A} = (G, X, S, F, \alpha)$ *be a finite automaton with* n *states. Prove:*

   (a) *If* $x \in L_{\mathfrak{A}}$ *with* $|x| >= n$, *then there exists words* $u, v, w \in X^*$ *with* $x = u \cdot v \cdot w, v \neq \epsilon$ *and* $u \cdot v^m \cdot w \in L_{\mathfrak{A}}\ \forall m \geq 1$ *("pumping lemma").*

   (b) $L_{\mathfrak{A}}$ *is infinite* $\iff \exists\, x \in X^*$ *with* $n \leq |x| \leq 2n$ *and* $x \in L_{\mathfrak{A}}$.

(3) *Show: The set* $L = \{a^n b^n \mid n \in \mathbb{N}\}$ *is not regular.*

(4) *Let* $L \in \mathbf{REG}(X^*),\ M \subset X^*$ *finite. Show:* $M^{-1}L \in \mathbf{REG}(X^*)$. *What if* $M \subset \mathbf{REG}(X^*)$ *is an arbitrary regular set?*

2. Rational sets in X^*, $\mathbf{RAT}(X^*)$

In this section we want to introduce a second characterization of the regular sets over free monoids. We need the following definition.

DEFINITION 2.1. *If M is a monoid, then $K \subset \mathfrak{P}(M)$ is called is called* **rationally closed** $\iff$ *K is closed unter union $\cup$, complex product $\cdot$ and Kleene-star *.*

DEFINITION 2.2. $\mathbf{RAT}(M)$ *is the smallest rationally closed subset of $\mathfrak{P}(M)$ which contains the single element sets $\{m\},\ m \in M$, and the empty set $\emptyset$.*

From main theorem 1 it immediately follows:

LEMMA 2.1. $\mathbf{RAT}(X^*) \subset \mathbf{REG}(X^*)$.

From the remark following the main theorems it follow that this also holds for arbitrary monoids M because $\{m\} \in \mathbf{REG}(M)$ for $m \in M$.

This lemma is the first part of the following theorem by Kleene:

THEOREM 2.1 (Kleene).

$$\mathbf{REG}(X^*) = \mathbf{RAT}(X^*)$$

PROOF. We only have to proof the inclusion $\mathbf{REG}(X^*) \subset \mathbf{RAT}(X^*)$.

Let $\mathfrak{A} = (G, X, S, F, \alpha),\ G = (V, E)$ be a finite automaton. We show that $L_{\mathfrak{A}}$ can be generated from $\emptyset$ and $\{x\},\ x \in X$ using the $\cup$, $\cdot$ and *-operations.

We prove this by induction over the number k of edges in the graph of the automaton when the number of vertices is fixed.

By lemma 1 from chapter II.1, we may assume that the automaton has a single start state, $\mathrm{card}(S) = 1$.

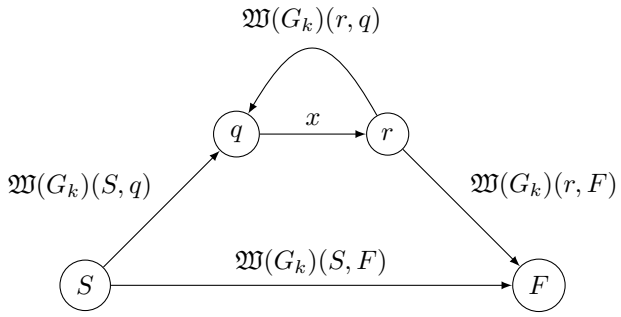
Induction base:
$k = 0$: Without loss of generality, we may assume $S \cap F = \emptyset$. It follows $L_{\mathfrak{A}} = \emptyset \in \mathbf{RAT}X^*$.

Induction step:
Let the claim be true for $i \leq k,\ k > 1$, and let G_k be a graph with k edges.

Add a new edge $e : q \xrightarrow{x} r$ labeled with $x \in X$ to the graph G_k. The resulting graph shall be G_{k+1}.

The set of accepting paths in G_{k+1} can be written as

$$\begin{aligned} \mathfrak{W}(G_{k+1})(S,F) &= \mathfrak{W}(G_k)(S,F) \\ &\cup \mathfrak{W}(G_k)(S,q) \cdot e \cdot \Big(\mathfrak{W}(G_k)(r,q) \cdot e\Big)^* \cdot \mathfrak{W}(G_k)(r,F) \end{aligned}$$



By induction hypothesis it holds:

$$\alpha(\mathfrak{W}(G_k)(S,F)) \in \mathbf{RAT}(X^*)$$
$$\alpha(\mathfrak{W}(G_k)(S,q)) \in \mathbf{RAT}(X^*)$$
$$\alpha(\mathfrak{W}(G_k)(r,F)) \in \mathbf{RAT}(X^*)$$
$$\alpha(\mathfrak{W}(G_k)(r,q)) \in \mathbf{RAT}(X^*)$$

Together we get $\alpha\left(\mathfrak{W}(G_{k+1})(S,F)\right) \in \mathbf{RAT}(X^*)$. This concludes the proof of Kleene's theorem. □

3. Sets recognizable by homomorphisms, $\mathbf{REC}(X^*)$

In this section we consider a third possibility for characterizing subsets of X^*.

DEFINITION 3.1.

$$\begin{aligned}\mathbf{REC}(X^*) \quad := \quad & \{L \subset X^* \mid \textit{ there exists a finite monoid } H \\ & \textit{and a monoid homomorphism } \mu : X^* \to H \\ & \textit{with } L = \mu^{-1}(T),\ T \subset H\}\end{aligned}$$

is the class of **recognizable sets** *over* X.

LEMMA 3.1.

$$\mathbf{REG}(X^*) \subset \mathbf{REC}(^*)$$

PROOF. Let $L \in \mathbf{REG}(X^*)$, then there exists a complete, deterministic finite automaton $\mathfrak{A} = (G, X, S, F, \alpha)$ with graph $G = (V, E)$ accepting L.

We define

$$H := Map(V, V) := \{f : V \to V \mid f \text{ is a mapping}\}$$

We define a homomorphism $\mu : X^* \to H$ as follows:

Let μ be the homomorphic continuation of the mapping μ' defined by

$$\mu'(x)(q) = r \iff$$
$$\text{there exists an edge } e \in E \text{ with } Q(e) = q, Z(e) = r \text{ and } \alpha(e) = x \in X$$

We define

$$T := \{f \in H \mid f(S) \in F\}$$

Then it holds $L_{\mathfrak{A}} = \mu^{-1}(T)$ (exercise). □

LEMMA 3.2.

$$\mathbf{REC}(X^*) \subset \mathbf{REG}(X^*)$$

PROOF. Sei $L \in \mathbf{REC}(X^*)$, let H be a finite monoid and $\mu : X^* \to H$ a monoid homomorphism and $L = \mu^{-1}(T),\ T \subset H$.

We construct a graph $G = (V, E)$ with

- vertex set $V := H$
- edge set $E := \{(h, a) \mid h \in H,\ a \in X\}$ with $Q((h, a)) = h,\ Z((h, a)) = h \cdot \mu(a)$

Let

$$\mathfrak{A} = (G, X, S, F, \alpha)$$

with start states $S = \{1_H\}$, final states $F = T$ and labelling $\alpha((h, a)) = a$.

We show: $L = L_{\mathfrak{A}}$.

"$\subset$": Let $w \in L$, i.e. $w \in X^*$ with $\mu(w) \in T$.

If $w = x_1 \cdots w_n$ with $x_i \in X$, then $\mu(x_1) \cdots \mu(x_n) = t \in T$.

We consider the path

$$\pi = (1_H, (1_H, x_1), (\mu(x_1), x_2), (\mu(x_1 \cdot x_2), x_3), \ldots, (\mu(x_1 \cdots x_{n-1}), x_n), \mu(w))$$

$\pi \in \mathfrak{W}(G)$ with label $\alpha(\pi) = x_1 \cdots x_n = w$.

It even holds $\pi \in \mathfrak{W}(G)(S, F)$ because $\mu(w) \in T$. From this follows $w \in L_{\mathfrak{A}}$.

"$\supset$": Let $w \in L_{\mathfrak{A}}$, then there exists a path $\pi \in \mathfrak{W}(G)(\{1_H\}, T)$ with label $\alpha(\pi) = w$.

For $w = x_1 \cdots x_n$ then obviously holds $\mu(x_1 \cdots x_n) \in T$, thus $w \in \mu^{-1}(T) = L$ from which the claim follows. □

From both lemmata immediately follows

THEOREM 3.1 (Regular sets and recognizable sets coincide over free monoids).

$$\mathbf{REG}(X^*) = \mathbf{REC}(X^*)$$

In the first section of this chapter we showed closure properties for regular sets. We show now closure under inverse homomorphism which is rather easy using our last theorem.

THEOREM 3.2 (Closure under inverse homomorphism). *Let $\phi : X^* \to Y^*$ be a monoid homomorphism.*

$$L \in \mathbf{REC}(Y^*) \Rightarrow \phi^{-1}(L) \in \mathbf{REC}(X^*)$$

PROOF. Let $L \in \mathbf{REC}(X^*)$, i.e. L is defined by a monoid homomorphism $\mu' : Y^* \to H$ as $L = \mu^{-1'}(T)$ for some subset $T \subset H$.

We set $\mu := \phi \circ \mu'$. Then $\mu : X^* \to H$ is also a monoid homomorphism and $\phi^{-1}(L) = \mu^{-1}(T)$ from which follows $\phi^{-1}(L) \in \mathbf{REC}(X^*)$. □

EXERCISE 3.1. *For the integer numbers $\mathbb{Z}$, the set of recognizable sets $\mathbf{REC}(\mathbb{Z})$ shall be defined in analogy to $\mathbf{REC}(X^*)$.*

Characterize $\mathbf{REC}(\mathbb{Z})$! Are there non-empty, infinite sets in $\mathbf{REC}(\mathbb{Z})$?

4. Right-linear languages, r-LIN(X^*)

In this section, we investigate an additional method for defining special subsets of X^*. We use the mechanism introduced in chapter I.6, the Chomsky grammars.

We consider a very simple, restricted type of productions and obtain the **right-linear** grammars.

DEFINITION 4.1. *A Chomsky grammar $G = (N, X, P, S)$ is called*

right-linear *if* $P \subset N \times (X \cdot N \cup X)$

left-linear *if* $P \subset N \times (N \cdot X \cup X)$

X is the **terminal alphabet**.

As in chapter I.6 we have the notions of *derivation* and *generated language*.

DEFINITION 4.2.

r-LIN$(X^*) = \{L \subset X^* \mid$ *there exists a right-linear grammar G with $L = L(G)$*$\}$ *is the class of* **right-linear languages** *in X^*.*

In the same way one defines the class of **left-linear languages**.

The following theorem holds:

THEOREM 4.1.

$$\mathbf{REG}(X^*) = \mathbf{r\text{-}LIN}(X^*)$$

PROOF. "$\supset$": Let $L \in \mathbf{r\text{-}LIN}(X^*)$ with $L = L(G),\ G = (N, X, P, S)$.

We construct a graph $\Gamma = (V, E)$ with vertex set $V = N \cup \{F\},\ F \notin N$, and edge set $E = P$, i.e. the productions of the grammar become the edges of our graph:

$$\begin{aligned} p : v \to xv' \in P \quad &\Rightarrow \quad e : v \xrightarrow{x} v' \in E \\ p : v \to x \in P \quad &\Rightarrow \quad e : v \xrightarrow{x} F \in E \end{aligned}$$

This defines a finite acceptor $\mathfrak{A} = (\Gamma, X, \{S\}, \{F\}, \alpha)$.

It is easily seen that

$$\pi \in \mathfrak{W}(\Gamma) \Rightarrow Q(\pi) \overset{*}{\underset{G}{\Rightarrow}} \alpha(\pi) \cdot Z(\pi), \text{ if } Z(\pi) \in N$$

For paths $\pi \in \mathfrak{W}(\Gamma)(S, F)$ we get $\alpha(\pi) \in L$. From this follows $L_{\mathfrak{A}} \subset L(G)$.

Let $w \in L(G)$ be a word generated by the grammar G, then there exists a sequence of derivation steps

$$S \underset{G}{\Rightarrow} x_1 v_1 \underset{G}{\Rightarrow} x_1 x_2 v_2 \underset{G}{\Rightarrow} \ldots \underset{G}{\Rightarrow} x_1 \cdots x_{n-1} v_{n-1} \underset{G}{\Rightarrow} w$$

Then the path

$$\pi = \big(S, (S, x_1 v_1), (v_1, x_2 v_2), \ldots, (v_{n-1}, x_{n-1} v_{n-1}), (v_{n-1}, x_n), F\big) \in \mathfrak{W}(\Gamma)(S, F)$$

is labelled by

$$\alpha(\pi) = x_1 \cdots x_n = w$$

which means that the word w is accepted by the finite acceptor, so $w \in L_{\mathfrak{A}}$.

Together, we get $L_{\mathfrak{A}} = L(G)$.

"$\subset$":

Let $L \in \mathbf{REG}(X^*),\ L = L_{\mathfrak{A}}$ with a finite acceptor

$$\mathfrak{A} = (\Gamma, X, S_{\mathfrak{A}}, F_{\mathfrak{A}}, \alpha), \Gamma = (V, E)$$

and labels $|\alpha(e)| = 1$ for each edge $e \in E$.

We define a right-linear grammar $G = (N, X, P, S)$ with non-terminals $N = E$, start symbol $S = S_{\mathfrak{A}}$ and production set

$$\begin{aligned} P &= \{q \to \alpha(e) \cdot r \mid \exists \text{ edge } e : q \to r \in E\} \\ &\cup \{q \to \alpha(e) \mid \exists \text{ edge } e : q \to r \in E,\ r \in F_{\mathfrak{A}}\} \end{aligned}$$

It holds $L(G) = L_{\mathfrak{A}}$ (exercise) which completes the proof of the theorem. □

Remark: It also holds **l-LIN**$(X^*) =$ **REG**(X^*). This can be seen using theorem 2 in chapter II.1 which states that **REG**(X^*) is closed under the mirror-operation.

Given a right-linear grammar G for L, one can directly define a left-linear grammar for the mirror language L^R: Replace each production $X \to t \cdot Y$ by a left-linear production $X \to Y \cdot t$. Because of $(L^R)^R = L$ it then holds $L \in l{-}LIN(X^*)$.

We can therefore in all of the following results replace **r-LIN**(X^*) by **l-LIN**(X^*).

We extend now our main theorems from section 1. Let X_∞ again be a countably infinite alphabet and let **RAT**(X_∞^*) and **REC**(X_∞^*) be defined as before.

Main Theorem 4.1.

(1) **REG**$(X_\infty^*) =$ **RAT**$(X_\infty^*) =$ **REC**$(X_\infty^*) =$ **r-LIN**$(X_\infty^*) =$ **l-LIN**(X_∞^*)

(2) **REG**(X_∞^*) *is closed under union, intersection, complement, Kleene-star, complex product, monoid homomorphisms and inverse homomorphisms.*

Each of these language classes motivates a different generalization. We want to shortly present them:

Regular languages: Generalizations into two directions come to mind: Replace the free monoid X^* by an arbitrary monoid or by special monoids like free groups or free commutative monoids. We already pointed that out and we will investigate two important cases later in this book.

The second generalization concerns the free (path) category. By adding a second operation we get simply computable, infinite categories. The paths in graphs are then replaced by trees or nets representing the morphisms of these new free category.

Rational languages: Instead of the free monoid one can use arbitrary monoids. We already made some statements about this. We already know that this generalization coincides with the corresponding generalization of the regular sets.

In an even stronger generalization the monoid can be replaced by other algebras, that is one adds other operations in addition to the monoid operation. The question arises if in this case the strong relationship between the rational and the regular sets is preserved.

Recognizable languages: Here also two different directions for generalization come to mind. First, one can keep monoids but replace finite monoids by simply computable, infinite monoids.

A second generalization concerns the replacement of monoids by other algebras. Especially the free monoid can be replaced by free algebras and the finite monoid by finite algebras.

Right-linear languages: Here one generalizes by using general Chomsky grammars (see definition 1, chapter I.6). This last generalization will be in our focus later.

Our main theorem motivates yet another generalization:

Abstract Families of Languages (AFL): The AFL-theory generalizes the notion of the family of rational languages. This is done by generalizing the notion of **Kleene-algebra** which is based on the operations of union, complex product and Kleene-star.

A set $\mathcal{A} \in \mathfrak{P}(X_\infty^*)$ is called **full AFL**, if $\mathcal{A}$ is closed under the operations of union, complex product, Kleene-star, homomorphism, inverse homomorphism and intersection with regular languages.

The idea is the following:

All these operations are "simple", therefore using these operations only "simple sets" mya be constructed from some given set of "simple sets". It should be possible to decide the common computer science questions for all elements in an AFL if one can decide them for the generating basic sets.

In this book we cannot treat all these topics. Especially, we will not develop the AFL-theory.

The interested reader is referred to the book by Jean Berstel [**Ber79**] which offers a large space to this theory.

5. The rational transducer

In this section, we extend the finite automaton by an output which leads to:

DEFINITION 5.1.

$$\mathfrak{T} = (G, X, S, F, \alpha, \beta)$$

is called a **finite transducer** *with input alphabet* X *and output alphabet* $Y \iff$

$$(G, X, S, F, \alpha)$$

is a finite automaton with graph $G = (V, E)$ *and*

$$\beta = (\mathfrak{W}(G), Y^*, \beta_1, \beta_2)$$

is a functor.

$\mathfrak{T}$ is called **length-preserving** if $|\alpha(e)| = |\beta(e)|$ for all edges $e \in E$.

$\mathfrak{T}$ computes the **finite transduction**

$$\begin{aligned} \tau(\mathfrak{T}) &= \{(u, v) \in X^* \times Y^* \mid \text{there exists a path } \pi \in \mathfrak{W}(G)(S, F) \\ &\quad \text{with } \alpha(\pi) = u \text{ and } \beta(\pi) = v\} \end{aligned}$$

Remark: For each length-preserving transducer $\mathfrak{T}$ there exists a transducer $\mathfrak{T}'$ with $|\alpha(e)| = |\beta(e)| = 1$ such that $\tau(\mathfrak{T}) = \tau(\mathfrak{T}')$.

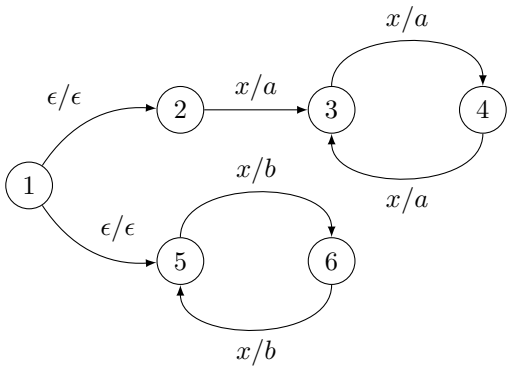
A finite transduction can equivalently be written as

$$\tau(\mathfrak{T})(u) = \beta(\alpha^{-1}(u) \cap \mathfrak{W}(G)(S, F)),\ u \in X^*$$

We also speak of **rational transduction** instead of finite transduction.

We give an example for a rational transducer:

Let $\mathfrak{T} = (G, \{x\}, \{a, b\}, \{1\}, \{3, 5\}, \alpha, \beta)$ be given by the following graph



The edges are labeled with the input/output symbols.

$\mathfrak{T}$ realizes the transduction $\tau(\mathfrak{T})$ defined by

$$\tau(\mathfrak{T})(x^n) = \begin{cases} a^n & \text{if } n \equiv 1 \bmod 2 \\ b^n & \text{if } n \equiv 0 \bmod 2 \end{cases}$$

LEMMA 5.1 (closure under inversion). *Let* $\tau(\mathfrak{T})$ *be a rational transduction. Then* $(\tau(\mathfrak{T}))^{-1}$ *is a rational transduction.*

PROOF. Let $\mathfrak{T} = (G, X, Y, S, F, \alpha, \beta)$ be a finite tranductor. Then

$$\mathfrak{T}' = (G, Y, X, S, F, \beta, \alpha)$$

is a finite transducer and

$$\tau(\mathfrak{T}') = (\tau(\mathfrak{T}))^{-1}$$

□

We define now the image of a language under a rational transduction:

DEFINITION 5.2. *Let $L \subset X^*$, then*

$$\begin{aligned} \tau(\mathfrak{T})(L) &:= \{v \in Y^* \mid \textit{there exists } u \in L \textit{ with } (u, v) \in \tau(\mathfrak{T})\} \\ &:= \bigcup_{u \in L} \tau(\mathfrak{T})(u) \end{aligned}$$

The following theorem holds:

THEOREM 5.1 (Regular languages are closed under rational transductions). *Let $L \in \mathbf{REG}(X^*)$ be a regular language and $\mathfrak{T}$ a finite transducer, then the image of L under τ is a regular language over Y^*:*

$$\tau(\mathfrak{T})(L) \in \mathbf{REG}(Y^*)$$

PROOF. By definition, the image of L can be presented as

$$\tau(\mathfrak{T})(L) = \beta\big(\alpha^{-1}(L) \cap \mathfrak{W}(G)(S, F)\big)$$

α may be continued to a monoid homomorphism from E^*, the free monoid over the edge set of G, to X^*.

We already know:

$$\begin{aligned} \alpha^{-1}(L) \in \mathbf{REG}(E^*) &\quad \text{by theorem 2, chapter II.3} \\ \mathfrak{W}(G)(S, F) \in \mathbf{REG}(E^*) &\quad \text{by lemma 4, chapter II.1} \\ \Rightarrow \alpha^{-1}(L) \cap \mathfrak{W}(G)(S, F) \in \mathbf{REG}(E^*) &\quad \text{by theorem 1, chapter II.1} \\ \Rightarrow \beta(\alpha^{-1}(L) \cap \mathfrak{W}(G)(S, F)) \in \mathbf{REG}(Y^*) &\quad \text{by theorem 7, chapter II.1} \end{aligned}$$

□

The next theorem gives an insight into the power of rational transductions.

THEOREM 5.2. *For each $L \in \mathbf{REG}(Y^*)$ there exists a length-preserving transducer $\mathfrak{T}_L$ with*

$$\tau(\mathfrak{T}_L)(\{x\}^*) = L$$

PROOF. For $L \in \mathbf{REG}(Y^*)$ there exists a finite automaton

$$\mathfrak{B} = (G, Y, S, F, \beta),\ G = (V, E)$$

with $L = \beta(\mathfrak{W}(G)(S, F))$.

For $X = \{x\}$, set $\alpha(e) = x$ for all edges $e \in E$.

Then $\alpha^{-1}(X^*) \cap \mathfrak{W}(G)(S, F) = \mathfrak{W}(G)(S, F)$, from which follows

$$\beta(\alpha^{-1}(X^*) \cap \mathfrak{W}(G)(S, F)) = L$$

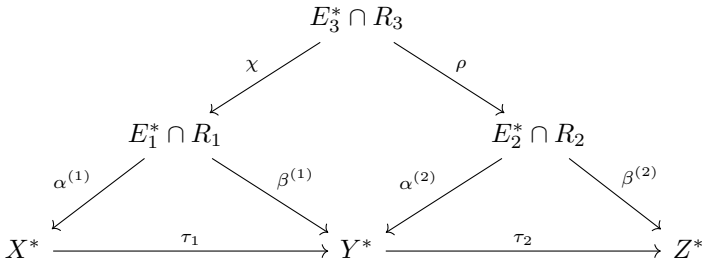
□

THEOREM 5.3. *If $\tau_1 : X^* \to Y^*$ and $\tau_2 : Y^* \to Z^*$ are finite, length-preserving transductions, then*

$$\tau_1 \circ \tau_2 : X^* \to Z^*$$

is a finite transduction.

PROOF. We want to illustrate the proof geometrically as shown in the following figure:



We have the following situation:

$$\tau_1(u) = \beta^{(1)}\big({\alpha^{(1)}}^{-1}(u) \cap R_1\big)$$

and

$$\tau_2(v) = \beta^{(2)}\big({\alpha^{(2)}}^{-1}(v) \cap R_2\big)$$

with $u \in X^*$, $v \in Y^*$ and $R_1 \subset E_1^*$, $R_2 \subset E_2^*$ regular languages and monoid homomorphisms

$$\alpha^{(1)} : E_1^* \to X^*, \quad \beta^{(1)} : E_1^* \to Y^*, \quad \alpha^{(2)} : E_2^* \to Y^*, \quad \beta^{(2)} : E_2^* \to Z^*$$

Let

$$E_3 := \{(e_1, e_2) \mid \beta^{(1)}(e_1) = \alpha^{(2)}(e_2)\}$$

and

$$R_3 := \chi^{-1}(R_1) \times \rho^{-1}(R_2) = \mathfrak{W}(S_1 \times S_2, F_1 \times F_2)$$

with monoid homomorphisms $\chi : e_3^* \to E_1^*$ and $\rho : E_3^* \to E_2^*$.

Now let $(u, v) \in \tau_1$ and $(v, w) \in \tau_2$. Then there exists paths $\omega_1 \in E_1^* \cap R_1$ and $\omega_2 \in E_2^* \cap R_2$ such that

$$u = \alpha^{(1)}(\omega_1), \quad v = \beta^{(1)}(\omega_1) = \alpha^{(2)}(\omega_2), \quad w = \beta^{(2)}(\omega_2)$$

The "parallel" path $\omega = (\omega_1(1), \omega_2(1)) \cdot (\omega_1(n), \omega_2(n))$ is an element of $E_3^* \cap R_3$ and $\chi(\omega) = \omega_1$, $\rho(\omega) = \omega_2$ (without loss of generality the length-restriction from the remark after definition 1 should hold here too).

Together this gives

$$(u, v) \in \rho \circ \beta^{(2)}\big({\alpha^{(1)}}^{-1} \circ \chi^{-1}(X^*) \cap R_3\big)$$

which means $(u, v) \in \tau_3$.

To the opposite, let $(u, v) \in \tau_3$. Then there exists a path $\omega \in E_3^* \cap R_3$ with $\chi(\alpha^{(1)})(\omega) = u$ and $\rho(\beta^{(2)})(\omega) = v$.

Let $\omega_1 = \chi(\omega)$ and $\omega_2 = \rho(\omega)$ ("projections"), then we get

$$\omega_1 \in E_1^* \cap R_1, \qquad \omega_2 \in E_2^* \cap R_2$$

By construction of E_3 it follows

$$\beta^{(1)}(\omega_1) = \alpha^{(2)}(\omega_2)$$

□

COROLLARY 5.1. *Length-preserving transductions with composition form a category.*

We generalize now our statement by allowing α and β to be arbitrary functors to X^* resp. Y^*.

Next, we investigate the case $|\alpha(\pi)| \leq |\pi|$ and $|\beta(\pi)| \leq |\pi|$. Afterwards we will reduce the general case to this special case.

Notation: Functors fulfilling the condition above are called **non-expanding**.

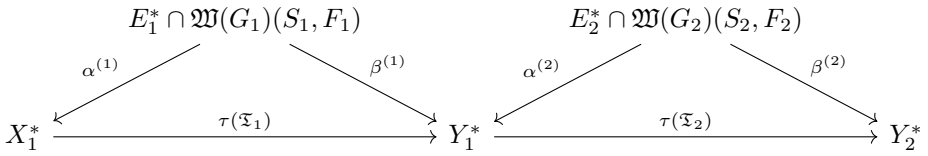
THEOREM 5.4 (The composition of non-expanding transductions is a transduction). *Let*

$$\mathfrak{T}_i = (G_i, X_i, S_i, F_i, \alpha^{(i)}, \beta^{(i)}),\ G_i = (V_i, E_i)$$

be transducers with non-expanding functors $\alpha^{(i)}$ and $\beta^{(i)}$, $i = 1, 2$.

If $Y_1 = X_2$ (the input alphabet of the second is the output alphabet of the first transducer), then the composition $\tau(\mathfrak{T}_1) \circ \tau(\mathfrak{T}_2)$ is a transduction.

PROOF. Our assumption is described by the following figure:



Remark: It should be clear that the addition of ϵ-loops ($e \in E,\ \alpha(e) = \beta(e) = \epsilon$) in the graph does not change the computed transduction.

We add to each vertex of $\mathfrak{T}_1$ and $\mathfrak{T}_2$ such a loop.

Because this does not change the computed transductions, this also is the case for the composition.

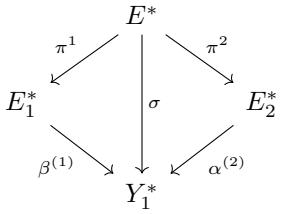
We define the vertex set of the composed transducer as $V := V_1 \times V_2$. The edge set is given by

$$E := \{(e_1, e_2) \in E_1 \times E_2 \mid \alpha^{(2)}(e_2) = \beta^{(1)}(e_1)\}$$

where

$$\begin{aligned} Q((e_1, e_2)) &= (Q(e_1), Q(e_2)) \\ Z((e_1, e_2)) &= (Z(e_1), Z(e_2)) \end{aligned}$$

With the projections $\pi^i : E^* \to E_i^*,\ i = 1, 2$, we get the commutative diagram:



and it holds: for $v \in \beta^{(1)}(E_1^*) \cap \alpha^{(2)}(E_2^*)$ there exists a path $u \in E^*$ with $\sigma(u) = v$. Define start and final state sets as follows:

$$\begin{aligned} S &= S_1 \times S_2 \\ F &= F_1 \times F_2 \end{aligned}$$

Then the composed transducer

$$\mathfrak{T} = (G, X_1, Y_2, S, F, \pi^1 \circ \alpha^{(1)}, \pi^2 \circ \beta^{(2)})$$

computes the composition of transductions τ_1 and τ_2:

$$\tau(\mathfrak{T}) = \tau(\mathfrak{T}_1) \circ \tau(\mathfrak{T}_2)$$

(exercise). □

In analogy to the corresponding result for finite automata it holds:

LEMMA 5.2. *For each transducer*

$$\mathfrak{T} = (G, X, Y, S, F, \alpha, \beta)$$

there exists a transducer

$$\mathfrak{T}' = (G', X, Y, S', F', \alpha', \beta'), \ (G' = (V', E')$$

with

$$\begin{gathered} \forall e \in E' : \alpha'(e) \in X \cup \{\epsilon\} \\ \forall e \in E' : \beta'(e) \in Y \cup \{\epsilon\} \\ \mathrm{card}(S') = 1 \\ \mathrm{card}(F') = 1 \end{gathered}$$

which computes the same transduction as $\mathfrak{T}$.

PROOF. Let $\mathfrak{T}$ be an arbitrary transducer. If an edge $e \in E$ has an input or output label longer than 1, that is $e \in E$ with $|\alpha(e)| = k > 1$ or $|\beta(e)| = l > 1$ and $m := max(k, l)$, then a path of length m can be decomposed as follows:

$$Q(e_1) \xrightarrow{e_1} Q(e_2) \xrightarrow{e_2} \ldots Q(e_m) \xrightarrow{e_m} Z(e_m)$$

(1) $|\alpha'(e_i)| \leq 1$

(2) $|\beta'(e_i)| \leq 1, \quad i = 1, \ldots, m$

(3) $\alpha(e) = \alpha'(e_1 \cdots e_m)$ and $\beta(s) = \beta'(e_1 \cdots e_m)$

For the resulting transducer $\mathfrak{T}'$ it holds

$$\tau(\mathfrak{T}) = \tau(\mathfrak{T}')$$

The construction for making this transducer initial (single start state) is completely analog to the construction in lemma 1 in chapter II.1. □

From lemma 2 and theorem 4 follows directly

THEOREM 5.5 (the composition of rational transductions is a rational transduction). *For rational transductions τ_1 and τ_2, the composition $\tau_1 \circ \tau_2$, if defined, is also a rational transduction.*

Remark: An equivalent definition of rational transductions can be given using matrices. They give a compact description for transductions which one gets by putting the output words that belong to a fixed input word into the matrix, for all state changes.

Multiplication of matrices corresponds to concatenation of paths in the graph of the transducer and to the union of the sets of output words of these paths.

The interested reader should refer to [**Ber79**] for more information.

6. Homomorphy and equivalence of finite automata

We consider again the finite automaton from chapter II.1 and turn to the question of structural relationship between finite automata.

We tackle the question how finite automata are related which accept the same language. We will formalize this by the existence of certain functors between finite automata.

To make statements about decidability wrt. the accepted languages it is an advantage to be able to give a "simplest" finite acceptor. We will prove the existence of such a "simplest" automaton.

First, we consider some simple decidability questions which get answered by the following theorems:

THEOREM 6.1. *Let $\mathfrak{A}$ be a finite automaton. Then the questions $L_{\mathfrak{A}} = \emptyset$? and $L_{\mathfrak{A}} = X^*$? are decidable.*

PROOF.

(1) "$L_{\mathfrak{A}} = \emptyset$?":

Because of $L_{\mathfrak{A}} = \emptyset \Leftrightarrow \mathfrak{W}(G)(S, F) = \emptyset$ this is a decidable problem (reachability in a finite graph). The formal proof is up to the reader.

(2) "$L_{\mathfrak{A}} = X^*$?":

We gave in chapter II.1 an effective procedure for constructing a deterministic, finite automaton $\mathfrak{A}'$ from an automaton $\mathfrak{A}$. From this deterministic automaton one can construct an automaton $\mathfrak{B}$ accepting the complement of $L_{\mathfrak{A}}$. From $L_{\mathfrak{B}} = \emptyset \Leftrightarrow L_{\mathfrak{A}} = X^*$ and (1) follows the claim.

□

DEFINITION 6.1. *$\mathfrak{A}$ is called* **weakly-equivalent** *to an automaton $\mathfrak{B}$ if the accepted languages are equal:*

$$L_{\mathfrak{A}} = L_{\mathfrak{B}}$$

THEOREM 6.2 (weak equivalence is decidable). *For finite automata $\mathfrak{A}$ and $\mathfrak{B}$ it is decidable if $L_{\mathfrak{A}} = L_{\mathfrak{B}}$.*

PROOF. Without loss of generality we may assume that $\mathfrak{A}$ and $\mathfrak{B}$ being complete and deterministic.

From $\mathfrak{A}$ and $\mathfrak{B}$ using our constructions we can directly give automata for

$$\bar{L}_{\mathfrak{A}},\ \bar{L}_{\mathfrak{B}},\ \bar{L}_{\mathfrak{A}} \cap L_{\mathfrak{B}},\ L_{\mathfrak{A}} \cap \bar{L}_{\mathfrak{A}}$$

and also for

$$L_{\mathfrak{C}} := (\bar{L}_{\mathfrak{A}} \cap L_{\mathfrak{B}}) \cup (L_{\mathfrak{A}} \cap \bar{L}_{\mathfrak{A}})$$

Because $L_{\mathfrak{A}} = L_{\mathfrak{B}} \Leftrightarrow L_{\mathfrak{C}} = \emptyset$, which is decidable by theorem 1, this proves theorem 2. □

We consider now the homomorphy between finite automata. Structure-preserving mappings between finite automata are called **automata homomorphisms**.

DEFINITION 6.2. *Let $\mathfrak{A}, \mathfrak{A}'$ be finite automata. A functor $\beta = (\beta', \beta'')$ is called an* **automata homomorphism** $\iff$

$$\beta' = (\mathfrak{W}(G), \mathfrak{W}(G'), \beta'_1, \beta'_2) \text{ is a functor,}$$

$$\beta'' : X^* \to X'^* \text{ is a monoid homomorphism}$$

and the following axioms hold:

(A1) *The following diagram commutes:*

$$\begin{array}{ccc} \mathfrak{W}(G) & \xrightarrow{\alpha} & X^* \\ \downarrow{\scriptstyle \beta'} & & \downarrow{\scriptstyle \beta''} \\ \mathfrak{W}(G') & \xrightarrow{\alpha'} & X'^* \end{array}$$

(A2) $\beta'_1(S) = S'$ *and* $\beta'_1(F) = F'$.

Let $G = (V, E)$ and $G' = (V', E')$.

β is called **length-preserving** $\iff$ $\beta'_2(e) \in E'$ *for each edge $e \in E$.*

β is called **non-expanding** $\iff$ $\beta'_2(e) \in E' \cup \{1_e \mid e \in E'\}$.

In the following we will always identify β' with β.

THEOREM 6.3. *For an automata homomorphism $\beta : \mathfrak{A} \to \mathfrak{A}'$ and*

$$\beta_2(\mathfrak{W}(G)(S, F)) = \mathfrak{W}(G')(S', F')$$

it holds

$$\beta''(L_{\mathfrak{A}}) = L_{\mathfrak{A}'}$$

PROOF.

"$\subset$":

Let $u \in L_{\mathfrak{A}}$, then there exists a path $\pi \in \mathfrak{W}(G)(S, F)$ labelled with u, i.e. $\alpha(\pi) = u$. Because $\beta_2(\pi) \in \mathfrak{W}(G')(S', F')$ by assumption, it follows $\alpha'(\beta_2(\pi)) \in L_{\mathfrak{A}'}$.

Because of the commutativity of the diagram it follows $\alpha'(\beta_2(\pi)) = \beta''(\alpha(\pi))$.

$$\Rightarrow \beta''(L_{\mathfrak{A}}) \subset L_{\mathfrak{A}'}$$

"$\supset$":

Let $u' \in L_{\mathfrak{A}'}$. Then there exists a path $\pi' \in \mathfrak{W}(G')(S', F')$ labeled with u', i.e $\alpha'(\pi') = u'$.

Because β_2 is surjective on $\mathfrak{W}(G')(S', F')$ it follows: there exists a path π with $\beta_2(\pi) = \pi'$, and because β is a functor it follows $\alpha(\pi) \in L_{\mathfrak{A}}$.

By definition, $\beta''(\alpha(\pi)) = \alpha'(\beta_2(\pi)) = \alpha'(\pi') = u'$, therefore $u' \in \beta''(L_{\mathfrak{A}})$.

$$\Rightarrow \beta''(L_{\mathfrak{A}}) \supset L_{\mathfrak{A}'}$$

Together this proves

$$L_{\mathfrak{A}'} = \beta''(L_{\mathfrak{A}})$$

□

DEFINITION 6.3. *Let $\mathfrak{A}$ and $\mathfrak{A}'$ be finite automata.*

A homomorphism $\beta : \mathfrak{A} \to \mathfrak{A}'$ is called a **reduction** *from $\mathfrak{A}$ onto $\mathfrak{A}'$ if*

$$\beta_2(\mathfrak{W}(G)(S,F)) = \mathfrak{W}(G')(S',F') \text{ and } \beta'' = id_{X^*}$$

β is called

length-preserving reduction $\iff$ *it is a reduction and the homomorphisms is length-preserving.*

non-expanding reduction $\iff$ *it is a reduction and the homomorphisms is non-expanding.*

closed homomorphism *or* **closed** $\iff$ *it is a homomorphism and it holds*

$$\beta_1{}^{-1}(S') = S \quad \text{and} \quad \beta_1{}^{-1}(F') = F$$

In the same way one defines a **closed reduction**.

We immediately get the following corollary to theorem 3:

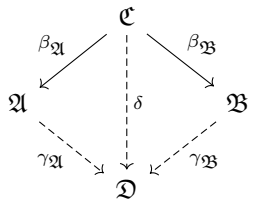
COROLLARY 6.1.

$$\beta : \mathfrak{A} \to \mathfrak{A}' \text{ reduction } \Rightarrow L_{\mathfrak{A}} = L_{\mathfrak{A}'}$$

Our goal will be to transform equivalent automata into each other using chains of reductions.

To reach this goal, we will need a number of lemmata.

LEMMA 6.1. *Let $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}$ be complete, deterministic, finite automata and let $\beta_{\mathfrak{A}} : \mathfrak{C} \to \mathfrak{A}$ and $\beta_{\mathfrak{B}} : \mathfrak{C} \to \mathfrak{B}$ be closed reductions. Then there exists a finite automaton $\mathfrak{D}$ and reductions $\gamma_{\mathfrak{A}}$ and $\gamma_{\mathfrak{B}}$ such that the following diagram commutes:*



PROOF. The idea of the proof is as follows: We create from $\mathfrak{C}$ a new automaton $\mathfrak{D}$ by putting those vertices and edges into classes which are identified under the reductions $\beta_{\mathfrak{A}}$ or $\beta_{\mathfrak{B}}$. In this way we get the reduction δ. The difficulty that arises in identifying the edges is that the labelling of the edges has to be respected.

To realize our idea we define equivalence relations on the edges and vertices of the graph $G_{\mathfrak{C}} = (V_{\mathfrak{C}}, E_{\mathfrak{C}})$ of $\mathfrak{C}$:

Let $v_1, v_2 \in V_{\mathfrak{C}}$ be vertices (states). Define

$$v_1 \underset{\mathfrak{A}}{\equiv} v_2 \quad :\iff \quad \beta_{\mathfrak{A}}(v_1) = \beta_{\mathfrak{A}}(v_2)$$
$$v_1 \underset{\mathfrak{B}}{\equiv} v_2 \quad :\iff \quad \beta_{\mathfrak{B}}(v_1) = \beta_{\mathfrak{B}}(v_2)$$

In the same way we define such an equivalence relation for the edges of the graph $G_{\mathfrak{C}}$.

We define an equivalence relation on the vertices (states) of the automaton $\mathfrak{C}$. For $v, v' \in V_{\mathfrak{C}}$:

$$v \equiv v' \quad :\Longleftrightarrow \quad \text{there is a chain of states } v = v_1, v_2, \ldots, v_k = v' \text{ with}$$
$$v_i \underset{\mathfrak{A}}{\equiv} v_{i+1} \quad \text{or} \quad v_i \underset{\mathfrak{B}}{\equiv} v_{i+1}, \quad i = 1, \ldots, k-1$$

Obviously this defines an equivalence relation on the vertex (state) set $V_{\mathfrak{C}}$. We do this in the same way for the edges.

We get equivalence classes

$$\begin{aligned} [v] &:= \{v' \in V_{\mathfrak{C}} \mid v \equiv v'\} \\ [e] &:= \{e' \in E_{\mathfrak{C}} \mid e \equiv e'\} \end{aligned}$$

We define the sets of equivalence classes by

$$\begin{aligned} \bar{V} &:= \{[v] \mid v \in V_{\mathfrak{C}}\} \\ \bar{E} &:= \{[e] \mid e \in E_{\mathfrak{C}}\} \end{aligned}$$

Claim: With $Q([e]) := [Q(e)]$ and $Z([e]) := [Z(e)]$, $e \in E_{\mathfrak{C}}$ we get well-defined source and target mappings for the edges of the graph of the automaton $\mathfrak{C}$.

Proof. Let $e' \in [e]$. To prove that $Q(e') \in [Q(e)]$, we use induction over the length of the chain $e = e_1, \ldots, e_k = e'$ where

$$s_i \underset{\mathfrak{A}}{\equiv} s_{i+1} \text{ or } s_i \underset{\mathfrak{B}}{\equiv} s_{i+1}, \ i = 1, \ldots, k-1$$

It is sufficient to show that the claim is correct for each single step. Because $\beta_{\mathfrak{A}}$ and $\beta_{\mathfrak{B}}$ are functors, it holds

$$e_i \underset{\mathfrak{A}}{\equiv} e_{i+1} \Rightarrow Q(e_i) \underset{\mathfrak{A}}{\equiv} Q(e_{i+1}) \text{ and } Z(e_i) \underset{\mathfrak{A}}{\equiv} Z(e_{i+1})$$

and

$$e_i \underset{\mathfrak{B}}{\equiv} e_{i+1} \Rightarrow Q(e_i) \underset{\mathfrak{B}}{\equiv} Q(e_{i+1}) \text{ and } Z(e_i) \underset{\mathfrak{B}}{\equiv} Z(e_{i+1})$$

From this it follows that Q and Z are well-defined on the equivalence classes. □

Let $\bar{G} = (\bar{V}, \bar{E})$ be the graph defined by the sets of vertex and edge equivalence classes. Define the labelling $\alpha_{\mathfrak{D}} : \bar{E} \to X^*$ for the edges of this graph by

$$\alpha_{\mathfrak{D}}([e]) := \alpha_{\mathfrak{C}}(e)$$

Claim: This mapping is well-defined.

Proof. Because $\beta_{\mathfrak{A}}$ is a reduction, it holds:

$$\alpha_{\mathfrak{A}}(\beta_{\mathfrak{A}}(e) = \beta''(\alpha_{\mathfrak{C}}(e)) = \alpha_{\mathfrak{C}}(e)$$

This means that for each $e' \in [e]_{\mathfrak{A}}$ we have $\alpha_{\mathfrak{A}}(\beta_{\mathfrak{A}}(e')) = \alpha_{\mathfrak{C}}(e)$. Thus our claim is correct if $e \underset{\mathfrak{A}}{\equiv} e'$. The same holds for $e \underset{\mathfrak{B}}{\equiv} e'$. Together we get that this holds for all $e \equiv e'$ so $\alpha_{\mathfrak{D}}$ is well-defined. □

To complete the definition of the automaton $\mathfrak{D}$ we have to define start and final state sets.

$$\mathfrak{D} := (\bar{G}, X, [S_{\mathfrak{C}}], [F_{\mathfrak{C}}], \alpha_{\mathfrak{D}})$$

with final state set $[F_{\mathfrak{C}}] := \{[f] \mid f \in F_{\mathfrak{C}}\}$.

Claim: The canonical mapping $e \to [e],\ v \to [v]$ is a reduction.

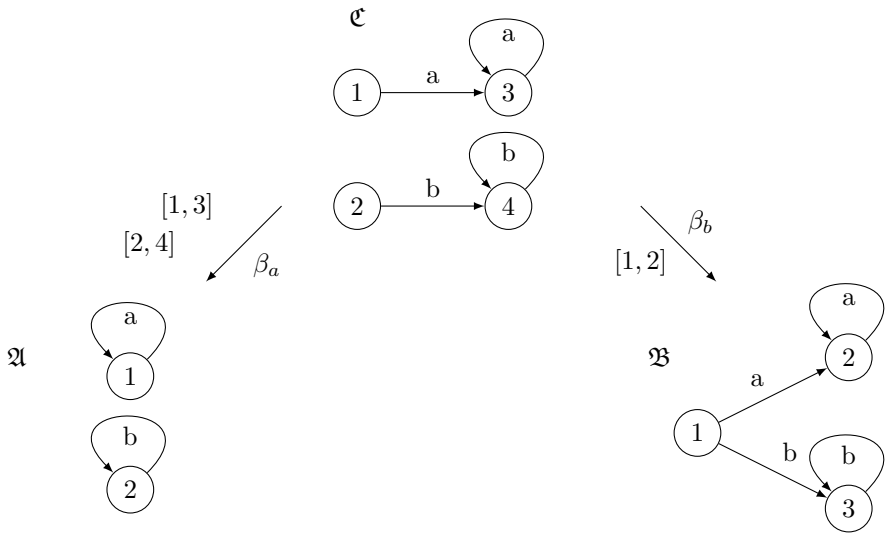
PROOF. Exercise (hint: use the fact that the automaton is complete and deterministic.) □

Finally we define the reductions $\gamma_{\mathfrak{A}}$ and $\gamma_{\mathfrak{B}}$ by

$$\begin{aligned}
\gamma_{\mathfrak{A}}(v) &:= [{\beta_{\mathfrak{A}}}^{-1}(v)], \quad v \in V_{\mathfrak{A}} \\
\gamma_{\mathfrak{B}}(v) &:= [{\beta_{\mathfrak{B}}}^{-1}(v)], \quad v \in V_{\mathfrak{B}} \\
\gamma_{\mathfrak{A}}(e) &:= [{\beta_{\mathfrak{A}}}^{-1}(e)], \quad e \in E_{\mathfrak{A}} \\
\gamma_{\mathfrak{B}}(e) &:= [{\beta_{\mathfrak{B}}}^{-1}(e)], \quad e \in E_{\mathfrak{B}}
\end{aligned}$$

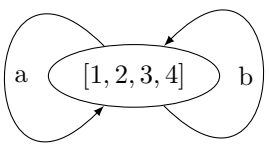
□

Remark: The requirement of our lemma, that the automata have to be complete, is essential, as the following example shows:



It holds: $L_{\mathfrak{A}} = L_{\mathfrak{B}} = L_{\mathfrak{C}} = \{a^n \mid n \in \mathbb{N}\} \cup \{b^n \mid n \in \mathbb{N}\}$.

From our construction it follows $4 \equiv 2 \equiv 1 \equiv 3$ and the automaton $\mathfrak{D}$ has the form

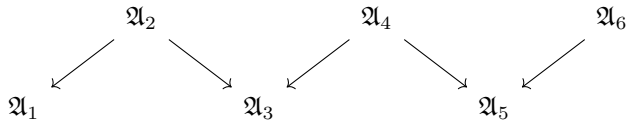


But $L_{\mathfrak{D}} = \{a, b\}^* \neq L_{\mathfrak{C}}$. It follows:

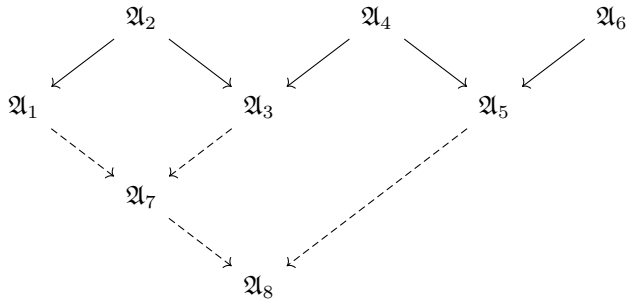
There exists no reduction from $\mathfrak{C}$ to $\mathfrak{D}$ because $\mathfrak{A}$, $\mathfrak{B}$ and $\mathfrak{C}$ are not complete *and* deterministic.

The motivation for lemma 1 can be easily seen in the following diagram:

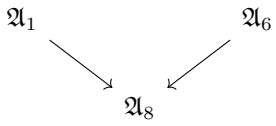
Let



be a chain of reductions. Then we can construct automata $\mathfrak{A}_7, \mathfrak{A}_8$ and reductions like

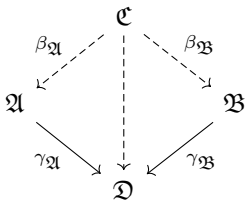


such that the diagram can be "shortened" to



LEMMA 6.2. *Let $\gamma_{\mathfrak{A}} : \mathfrak{A} \to \mathfrak{D}$ and $\gamma_{\mathfrak{B}} : \mathfrak{B} \to \mathfrak{D}$ be length-preserving reductions.*

Then there exists an automaton $\mathfrak{C}$ and reductions $\beta_{\mathfrak{A}}$ and $\beta_{\mathfrak{B}}$ such that the following diagram commutes:



PROOF. We construct the automaton

$$\mathfrak{C} = (G_{\mathfrak{C}}, X, S_{\mathfrak{C}}, F_{\mathfrak{C}}, \alpha_{\mathfrak{C}}),\ G_{\mathfrak{C}} = (V_{\mathfrak{C}}, E_{\mathfrak{C}})$$

- vertex set: $V_{\mathfrak{C}} = V_{\mathfrak{A}} \times V_{\mathfrak{B}}$
- edge set: $E_{\mathfrak{C}} = \{(e, e') \in E_{\mathfrak{A}} \times E_{\mathfrak{B}} \mid \gamma_{\mathfrak{A}}(e) = \gamma_{\mathfrak{B}}(e')\}$
- start states: $S_{\mathfrak{C}} = S_{\mathfrak{A}} \times S_{\mathfrak{B}}$
- final states: $F_{\mathfrak{C}} = F_{\mathfrak{A}} \times F_{\mathfrak{B}}$

- labelling: $\alpha_{\mathfrak{C}}((e,e')) = \alpha_{\mathfrak{A}}(e)$

$\beta_{\mathfrak{A}}$ and $\beta_{\mathfrak{B}}$ are the projections onto $\mathfrak{A}$ and $\mathfrak{B}$.

Claim: $\beta_{\mathfrak{A}}$ and $\beta_{\mathfrak{B}}$ are reductions.

We show the claim for $\beta_{\mathfrak{A}}$ only, the proof for $\beta_{\mathfrak{B}}$ is analogous.

(1) $\alpha_{\mathfrak{A}}(\beta_{\mathfrak{A}}((e,e'))) = \alpha_{\mathfrak{A}}(e) = \beta''_{\mathfrak{A}}(\alpha_{\mathfrak{C}}((e,e')) = \beta''_{\mathfrak{A}}(\alpha_{\mathfrak{A}}(e))$.
With $\beta''_{\mathfrak{A}}$ the equation above holds.

(2) $\beta_{\mathfrak{A}}(S_{\mathfrak{C}}) = S_{\mathfrak{A}}$ and $\beta_{\mathfrak{A}}(F_{\mathfrak{C}}) = F_{\mathfrak{A}}$ by definition.

(3) $\beta_{\mathfrak{A}}\Big(\mathfrak{W}(G_{\mathfrak{C}})(S_{\mathfrak{C}}, F_{\mathfrak{C}})\Big) \stackrel{!}{=} \mathfrak{W}(G_{\mathfrak{A}})(S_{\mathfrak{A}}, F_{\mathfrak{A}})$

Let $\pi \in \mathfrak{W}(G_{\mathfrak{A}})(S_{\mathfrak{A}}, F_{\mathfrak{A}})$ be an accepting path. We have to show that there exists an accepting path $\pi' \in \mathfrak{W}(G_{\mathfrak{C}})(S_{\mathfrak{C}}, F_{\mathfrak{C}})$ with $\beta_{\mathfrak{A}}(\pi') = \pi$.

Because $\gamma_{\mathfrak{A}}$ is a reduction, there exists an accepting path $\pi_{\mathfrak{D}} \in \mathfrak{W}(G_{\mathfrak{D}})(S_{\mathfrak{D}}, F_{\mathfrak{D}})$ with $\gamma_{\mathfrak{A}}(\pi) = \pi_{\mathfrak{D}}$.

Further there exists an accepting path $\pi_{\mathfrak{B}} \in \mathfrak{W}(G_{\mathfrak{B}})(S_{\mathfrak{B}}, F_{\mathfrak{B}})$ with $\gamma_{\mathfrak{B}}(\pi_{\mathfrak{B}}) = \pi_{\mathfrak{D}}$, because $\gamma_{\mathfrak{B}}$ is a reduction (and therefore is surjective).

From the paths π and $\pi_{\mathfrak{B}}$ we construct the path we are searching for:

$$\pi = e_1 \cdots e_n \qquad \pi_{\mathfrak{D}} = \gamma_{\mathfrak{A}}(e_1) \cdots \gamma_{\mathfrak{A}}(e_n) = \gamma_{\mathfrak{B}}(\pi_{\mathfrak{B}})$$

Because $\gamma_{\mathfrak{A}}$ and $\gamma_{\mathfrak{B}}$ are length-preserving it follows

$$\pi_{\mathfrak{B}} = e'_1 \cdots e'_n \text{ with } e'_i \in E_{\mathfrak{B}}$$

Now create the path $\pi_{\mathfrak{C}} = (e_1, e'_1) \cdots (e_n, e'_n) \in \mathfrak{W}(G_{\mathfrak{C}})(S_{\mathfrak{C}}, F_{\mathfrak{C}})$.

$$\begin{aligned} &\Rightarrow \quad \mathfrak{W}(G_{\mathfrak{A}})(S_{\mathfrak{A}}, F_{\mathfrak{A}}) \subset \beta_{\mathfrak{A}}(\mathfrak{W}(\mathfrak{C})(S_{\mathfrak{C}}, F_{\mathfrak{C}})) \\ &\Rightarrow \quad \mathfrak{W}(G_{\mathfrak{A}})(S_{\mathfrak{A}}, F_{\mathfrak{A}}) = \beta_{\mathfrak{A}}(\mathfrak{W}(\mathfrak{C})(S_{\mathfrak{C}}, F_{\mathfrak{C}})) \end{aligned}$$

□

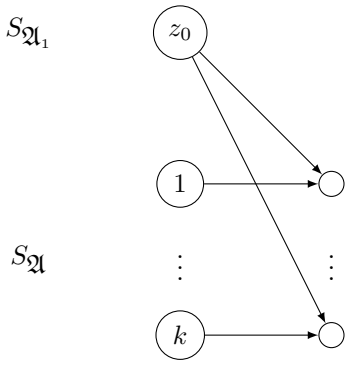
We want to turn again to the question from the beginning of this section: How are automata related that accept the same language?

First, we want to render more precisely: How does a non-deterministic automaton $\mathfrak{A}$ relate to the complete, deterministic automaton $\mathfrak{A}$'?

We remember the constructions from the first section.

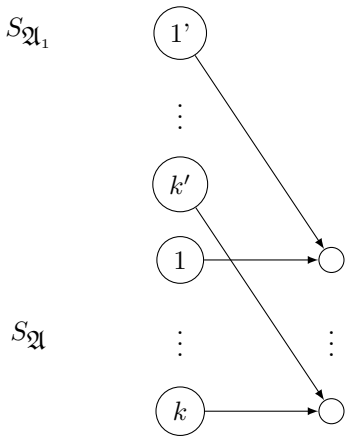
At first, we want to start with an ϵ-free automaton with $\alpha(e) \in X$ for each edge e.

Consider the step which makes this automaton initial (single start state). The construction was as follows:



Here, $\mathfrak{A}_1$ is the initial automaton created from $\mathfrak{A}$ and z_0 is the single initial state. Let $G = (V, E)$ be the graph of $\mathfrak{A}$.

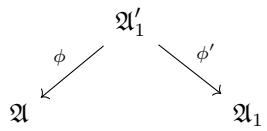
We construct an automaton $\mathfrak{A}_1'$ by adding for each start state $i \in S_{\mathfrak{A}}$ a state $i' \in V_G$ and setting $S_{\mathfrak{A}'} = \{1', \ldots, k'\}$:



The homomorphism $\phi : \mathfrak{A}_1' \to \mathfrak{A}$ is defined by $\phi_1(i) = \phi_1(i') = i,\ i = 1, \ldots, k$ and the identity otherwise. Then ϕ is a reduction.

The homomorphism $\phi' : \mathfrak{A}_1' \to \mathfrak{A}$ is defined by $\phi_1'(i') = z_0,\ i = 1, \ldots, k$ and $\phi_1'(i) = i$.

ϕ' is also a reduction thus making an automaton initial can be described by the following reduction diagram:



The construction of the final states works completely similar.

As the next step, we want to make our initial automaton $\mathfrak{A}_1$ *complete*.

This is realized by introducing a new vertex ω to which from all "incomplete" vertices new edges with the corresponding label will be drawn. Each edge starting in ω also ends in ω.

Let's name the automaton which results from this step with $\mathfrak{A}_2$. Obviously we get a reduction $\phi'' : \mathfrak{A}_1 \to \mathfrak{A}_2$.

Now we make $\mathfrak{A}_2$ deterministic. The resulting automaton shall be named $\mathfrak{A}'$.

We construct an automaton $\mathfrak{A}'_2$ by constructing edges of the following form:

$$e : (\{p_1, \ldots, p_r\}, p_i) \xrightarrow{x} (\{q_1, \ldots, q_s\}, q_j)$$

if $p_i \xrightarrow{e'} q_j \in E_{\mathfrak{A}_2}$ and $\alpha(e') = x$ for all $i = 1, \ldots, r,\ j = 1, \ldots, s$ in the construction of the deterministic automaton.

We get homomorphisms $\tilde{\phi}$ defined by

$$\tilde{\phi}(e) := (p_i \xrightarrow{x} q_j)$$

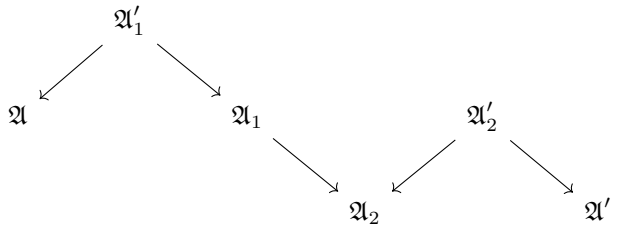
and $\hat{\phi}$ defined by

$$\hat{\phi}(e) := (\{p_1, \ldots, p_r\} \xrightarrow{x} \{q_1, \ldots, q_s\})$$

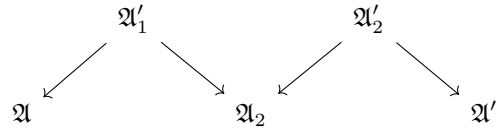
for all edges e as described above.

Both homomorphisms are reductions.

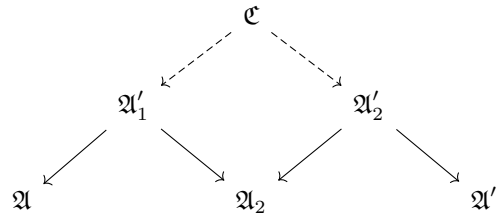
If we compose all these reductions, we get the following diagram:



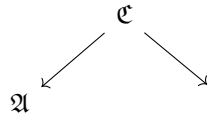
Because the composition of reductions is also a reduction, we can modify this diagram as follows:



Because all reductions in this diagram are length-preserving, we can complete this diagram to:



and finally to



Remark: All the reductions that we constructed for make an automaton initial, complete and deterministic have been length-preserving. In chapter II.1 we additionally made a construction which transforms an automaton with labelling $\alpha(e) \in$

X^* into an automaton with $\alpha(e) \in X$. We had to split edges and make the automaton ϵ-free. We can represent this also by reduction diagrams, but in that case the reductions are no longer length-preserving (non-expanding).

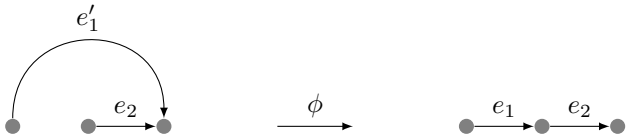
The consequence is that lemma 2 cannot be applied anymore.

We now describe these reductions informally.

(1) Splitting of edges:

Here, an edge is mapped to a path. The reduction properties are fulfilled.

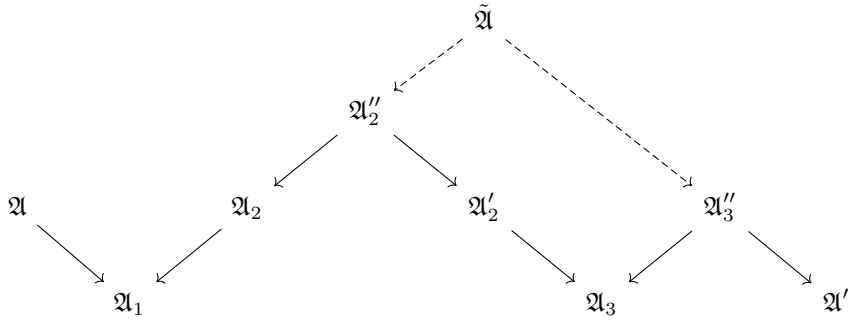
(2) Removal of ϵ-edges after insertion of bridges:



$$\begin{aligned} \phi(e_1') &= e_1 \cdot e_2, \quad Q(e_1') = Q(e_1),\ Z(e_1') = Z(e_2) \\ \phi(e_2) &= e_2, \text{ identity otherwise.} \end{aligned}$$

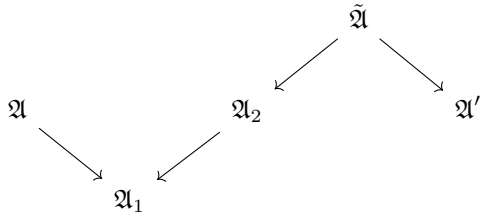
Here the properties of a reduction are also fulfilled.

We want to visualize the complete construction graphically:



The left-most reduction $(\mathfrak{A}, \mathfrak{A}_2) \to \mathfrak{A}_1$ is expanding, all other reductions are length-preserving.

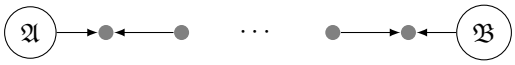
This diagram can be simplified after our previous considerations and using the categorial closure of length-preserving reductions into:



We can now concretize our initial goal of describing the relation between two *equivalent* automata.

We want to achieve the following:

$L_{\mathfrak{A}} = L_{\mathfrak{B}}$, then there should exist a diagram



We define for the complete, deterministic automaton a *normal form*:

DEFINITION 6.4. *A complete, deterministic automaton is called* **reduced** $\iff$ *each reduction* $\rho : \mathfrak{A} \to \mathfrak{B}$ *is a monomorphism (injective homomorphism).*

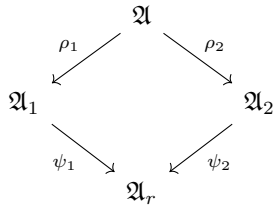
LEMMA 6.3. *For each complete, deterministic finite automaton* $\mathfrak{A}$ *there exists a reduced automaton* $\mathfrak{A}_r$ *and a closed reduction* $\rho : \mathfrak{A} \to \mathfrak{A}_r$.

If $\mathfrak{A}_r$ *does not contain superfluous elements, then it is unique up to isomorphism.*

PROOF. We prove first that every two automata which are reduced and belong to $\mathfrak{A}$ are isomorphic.

Let $\rho_i : \mathfrak{A} \to \mathfrak{A}_i,\ i = 1, 2$ be closed reductions. We can apply lemma 1 and get the following situation:

There exist a reduced automaton $\mathfrak{A}_r$ and reductions $\psi_i : \mathfrak{A}_i \to \mathfrak{A}_r$ with



By assumption, $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are reduced, therefore ψ_1 and ψ_2 are monomorphisms (injective) and ρ_1 and ρ_2 are surjective, $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are complete and deterministic

$\Rightarrow$ the images under ψ_1 and ψ_2 are equal

$\Rightarrow$ ψ_1, ψ_2 are isomorphisms on the images.

□

It remains to prove that for each automaton $\mathfrak{A}$ there exists a reduced automaton $\mathfrak{A}_r$.

This follows from the fact that the automata are finite and is shown by the following construction.

Remark: For simplicity (without loss of generality) we want to assume that our automata in the following do not contain superfluous vertices (states).

We consider now a restriction of the syntactic congruence defined in chapter I.2.

DEFINITION 6.5. $u, v \in X^*$ *are called* **congruent modulo** L

$$u \equiv v \pmod{L} : \iff \text{for all } w \in X^* \text{ holds: } u \cdot w \in L \Leftrightarrow v \cdot w \in L$$

Notation: $[w]_L := \{v \mid w \equiv v \pmod{L}\}$

$$X^*/L := \{[w]_L \mid w \in X^*\}$$

With this notatation it holds

LEMMA 6.4.

$$\operatorname{card}(X^*/L) \text{ is finite} \iff L \in \mathbf{REG}(X^*)$$

PROOF.

"$\Rightarrow$":

One sees immediately: $w \equiv w' \pmod{L} \Rightarrow w \cdot x \equiv w' \cdot x \pmod{L}$

We construct the graph $G' = (X^*/L, E)$ with edge set

$$E = \{([w], [wx]) \mid w \in X^*,\ x \in X\}$$

which by assumption is finite.

Define start and final state sets by

$$S := \{[\epsilon]\}, \quad F := \{[w] \mid w \in L\}$$

and define the labelling α by

$$\alpha(([w], [wx])) := x$$

We obtain a finite automaton

$$\mathfrak{A} = \mathfrak{A}(L) = (G', X, S, F, \alpha)$$

with the property $L_{\mathfrak{A}} = L$ (exercise).

"$\Leftarrow$":

(The proof in original book has been slightly reformulated.)

Let $L \in \mathbf{REG}(X^*)$ be a regular language, then there exists a finite acceptor $\mathfrak{A}$ accepting L. Assume that $\mathfrak{A}$ is complete and deterministic with graph $G = (V, E)$.

Consider paths π and π' both starting at some start state of $\mathfrak{A}$

$$Q(\pi) \in S,\ Q(\pi') \in S$$

with labels being congruent modulo L:

$$\alpha(\pi) \equiv \alpha(\pi') \pmod{L}$$

For any pair of paths ω, ω' that can be appended to π and π' and which have equal labels, that is

$$Q(\omega) = Z(\pi),\ Q(\omega') = Z(\pi'),\ \alpha(\omega) = \alpha(\omega')$$

it holds

$$Z(\omega) \in F \iff Z(\omega') \in F$$

(Otherwise, we would have for example $Z(\omega) \in F,\ Z(\omega') \notin F$. But from this would follow $\alpha(\pi) \cdot \alpha(\omega) \in L$ and $\alpha(\pi') \cdot \alpha(\omega) \notin L$ which violates our assumption.)

Now assume to the opposite that for each pair of paths ω, ω' that can be appended to π and π' and $\alpha(\omega) = \alpha(\omega')$ it holds

$$Z(\omega) \in F \iff Z(\omega') \in F$$

Then for any pair of paths π, π' with

$$Z(\pi) = Q(\omega),\ Z(\pi') = Q(\omega'), \quad Q(\pi) \in S,\ Q(\pi') \in S$$

it follows

$$\alpha(\pi) \equiv \alpha(\pi') \pmod{L}$$

This gives us a mapping that assigns to each vertex (state) $v \in V$ a congruence class, namely $[\alpha(\pi)]$.

From this we get: $\operatorname{card}(X^*/L)$ is finite, especially $\operatorname{card}(X^*/L) \leq \operatorname{card}(V)$. □

In our lemma we even proved a bit more:

Let $\rho_1 : V \to X^*/L$ with $\rho_1(v) = [u]_L$, if $\pi \in \mathfrak{W}(G)(S, V)$ where $Z(\pi) = v$ and $\alpha(\pi) = u$ be the mapping from the lemma. Then it holds

LEMMA 6.5. *ρ_1 induces a reduction from $\mathfrak{A}$ to $\mathfrak{A}(L)$.*

PROOF. Let $e \in E$ be an edge with $Q(e) = v, Z(e) = v'$ and $\alpha(e) = x$, then the mapping ρ_2 with

$$\rho_2(e) = e' \in E',\ Q(e') = [u] = \rho_1(v),\ Z(e') = [ux] = \rho_1(v')$$

induces a reduction. □

We get immediately

THEOREM 6.4. *If $\mathfrak{A}$ is a complete, deterministic finite automaton, then there exists a closed reduction*

$$\rho : \mathfrak{A} \to \mathfrak{A}_L$$

PROOF. Let ρ be the mapping defined by ρ_1 and ρ_2. It holds:

(1) ρ is a homomorphism

(2) $\rho_1^{-1}([L]) = F, \qquad \rho_1^{-1}([\epsilon]) = S$

(3) ρ is surjective!

ρ_1: clear, because to each $v \in V$ there exists a corresponding congruence class.

ρ_2 is surjective because $\mathfrak{A}$ is complete. □

Remark: ρ is even length-preserving.

COROLLARY 6.2. *$\mathfrak{A}(L)$ is reduced. (exercise)*

With all this we can set $\mathfrak{A}_r := \mathfrak{A}(L)$ and we have completely proved lemma 3.

The question remains: how are equivalent, reduced, complete and deterministic finite automata related?

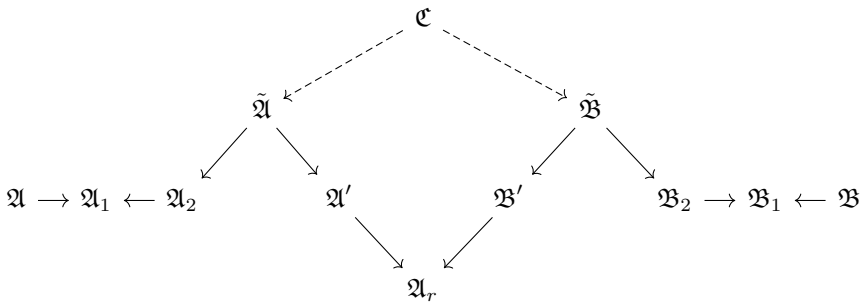
THEOREM 6.5. *If $\mathfrak{A}$ and $\mathfrak{B}$ are reduced, complete and deterministic finite automata which accept the same language, then they are isomorphic.*

PROOF. $\mathfrak{A}$ and $\mathfrak{B}$ are reduced, therefore $\mathfrak{A} \cong \mathfrak{A}(L) \cong \mathfrak{B}$. □

COROLLARY 6.3. *If $\mathfrak{A}$ is equivalent to $\mathfrak{B}$ then there exists a chain of reductions*

$$\mathfrak{A} \longrightarrow \mathfrak{A}_1 \longleftarrow \mathfrak{A}_2 \longleftarrow \mathfrak{C} \longrightarrow \mathfrak{B}_1 \longleftarrow \mathfrak{B}$$

PROOF. We look at the construction we have done above:

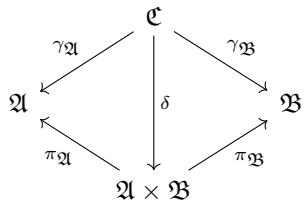


$\mathfrak{A}_r$ exists by theorems 4 and 5. $\mathfrak{C}$ exists by lemma 2. □

LEMMA 6.6. *Let $\gamma_{\mathfrak{A}} : \mathfrak{C} \to \mathfrak{A}$, $\gamma_{\mathfrak{B}} : \mathfrak{C} \to \mathfrak{B}$ be length-preserving reductions. Then there exists an automaton $\mathfrak{A} \times \mathfrak{B}$ and reductions*

$$\pi_{\mathfrak{A}} : \mathfrak{A} \times \mathfrak{B} \to \mathfrak{A}$$
$$\pi_{\mathfrak{B}} : \mathfrak{A} \times \mathfrak{B} \to \mathfrak{B}$$

such that the following diagram commutes:



PROOF. Let

$$G := (V_{\mathfrak{A}} \times V_{\mathfrak{B}}, \{(e, e') \in E_{\mathfrak{A}} \times E_{\mathfrak{B}} \mid \alpha_{\mathfrak{A}}(e) = \alpha_{\mathfrak{B}}(e')\})$$

Let the finite automaton $\mathfrak{A} \times \mathfrak{B}$ be defined by

$$\mathfrak{A} \times \mathfrak{B} := (G, X, S_{\mathfrak{A}} \times S_{\mathfrak{B}}, F_{\mathfrak{A}} \times F_{\mathfrak{B}}, \alpha)$$

Define

$$\begin{aligned} \delta(v) &= (\gamma_{\mathfrak{A}}(v), \gamma_{\mathfrak{B}}(v)),\ v \in V_{\mathfrak{C}} \\ \delta(e) &= (\gamma_{\mathfrak{A}}(e), \gamma_{\mathfrak{B}}(e)),\ e \in E_{\mathfrak{C}} \end{aligned}$$

Then δ is a homomorphism from $\mathfrak{C}$ to $\mathfrak{A} \times \mathfrak{B}$.

Now $\gamma_{\mathfrak{A}} := \delta \circ \pi_{\mathfrak{A}}$ and $\gamma_{\mathfrak{B}} := \delta \circ \pi_{\mathfrak{B}}$ are reductions

$$\Rightarrow \pi_{\mathfrak{A}} \text{ and } \pi_{\mathfrak{B}} \text{ are reductions.}$$

□

With lemma 6 we get

THEOREM 6.6. *If $\mathfrak{A}$ is equivalent to $\mathfrak{B}$, then $\mathfrak{A}$ can be transformed into $\mathfrak{B}$ by the following chain of reductions:*

$$\mathfrak{A} \longrightarrow \mathfrak{A}_1 \longleftarrow \mathfrak{A}_2 \xleftarrow{\pi_{\mathfrak{A}_2}} \mathfrak{A}_2 \times \mathfrak{B}_2 \xrightarrow{\pi_{\mathfrak{B}_2}} \mathfrak{B}_2 \longrightarrow \mathfrak{B}_1 \longleftarrow \mathfrak{B}$$

PROOF. Clear using our constructions from above. □

We can give now a **decision algorithm** for the equivalence of finite automata $\mathfrak{A}$ and $\mathfrak{B}$.

(1) From $\mathfrak{A}$ and $\mathfrak{B}$ construct the automata $\mathfrak{A}_2$ and $\mathfrak{B}_2$.

(2) Check if $\mathfrak{A}_2 \xleftarrow{\pi_{\mathfrak{A}_2}} \mathfrak{A}_2 \times \mathfrak{B}_2 \xrightarrow{\pi_{\mathfrak{B}_2}} \mathfrak{B}_2$ are reductions. If this is the case, $L_{\mathfrak{A}} = L_{\mathfrak{B}}$.

We want to specify the **complexity** of this decision algorithm:

As measure of complexity of $\mathfrak{A}$ we define

$$\|\mathfrak{A}\| := \sum_{e \in E} |\alpha(e)|$$

"Splitting" the edges:	$O(\|\mathfrak{A}\|)$
Creating the bridges:	$(\mathrm{card}(E))^2$
Transitive closure of ϵ-graph	$O(\|\mathfrak{A}\|^3)$
Construction of $\mathfrak{A} \times \mathfrak{B}$:	$\|\mathfrak{A}\| \cdot \|\mathfrak{B}\|$
Test for reduction	
1. Removal of superfluous states:	$O(\|\mathfrak{A}\|^3)$
2. Surjectivity test on the rest:	$O(2^{\|\mathfrak{A}\|})$

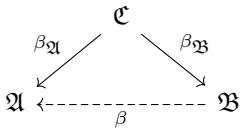
The test for surjectivity takes most of the time and dominates the complexity of the decision algorithm. Our algorithm takes the same time as the other algorithms from the literature [**AHU76**] because we start from non-deterministic automata.

Remark: If we would have required for our automata $\mathfrak{A}$ and $\mathfrak{B}$ that $\alpha_{\mathfrak{A}}(e) \in X$ and $\alpha_{\mathfrak{B}}(e) \in X$, then the diagram from the corollary to theorem 5 could have been simplified to $\mathfrak{A} \longleftarrow \mathfrak{C} \longrightarrow \mathfrak{B}$. For the compexity of our decision algorithm this however would be insignificant.

EXERCISE 6.1.

(1) *Let $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$ be finite automata, $\beta_{\mathfrak{A}} : \mathfrak{C} \to \mathfrak{A}$, $\beta_{\mathfrak{B}} : \mathfrak{C} \to \mathfrak{B}$ reductions and $\equiv_{\mathfrak{A}}, \equiv_{\mathfrak{B}}$ the equivalence relations from the proof of lemma 1.*

Show: $(\equiv_{\mathfrak{A}} \subset \equiv_{\mathfrak{B}}) \Rightarrow$ there exists a reduction $\beta : \mathfrak{B} \to \mathfrak{A}$ such that the following diagram commutes:



(2) *Let $\mathfrak{A}$ be a complete, deterministic finite automaton, $\mathfrak{B}$ a finite automaton and $\beta : \mathfrak{A} \to \mathfrak{B}$ a homomorphism with $\beta(E_{\mathfrak{A}}) = E_{\mathfrak{B}}$. Show:*

- *β is a reduction*
- *The given requirements are necessary.*

7. Regular sets in $X^{(*)}$, $\mathbf{REG}(X^{(*)})$

In the first section of this chapter, we considered $\mathbf{REG}(X^*)$, the class of regular languages over the free monoid X^*.

We replace now the free monoid by an important special case of a monoid, namely the polycyclic monoid $X^{(*)}$ which is the syntactic monoid of the Dyck language over the alphabet $X \cup X^{-1}$.

We represent the elements of $X^{(*)}$ by their *reduced words* (see chapter I.3).

DEFINITION 7.1. *Let $|w|$ be the reduced word of $w \in (X \cup X^{-1})^*$ with respect to the polycyclic monoid $X^{(*)}$, and let $L \subset (X \cup X^{-1})^*$ be a language. Then*

$$|L| \quad := \quad \{|w| \mid w \in L,\ |w| \not\equiv 0\}$$

is the **set of reduced words** *of L.*

DEFINITION 7.2.

$$|\mathbf{REG}|(X^{(*)}) \quad := \quad \{L \subset (X \cup X^{-1})^* \mid \textit{there exists } L' \in \mathbf{REG}(X^{(*)}) \textit{ with } L = \{|w| \mid [w] \in L'\}\}$$

Our goal is to show the inclusion

$$|\mathbf{REG}|(X^{(*)}) \subset \mathbf{REG}((X \cup X^{-1})^*)$$

This inclusion does not state that the word problem of $X^{(*)}$ is rational, or in other words that $[w]$ is a rational set. But the structure of $\mathbf{REG}(X^{(*)})$ is determined for the biggest part by $\mathbf{REG}((X \cup X^{-1})^*)$.

Of course it is

$$|\mathbf{REG}|(X^{(*)}) \neq \mathbf{REG}((X \cup X^{-1})^*)$$

But we can completely characterize $|\mathbf{REG}|(X^{(*)})$ as a subset of $\mathbf{REG}((X \cup X^{-1})^*)$.

To do so, we first prove some lemmata.

LEMMA 7.1.

$$L \in \mathbf{REG}((X \cup X^{-1})^*) \Rightarrow |L| \in \mathbf{REG}((X \cup X^{-1})^*)$$

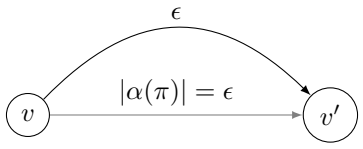
(The proof from the orginal book has been reformulated for better readability.)

PROOF. Let $\mathfrak{A} = (G, X \cup X^{-1}, S, F, \alpha)$ be a finite acceptor of L. We construct in several steps an acceptor $\mathfrak{A}'$ for $|L|$.

This construction will play an exceptional role in later chapters.

Without loss of generality we may assume that for each edge e of our graph $G = (V, E)$ it holds $\alpha(e) \in X \cup X^{-1} \cup \{\epsilon\}$.

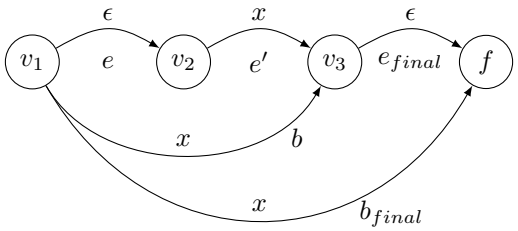
Step 1: If there is a path $\pi \in \mathfrak{W}(G)(v, v')$ with $|\alpha(\pi)| = \epsilon$, then we add a new edge $e : v \to v'$ to E and set $\alpha(e) = \epsilon$. Any ϵ-loops are removed.



We do this for each such pair $(v, v') \in V \times V$ and get a graph $G_1 = (V_1, E_1)$ with labelling α.

Step 2: For consecutive edges $e : v_1 \xrightarrow{\epsilon} v_2,\ e' : v_2 \xrightarrow{x} v_3 \in E_1,\ x \in X \cup X^{-1}$, we add a "bridge" $b : v_1 \xrightarrow{x} v_3$ with label x.

For each edge $e_{final} : v_3 \xrightarrow{\epsilon} f$, where $f \in F$ is a final state, we add a "bridge" $b_{final} : v_1 \xrightarrow{x} f$ with label x.



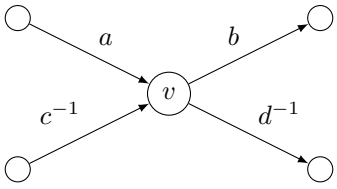
If we do this for all such pairs of edges (e, e'), we get a graph $G_2 = (V_2, E_2)$ with labelling α as described.

Step 3: We remove all edges $e \in E_2$ where $\alpha(e) = \epsilon$. The resulting graph shall be $G_3 = (V_3, E_3)$ with labelling α.

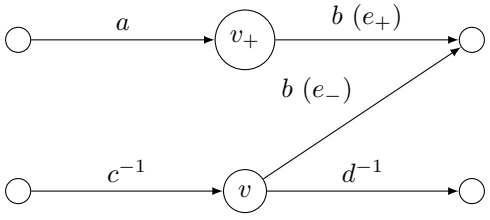
Because of our construction we get:

$$L_{\mathfrak{A}_3} = \{v \mid v \text{ is reduced word of } w \in L_{\mathfrak{A}} \text{ wrt. } x \cdot x^{-1} = 1\}$$

Step 4: We cut all paths whose label contains the word $a \cdot d^{-1},\ a, d \in X,\ a \neq d$.



To do so, we duplicate vertices and edges of G_3. The graph structure above will be transformed into this structure:



Let v be a vertex which is the target of an edge with label $a \in X$ and the source of an edge with label $d^{-1} \in X^{-1}$.

For each vertex v we add a copy v_+ to the graph and add new edges as follows:

- All edges entering v with label $a \in X$ are rerouted to the copy v_+.
- All edges entering v with label $c^{-1} \in X^{-1}$ are left unchanged.
- All edges e leaving v with label $b \in X$ will be duplicated into edges e_+ and e_-. Edge e_+ leads from v_+ to $Z(e)$ and the other edge e_- from v to $Z(e)$. The label for both new edges is $\alpha(e) = b$.
- All edges leaving v with label $d^{-1} \in X^{-1}$ stay unchanged.

We do this with every such vertex of G_3 and get a graph G'.

With start and final state sets $S' := S$ and $F' := F \cup \{v^+ \mid v \in F\}$ we get a finite automaton

$$\mathfrak{A}' = (G', X \cup X^{-1}, S', F', \alpha')$$

Claim: $L_{\mathfrak{A}'} \stackrel{!}{=} |L|$

We observe that $L_{\mathfrak{A}'}$ contains only reduced words which are different from 0. Additionally, the reversal of the construction of G' from G_3 is a functor which preserves the edge labelling.

Therefore $L_{\mathfrak{A}'} \subset |L|$.

Let to the opposite π be a path in $\mathfrak{W}(G_3)$ with label $\alpha(\pi) = |\alpha(\pi)|$, then there is no $(+,-)$-change in the label of π.

Therefore the path π stays unchanged in G' as long as the label of the edges is in X^{-1}.

If the label changes, then the path switches to the $+$-vertices and stays "positive" until its target vertex. Let $\pi' \in \mathfrak{W}(G')$ denote this path belonging to π, then $\alpha(\pi) = \alpha'(\pi')$.

Therefore it holds $|L| \subset L_{\mathfrak{A}'}$.

Together we get $|L| = L_{\mathfrak{A}'}$. □

In this proof we showed even more, we summarize this in the following corollary.

COROLLARY 7.1. *To each path*

$$\pi = e_1 \cdots e_k \in \mathfrak{W}(G)(S, -),\ e_i \in E$$

with $\alpha(e_k) \neq \epsilon$ and $\alpha(\pi) \neq \epsilon$ there is assigned a unique ϵ-free path $\pi' \in \mathfrak{W}(G_3)(S, -)$ with $\alpha(\pi) = \alpha(\pi')$.

Additionally, we can define a functor $\nu : \mathfrak{W}(G_3) \to \mathfrak{W}(G)$ such that the following holds:

For each accepted word $w \in L$ there exists an accepting path

$$\pi = e_1 \cdots e_n \in \mathfrak{W}(G_3)(S, F)$$

with $e_i \in E_3,\ \alpha(e_i) \neq \epsilon$ such that $\alpha(\nu(\pi)) = w$.

PROOF. Exercise for the reader. □

Our construction shows even more: the number of paths π with label $\alpha(\pi) = w$ for a given $w \in |L|$ is the same in $\mathfrak{W}(G_3)$ as in $\mathfrak{W}(G')$. Therefore this number is in $\mathfrak{W}(G')$ less or equal to the number in $\mathfrak{W}(G_2)$. A similar statement holds for $\mathfrak{W}(G_1)$ and $\mathfrak{W}(G)$.

This fact, which will be of importance later in this book, is summarized in the following corollary. We define

DEFINITION 7.3.

$$\mathrm{INDEX}(|\mathfrak{A}|, w) := \mathrm{card}(\{\pi \in \mathfrak{W}(G)(S, F) \mid |\alpha(\pi)| = w\})$$

COROLLARY 7.2.

$$\mathrm{INDEX}(|\mathfrak{A}|, w) \geq \mathrm{INDEX}(|\mathfrak{A}'|, w), \;\; w \in L_{\mathfrak{A}'}$$

Remark: Because of $L_{\mathfrak{A}'} = |L_{\mathfrak{A}'}|$ it holds

$$\mathrm{INDEX}(|\mathfrak{A}'|, w) = \mathrm{INDEX}(\mathfrak{A}', w)$$

Now we can prove the announced theorem.

THEOREM 7.1.

$$|\mathbf{REG}|(X^{(*)}) = \{|L| \mid L \in \mathbf{REG}((X \cup X^{-1})^*)\}$$

PROOF. The inclusion "$\supset$" is clear.

"$\subset$": Every functor $\alpha : \mathfrak{W}(G) \to X^{(*)}$ can be continued to a functor $\alpha' : \mathfrak{W}(G) \to (X \cup X^{-1})^*$ by defining $\alpha'(e) := |w|$ for each edge $e \in E$ with label $\alpha(e) = [w]$.

α' is continued to a functor and for the finite automaton

$$\mathfrak{A}' = (G', X \cup X^{-1}, S', F', \alpha')$$

it holds:

$$L_{\mathfrak{A}} = \{[w] \mid w \in L_{\mathfrak{A}'}\}$$

and with lemma 1 we get:

$$L_{\mathfrak{A}} = \{[w] \mid w \in |L_{\mathfrak{A}'}|\}$$

which means

$$|L_{\mathfrak{A}}| = \{|w| \mid w \in L_{\mathfrak{A}'}\}$$

□

The next theorem gives an even more precise characterization for $|\mathbf{REG}|(X^{(*)})$.

THEOREM 7.2. *Let $\mathfrak{A}$ be a finite automaton*

$$\mathfrak{A} = (G, X \cup X^{-1}, S, F, \alpha)$$

where $G = (V, E)$ and $\mathrm{card}(E) = c \in \mathbb{N}$.

Then there exist c pairs (L_i^+, L_i^-) with

$$L_i^+ \in \mathbf{REG}(X^*), \qquad L_i^- \in \mathbf{REG}((X^{-1})^*)$$

such that

$$|L_{\mathfrak{A}}| = \bigcup_{i=1}^{c} L_i^- \cdot L_i^+$$

PROOF. The proof follows directly from the construction of the automaton $\mathfrak{A}'$.

If V' is the set of those vertices v which have an associated vertex v^+ in G', then every path $\pi \in \mathfrak{W}(G')(S', F')$ can be uniquely decomposed into a product $\pi_1 \cdot \pi_2$ where the target $Z(\pi_1)$ is such a vertex v.

Therefore one can write

$$|L_{\mathfrak{A}}| = \bigcup_{v \in V} \left(L^-_{\mathfrak{A}(S,v)} \cdot L^+_{\mathfrak{A}(v,F)} \right)$$

□

COROLLARY 7.3.

$$\mathrm{INDEX}(\mathfrak{A}', w) = \sum_{\substack{v \in V' \\ w_1 \cdot w_2 = w}} \big(\mathrm{INDEX}(\mathfrak{A}'(S, v), w_1) \cdot \mathrm{INDEX}(\mathfrak{A}'(v, F), w_2) \big)$$

LEMMA 7.2. *Let* $L = L^+_{\mathfrak{A}} \subset X^*$ *and* $L' = L^-_{\mathfrak{A}} \subset (X^{-1})^*$.

Then there exist

$$L_1^+, \ldots, L_p^+ \in \mathbf{REG}(X^*), \quad L_1^-, \ldots, L_q^- \in \mathbf{REG}((X^{-1})^*)$$

with

$$|L \cdot L'| = \bigcup_{i=1}^{p} L_i^+ \ \cup \ \bigcup_{j=1}^{q} L_j^-$$

Here $p, q \leq \mathrm{card}(F(\mathfrak{A}'))$.

PROOF. We construct $\mathfrak{A} = \mathfrak{A}^+ \circ \mathfrak{A}^-$ with $L_{\mathfrak{A}} = L \cdot L'$ and thereafter $\mathfrak{A}'$ with $L_{\mathfrak{A}'} = |L_{\mathfrak{A}}|$.

Because for every reduced word $w \in (X \cup X^{-1})^*$ it holds $w = w^- \cdot w^+$ with $w^- \in (X^{-1})^*,\ w^+ \in X^*$, the first part of the lemma is proved.

From our construction of $\mathfrak{A}'$ it follows that the final state set F of $\mathfrak{A}'$ is split into states for sets from $(X^{-1})^*$ and those for sets from X^*. From that follows the rest of the lemma. □

In the following, we want to investigate the set of regular sets which can be defined with a fixed graph and to prove two simple but important properties of these sets.

Let $G = (V, E)$ be a graph and $\alpha : E \to (X \cup X^{-1})^*$. We may assume again that $\alpha(e) \in X \cup X^{-1} \cup \{\epsilon\}$.

We define

$$\mathcal{R}(G) := \{ L_{\mathfrak{A}(\tilde{S}, \tilde{F})} \mid \mathfrak{A} = (G, X \cup X^{-1}, S, F, \alpha),\ \tilde{S} \subset V,\ \tilde{F} \subset V \}$$

For $c := \mathrm{card}(V)$ it trivially holds

$$\mathrm{card}(\mathcal{R}(G)) \leq 2^{2 \cdot c}$$

Remark: The set $\mathcal{R}(G)$ is in general not closed under union, but it contains a simple base for the union-closure $<\mathcal{R}(G)>$.

For shorter notation, we set $\mathcal{R}(\tilde{S}, \tilde{F}) := L_{\mathfrak{A}(\tilde{S},\tilde{F})}$.

LEMMA 7.3.

$$\mathcal{R}(\tilde{S}, \tilde{F}) = \bigcup_{\substack{v \in \tilde{S} \\ v' \in \tilde{F}}} \mathcal{R}(v, v')$$

PROOF. Exercise for the reader. □

Let $\mathfrak{A}'$ be the automaton constructed as in lemma 1. We construct the graph $G' = (V', E')$ from the graph $G = (V, E)$. Define

$$\mathcal{B}(G') := \{L_{\mathfrak{A}'(v,v')} \mid v, v' \in V'\}$$

LEMMA 7.4. *Let $L^+ \subset X^*, L^- \subset (X^{-1})^*$ and $L^+, L^- \in \mathcal{B}(G')$. Then $|L^+ \cdot L^-|$ is contained in the union-closure $<\mathcal{B}(G')>$ of $\mathcal{B}(G')$.*

PROOF. Let $L^+ = L_{\mathfrak{A}'(v,v')}$ and $L^- = L_{\mathfrak{A}'(q,q')}$.

We create an automaton

$$\mathfrak{A}'' := \mathfrak{A}'(v, v') \circ \mathfrak{A}'(q, q')$$

with $L_{\mathfrak{A}''} = L^+ \cdot L^-$.

We get $\mathfrak{A}''$ by connecting an instance of $\mathfrak{A}'(q, q')$ using an ϵ-edge which leads from v' to q. Now all words $w \in L_{\mathfrak{A}''}$ have the form $w = w^+ \cdot w^-$.

But we are only interested in reduced words.

There are three cases:

$$|w^+ \cdot w^-| = \begin{cases} w_1^+ \\ 0 \\ w_1^- \end{cases}$$

Because every path π of $\mathfrak{A}''$ leading from v to q' is divided into two sections

$$\pi = \pi^+ \circ e \circ \pi^- \text{ where } \pi^+ : v \to v' \in \mathfrak{A}'(v, v') \text{ and } \pi^- : q \to q' \in \mathfrak{A}'(q, q')$$

there exists in case $|w^+ \cdot w^-| = w_1^+$ a decomposition $\pi^+ = \pi_1^+ \circ \pi_2^+$ with labels $\alpha(\pi_1^+) = w_1^+$ and $|\alpha(\pi_2^+ \circ e \circ \pi^-)| = \epsilon$.

Opposite direction:

If $\bar{\pi}_1$ is a path with $Q(\bar{\pi}_1) = Q(\pi_1^+),\ Z(\bar{\pi}_1) = Z(\pi_1^+)$, then $\alpha(\bar{\pi}_1) \in |L^+ \cdot L^-|$.

The case for $|w^+ \cdot w^-| = w_1^-$ is symmetrical.

Therefore there exist sets $V^+,\ V^- \subset V'$ with

$$|L^+ \cdot L^-| = \bigcup_{v^+ \in V^+} \mathcal{R}(v, v^+) \ \cup \ \bigcup_{q^- \in V^-} \mathcal{R}(q^-, q')$$

□

Remark: These facts will be useful in connection to the normal-form theorems for context-free grammars (chapter IV). This is especially true for the following

COROLLARY 7.4. *If*

$$\mathcal{B}(G', S, F) = \{L^+_{\mathfrak{A}'(S,v)},\ L^-_{\mathfrak{A}'(q,F)} \mid v, q \in V\}$$

Then $\mathcal{B}(G', S, F)$ *is a union-base for*

$$\{|L^+ \cdot L^-| \mid L^+, L^- \in \mathcal{B}(G', S, F)\}$$

This corollary gives us a smaller base. For fixed S and F we can represent every set $|L^+ \cdot L^-|$ as the union of at most $2 \cdot \mathrm{card}(V)$ sets of type $L^+_{\mathfrak{A}(S,v)}$ resp. $L^-_{\mathfrak{A}(q,F)}$.

We assign now to $\mathcal{B}(G)$ a system

$$\begin{array}{rcl} \mathcal{P}(G) & \subset & \mathcal{B}(G) \times \mathcal{B}(G) \cdot \{;\} \cdot \mathcal{B}(G) \\ & \cup & \{\epsilon\} \times \mathcal{B}(G) \end{array}$$

in the following way:

$$\begin{array}{rcl} L \to L^+; L^- \in \mathcal{P}(G) & \iff & \text{in the representation } L^+ \cdot L^- = \cup L_i \text{ from lemma 2} \\ & & \text{there exists an index } i \text{ with } L = L_i \\ \epsilon \to L \in \mathcal{P}(G) & \iff & \epsilon \in L \end{array}$$

LEMMA 7.5. $\epsilon \in |L_1 L_2 \cdots L_n|$ *with* $L_i \in \mathcal{B}(G) \iff L_1; L_2; \cdots; L_n$ *can be reduced to* ϵ *using the rules from* $\mathcal{P}(G)$.

PROOF. "$\Rightarrow$":

$$\begin{array}{rcl} \epsilon \in |L_1 L_2 \cdots L_n| & \Rightarrow & \text{there exists } w_i \in L_i,\ i = 1, \ldots, n, \text{ with } |w_1 \cdots w_n| = \epsilon \\ & \Rightarrow & \text{there exists an index } j \text{ with } L_j \subset X^*,\ L_{j+1} \subset (X^{-1})^* \end{array}$$

Because $|w_1 \cdots w_n| = \epsilon$ it holds $|w_j \cdot w_{j+1}| \neq 0$.

Therefore $w_j \cdot w_{j+1}$ can be reduced to a word $\bar{w}_j$ with $\bar{w}_j \in X^*$ or $\bar{w}_j \in (X^{-1})^*$.

In the decomposition $L_j \cdot L_{j+1} = \cup L'_i$ there exists an L'_j with $\bar{w}_j \in L'_j$ and it holds $L'_j \to L_j; L_{j+1} \in \mathcal{P}(G)$.

If we replace $L_j; L_{j+1}$ by L'_j then we have again

$$\epsilon \in |L_1 \cdots L_{j-1} \cdot L'_j \cdot L_{j+2} \cdots L_n|$$

Inductively we may assume that $L_1 \cdots L_n$ can be reduced using $\mathcal{P}(G)$ to some L' and that $\epsilon \in |L'|$ holds.

Because $|L'| = L$ it follows $\epsilon \in L$. Therefore $L_1; \cdots; L_n$ can indeed be reduced to ϵ.

"$\Leftarrow$":

Let $L_1; \cdots; L_n$ be reducible to ϵ using $\mathcal{P}(G)$ and let $L'_j \to L_j; L_{j+1} \in \mathcal{P}(G)$.

Then for each $w_i \in L_i,\ i = 1, \ldots, j-1, j+2, \ldots, n$ and $\bar{w}_i \in L'_j$ there exist $w_j \in L_j,\ w_{j+1} \in L_{j+1}$ with

$$|w_1 \cdots w_n| = |w_1 \cdots w_{j-1} \cdot \bar{w}_j \cdot w_{j+2} \cdots w_n|$$

If $\epsilon \in L_j$ then $\epsilon \to L_j \in \mathcal{P}(G)$ and it holds $\bar{w}_j = \epsilon$. Therefore

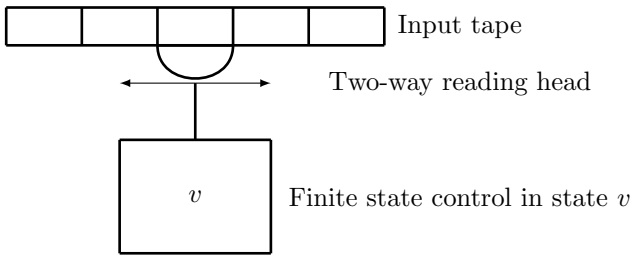
$$|w_1 \cdots w_{j-1} \cdot w_{j+1} \cdots w_n| = |w_1 \cdots w_n|$$

By induction it follows: if $L_1; \cdots ; L_n$ can be derived in $\mathcal{P}(G)$ from ϵ, then $\epsilon \in |L_1 \cdots L_n|$. $\square$

8. The finite 2-way automaton

We start now with generalizing the concept of the finite automaton as described in chapter II.4.

We consider a finite automaton where the reading head can traverse the input tape in two directions as indicated by the following figure:



The representation of such an automaton by a state graph is not trivial. For this reason, we first define this type of automaton as usual.

Afterwards, we will show how to represent this automaton by a state graph.

(The definition from the original book has been adapted to be closer to the common namings.)

DEFINITION 8.1. *A* **(non-deterministic) two-way finite automaton** *is defined by*

$$\mathfrak{B} = (X, Q, \delta, S, F, \{L, R\})$$

where

- X *is the input alphabet with* $\{L, R\} \cap X = \emptyset$
- Q *is the state set with* $\{L, R\} \cap Q = \emptyset$
- $S \subset Q$ *is the set of start states*
- $F \subset Q$ *is the set of final states of* $\mathfrak{B}$
- $\delta \subset (Q \times X) \to (Q \times \{L, R\})$ *is a relation describing the functioning of* $\mathfrak{B}$

If $\delta : (Q \times X) \to Q \times \{L, R\}$ *is a map, then* $\mathfrak{B}$ *is called a* **complete, deterministic two-way finite automaton**.

DEFINITION 8.2. *A* **computation** *of* $\mathfrak{B}$ *on the input word* $v \in X^*$ *is a sequence of single computation steps*

$$f_k = (q_0 v \to u_1 q_1 v_1 \to u_2 q_2 v_2 \to \ldots \to u_k q_k v_k)$$

with $u_i, v_i \in X^*,\ q_i \in Z$ *for* $i = 1, \ldots, k$ *and* $q_0 \in S$.

A single computation step $u_i q_i v_i \to u_{i+1} q_{i+1} v_{i+1}$ is defined by the following alternatives:

- If $v_i = a v_i'$, then $u_{i+1} = u_i a,\ v_{i+1} = v_i'$, if $(q_{i+1}, R) \in \delta(q_i, a)$,
  i.e. the reading head moves to the right and the automaton changes into state q_{i+1}, if it reads a in state q_i.
- If $u_i = u_i' b,\ v_i = a v_i'$, then $u_{i+1} = u_i'$ and $v_{i+1} = b v_i$, if $(q_{i+1}, L) \in \delta(q_i, a)$,
  i.e. the reading head moves to the left and the automaton changes into state q_{i+1}, if it reads a in state q_i.

DEFINITION 8.3. *A word $w \in X^*$ is* **accepted by $\mathfrak{B}$** *if there exists a computation f_k with $v_k = \epsilon$ and $q_k \in F$.*

The language of $\mathfrak{B}$ is

$$L_{\mathfrak{B}} = \{w \in X^* \mid w \text{ is accepted by } \mathfrak{B}\}$$

We want to consider again the monoid $X^{[*]}$ (H-group, involutive monoid), see chapter I.3, and construct an automaton

$$\mathfrak{D} = (G, X^{[*]}, S, F, \alpha)$$

with graph $G = (V, E)$ as follows:

- $V = Q \times \{L_1, L_2, R_1, R_2\}$
- Edge set E and labelling $\alpha : E \to X^{[*]}$ are defined as follows:

  E contains an edge $e : (q, y) \xrightarrow{a} (q', y') \iff$ one of the following cases occurs:

  (1) $y = R_2,\ y' = R_2,\ (q', R) \in \delta(q, a)$

  (2) $y = R_2,\ y' = L_1,\ (q', L) \in \delta(q, a)$

  (3) $y = R_1,\ y' = R_2,\ q = q'$ and there exists $\bar{q} \in Q$ with $(q, R) \in \delta(\bar{q}, a)$

  E contains an edge $e : (q, y) \xrightarrow{a^{-1}} (q', y') \iff$ one of the following cases occurs:

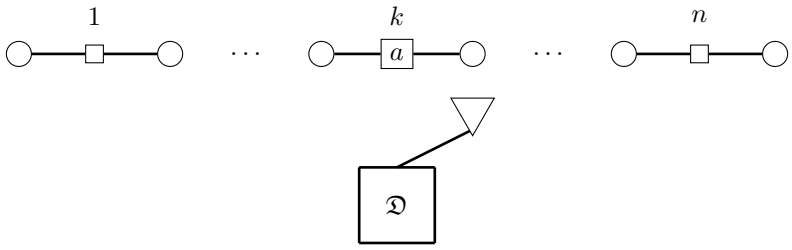
  (1) $y = L_2,\ y' = L_2,\ (q', L) \in \delta(q, a)$

  (2) $y = L_2,\ y' = R_1,\ (q', R) \in \delta(q, a)$

  (3) $y = L_1,\ y' = L_2,\ q = q'$ and there exists $\bar{q} \in Q$ with $(q, L) \in \delta(\bar{q}, a)$

To clarify this definition, we imagine the following model:

The input tape of the automaton is alternatingly divided into rectangular and oval cells.



The ovals represent "idle-positions" for the reading head, the rectangles contain the single symbols of the input word.

The reading head moves from an "idle-position" to the neighbor "idle-position" to the left or to the right.

If it moves from left to right over the cell k with symbol a, then the automaton reads a, if it moves from right to left over that cell, it reads a^{-1}. (This kind of reading when traversing a cell corresponds to the physical model of a magnetic disc).

In some cases, our model has to perform two computation steps when the automaton $\mathfrak{B}$ performs only a single step.

The following cases may occur:

(1) The automaton is in idle position to the left of a cell k. It remembers that it should move to the right, i.e. it is in some state (q, R_2). It traverses now the kth cell, reads the symbol a, reaches the subsequent idle position and finds that it should have gone to the left. This is desribed by the state (q', L_1). Therefore it moves back to its previous idle position and changes into state (q', L_2). Together it has read aa^{-1}.

This kind of movement can be expressed using the following path:

$$(q, R_2) \xrightarrow{a} (q', L_1) \xrightarrow{a^{-1}} (q', L_2)$$

(2) In the same way, the following path is to be understood:

$$(q, L_2) \xrightarrow{a^{-1}} (q', R_1) \xrightarrow{a} (q', R_2)$$

The definition of $\mathfrak{D}$ is completed by setting

$$S = S_{\mathfrak{B}} \times \{R_2\}, \quad F = F_{\mathfrak{B}} \times \{R_2\}$$

We write again $|u| \in (X \cup X^{-1})^*$ for the reduced word for an element $[u] \in X^{[*]}$ and define:

Definition 8.4.

$$\begin{aligned} <L_{\mathfrak{D}}> \quad := \quad & \{|\alpha(\pi)| \mid |\alpha(\pi)| \in X^*,\ \pi \in \mathfrak{W}(G)(S, F) \\ & \textit{and for all paths prefixes } \omega \prec \pi \textit{ holds: } |\alpha(\omega)| \prec |\alpha(\pi)|\} \end{aligned}$$

Here, the symbol $\prec$ denotes the path-prefix relation

$$\prec \subset \mathfrak{W}(G) \times \mathfrak{W}(G)$$

as well as the word-prefix relation

$$\prec \subset (X \cup X^{-1})^* \times (X \cup X^{-1})^*$$

First, we want to give an example to clarify the construction given above.

The following two-way finite automaton accepts the language

$$\begin{aligned} L \quad = \quad & \{w \in \{a, b\}^+ \cdot c \cdot \{a, b\}^+ \mid w = w_1 \cdots w_n,\ w_i \in X \\ & \text{with } w_i = c \Rightarrow w_{i-1} = w_{i+1},\ 1 < i < n\} \end{aligned}$$

The transition function shall be defined as follows:

$$\begin{aligned}
\delta(q_0, a) &= \delta(q_1, a) = (q_1, R) \\
\delta(q_0, b) &= \delta(q_1, b) = (q_1, R) \\
\delta(q_1, c) &= (q_2, L) \\
\delta(q_2, a) &= (q_a, R) \\
\delta(q_2, b) &= (q_b, R) \\
\delta(q_a, c) &= (q_{a'}, R) \\
\delta(q_b, c) &= (q_{b'}, R) \\
\delta(q_{a'}, a) &= (q_e, R) \\
\delta(q_{b'}, b) &= (q_e, R) \\
\delta(q_e, a) &= \delta(q_e, b) = q_e, R)
\end{aligned}$$

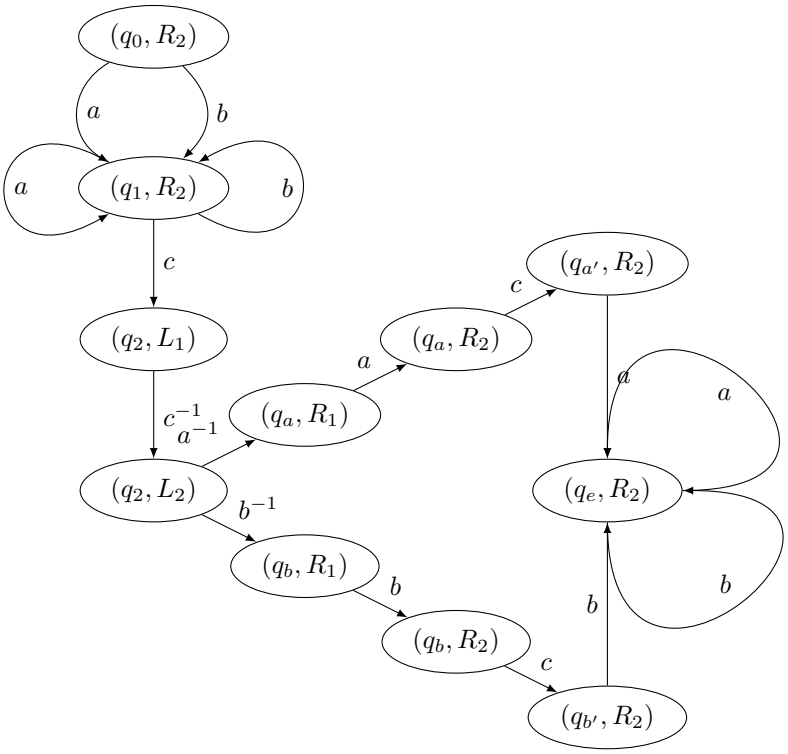
The two-way finite automaton $\mathfrak{A}$ is defined by

$$\mathfrak{A} = (\{q_0, q_1, q_2, q_a, q_b, q_{a'}, q_{b'}, q_e\}, \{a, b, c\}, \{q_0\}, \{q_e\}, \delta)$$

From this we want to construct our automaton

$$\mathfrak{D} = (G, \{a, b, c\}^{[*]}, \{(q_0, R_2)\}, \{(q_e, R_2)\}, \alpha)$$

The graph $G = (V, E)$ has the following form (the markings at the edges are the edge labels):



We begin with proving the equivalence between both automata models. To do so, we prove the following lemma.

LEMMA 8.1.

$$L_{\mathfrak{B}} \subset <L_{\mathfrak{D}}>$$

PROOF. To each computation $f_k,\ k \in \mathbb{N}$, of the automaton $\mathfrak{B}$ we assign a path $\pi \in \mathfrak{W}(G)(S,V)$. We use induction over the length of the computations.

Induction base:

To the empty computation f_0 we assign the empty path $1_{(q_0,R_2)}$ where $q_0 \in S_{\mathfrak{B}}$ is a start state of $\mathfrak{B}$.

It holds: $\alpha(1_{(q_0,R_2)}) = \epsilon = u_0$.

Induction step:

For all computations f_k of length k we assume to have a path $\pi_k \in \mathfrak{W}(G)(S,V)$ such that $Z(\pi_k) \in Q \times \{(R_2, L_2)\}$ and

$$|\alpha(\pi_k)| = \begin{cases} u_k & \text{for } Z(\pi_k) \in Q \times \{R_2\} \\ u_k \cdot First(v_k) & \text{for } Z(\pi_k) \in Q \times \{L_2\} \end{cases}$$

Here $First(w)$ denotes the first symbol of the word w.

Let f_{k+1} be a computation of length $k+1$ which ends with the computation step

$$u_k q_k v_k \to u_{k+1} q_{k+1} v_{k+1}$$

Case 1: $Z(\pi_k) = (q_k, R_2),\ v_k = a\bar{v}_k,\ (q_{k+1}, R) \in \delta(q_k, a)$

By construction, G contains an edge $e : (q_k, R_2) \xrightarrow{a} (q_{k+1}, R_2)$.

We set $\pi_{k+1} := \pi_k \cdot e$ and get

$$|\alpha(\pi_{k+1})| = |\alpha(\pi_k)| \cdot a = u_{k+1}$$

Case 2: $Z(\pi_k) = (q_k, R_2),\ u_k = \bar{u}_k b,\ v_k = a\bar{v}_k,\ (q_{k+1}, L) \in \delta(q_k, a)$

By construction, in G there exists a path

$$(q_k, R_2) \xrightarrow{e_1/a} (q_{k+1}, L_1) \xrightarrow{e_2/a^{-1}} (q_{k+1}, L_2),\ e_1, e_2 \in E$$

We set $\pi_{k+1} := \pi_k \cdot e_1 \cdot e_2$, then it is

$$|\pi_{k+1}| = |\pi_k| = u_{k+1} \cdot b = u_{k+1} \cdot First(v_{k+1})$$

and $Z(\pi_{k+1}) \in Q \times \{L_2\}$.

Case 3: $Z(\pi_k) = (q_k, L_2),\ v_k = a\bar{v}_k,\ (q_{k+1}, R) \in \delta(q_k, a)$

In G there exists a path

$$(q_k, L_2) \xrightarrow{e_1/a^{-1}} (q_{k+1}, R_1) \xrightarrow{e_2/a} (q_{k+1}, R_2),\ e_1, e_2 \in E$$

We set $\pi_{k+1} := \pi_k \cdot e_1 \cdot e_2$, then it is $Z(\pi_{k+1}) \in Q \times \{R_2\}$ and

$$|\alpha(\pi_{k+1})| = |u_k a a^{-1} a| = u_k a = u_{k+1}$$

Case 4: $Z(\pi_k) = (q_k, L_2),\ u_k = \bar{u}_k b,\ v_k = a\bar{v}_k,\ (q_{k+1}, L) \in \delta(q_k, a)$

In G there exists an edge $e : (q_k, L_2) \xrightarrow{a^{-1}} (q_{k+1}, L_2)$.

We set $\pi_{k+1} := \pi_k \cdot e$, then $Z(\pi_{k+1}) = (q_{k+1}, L_2)$ and it is

$$|\alpha(\pi_{k+1})| = |\alpha(\pi_k)| \cdot a^{-1} = |u_k \cdot a a^{-1}| = |u_k| = u_{k+1} \cdot First(v_{k+1})$$

These are all possible cases for π_{k+1} and π_{k+1} fulfills the induction claim for $k+1$.

Let now f_n be an accepting computation for a word v, so $v_n = \epsilon,\ u_n = v$ and $q_n \in F$.

The last move of $\mathfrak{B}$ was a right-move. It holds $Z(\pi_n) = (q_n, R_2)$ and therefore it holds $|\alpha(\pi_n)| = u_n = v$.

For each subpath ω of a path π_n constructed in that way we get

$$\forall \omega \prec \pi_n \text{ it holds: } |\alpha(\omega)| \prec |\alpha(\pi_n)|$$

Because f_n is an *accepting* computation, i.e. the reading head is placed to the right of the right-most input symbol, each input symbol must have been traversed one time more often to the right than to the left.

This means, if a subpath $e \cdot e'$ with label $\alpha(e \cdot e') = aa^{-1}$ exists in π_n, then it must be followed by an edge e'' with $\alpha(e'') = a$.

The prefix condition of π_n therefore cannot be violated by cancellation of an edge labelled with a^{-1}.

It therefore holds: $|\alpha(\pi_n)| = u_n = v \in <L_{\mathfrak{D}}>$. □

Lemma 8.2.

$$<L_{\mathfrak{D}}> \subset L_{\mathfrak{B}}$$

Proof. Let $G = (V, E)$ be the graph of $\mathfrak{D}$, $v \in <L_{\mathfrak{D}}>$ and $\pi \in \mathfrak{W}(G)(S, F)$ be a path with label $|\alpha(\pi)| = v$, and for any path prefix $\omega \prec \pi$ it should hold $|\alpha(\omega)| \prec |\alpha(\pi)|$.

We want to construct a computation f which corresponds to the path π in the sense of the previous lemma.

We consider the sequence of subpaths

$$1_{(q_0,R_2)} \prec \pi_1 \prec \pi_2 \ldots \prec \pi_n = \pi$$

with $\text{length}(\pi_i) - \text{length}(\pi_{i-1}) \leq 2$ and $Z(\pi_i) \in Q \times \{R_2, L_2\}$ for $i = 1, \ldots, n$ and for $\pi_i \prec \omega \prec \pi_{i+1},\ \pi_i \neq \omega \neq \pi_{i+1}, \Rightarrow Z(\omega) \in Q \times \{R_1, L_1\}$.

We apply the automaton $\mathfrak{B}$ on the input v and show that there exists a computation f which corresponds to the path π.

To the empty path $\pi_0 = 1_{(q_0,R_2)}$ corresponds the computation $q_0 v$.

Let f_k be a computation ending with $u_k q_k v_k$.

Here, we also have to consider 4 cases:

Case 1: $Z(\pi_k) = (q_k, R_2),\ \pi_{k+1} = \pi_k \cdot e$ with $Z(e) = (q_{k+1}, R_2)$ and $\alpha(e) = a$.

It then holds $v_k = a\bar{v}_k$ and by construction of $\mathfrak{D}$ it holds $(q_{k+1,R}) \in \delta(q_k, a)$, therefore $u_{k+1} q_{k+1} v_{k+1}$ is a configuration following $u_k q_k v_k$.

Case 2: $Z(\pi_k) = (q_k, R_2),\ \pi_{k+1} = \pi_k \cdot e_1 \cdot e_2$.

We then have $Z(e_1) = (q_{k+1}, L_1),\ Z(e_2) = (q_{k+1}, L_2)$ and $\alpha(e_1) = a$ and $\alpha(e_2) = b^{-1}$.

Because of $|\alpha(\pi)| \in X^*$ it follows $b = a$ and we have

$$(q_{k+1}, L) \in \delta(q_k, a) \text{ and } u_{k+1} = ua^{-1},\ v_{k+1} = av_k$$

and $u_{k+1}q_{k+1}v_{k+1}$ is a configuration following $u_kq_kv_k$.

Case 3: $Z(\pi_k) = (q_k, L_2),\ \pi_{k+1} = \pi_k \cdot e$.

It then holds $Z(e) = (q_{k+1}, L_2)$ and $\alpha(e) \in X^{-1}$. Let $\alpha(e) = a^{-1}$. We then have $|\alpha(\pi_{k+1})| = |\alpha(\pi_k) \cdot a^{-1}|$. Because of $|\alpha(\pi| \in X^*,\ \pi \in \mathfrak{W}(G)(S, V)$ and $|\alpha(\pi_k| = u_k \cdot First(v_k)$ it follows $First(v_k) = a$.

If we have $u_k = \bar{u}_k b$, then for $u_{k+1} = \bar{u}_k$ and $v_{k+1} = bav_k$ it holds: $u_{k+1}q_{k+1}v_{k+1}$ is a configuration following $u_kq_kv_k$.

Case 4: $Z(\pi_k) = (q_k, L_2),\ \pi_{k+1} = \pi_k \cdot e_1 \cdot e_2$.

It follows $Z(e_1) = (q_{k+1}, R_1)$ and $Z(e_2) = (q_{k+1}, R_2)$ and $\alpha(e_1) = a^{-1} \in X^{-1},\ \alpha(e_2) = b \in X$.

Because of $\alpha(\pi) \in X^*,\ \pi \in \mathfrak{W}(G)(S, V)$ it follows $|\alpha(\pi_k)| = u_ka$ and because $u_kaa^{-1}b \prec |\alpha(\pi)|$ and $u_ka \prec |\alpha(\pi_{k+1})|$ it follows $b = a$.

Therefore $\alpha(\pi_{k+1}) = u_ka$. If we set $u_{k+1} = u_ka$ and $v_{k+1} = a^{-1}v_k$ ($v_k = a\bar{v}_k$), then by construction of $\mathfrak{D}$, $u_{k+1}q_{k+1}v_{k+1}$ is a configuration following $u_kq_kv_k$ and the path π_{k+1} corresponds to the computation f_k.

Let $v \in <L_{\mathfrak{D}}>$ and $\pi_n \in \mathfrak{W}(G)(S, F)$ with label $|\alpha(\pi_n)| = v$ and let for $\omega \prec \pi_n$ hold $|\alpha(\omega)| \prec |\alpha(\pi_n)|$, then there exists a computation $f_n = (q_0v \to \ldots \to vq_n)$ with $q_0 \in S_{\mathfrak{B}}, q_n \in F_{\mathfrak{B}}$ and therefore $v \in L_{\mathfrak{B}}$. □

Both lemmata are summarized in the following theorem:

THEOREM 8.1.

$$L_{\mathfrak{B}} = <L_{\mathfrak{D}}>$$

We will show now that for all finite two-way automata $\mathfrak{B}$ it holds

$$L_{\mathfrak{B}} \in \mathbf{REG}(X^*)\ (= \mathbf{RAT}(X^*))$$

Before doing that, we show a relationship between $\mathbf{RAT}(X^*)$ and $\mathbf{RAT}(X^{[*]})$.

For the automaton $\mathfrak{D}$ constructed above we set

$$L_{\mathfrak{D}} := \{|\alpha(\pi)| \mid \pi \in \mathfrak{W}(G)(S, F)\}$$

where $G = (V, E)$ is the graph of $\mathfrak{D}$.

In this way, $\mathbf{RAT}(X^{[*]})$ is embedded in a natural way into $(X \cup X^{-1})^*$, see chapter II.7.

We set

$$|\mathbf{RAT}(X^{[*]})| := \{|L| \mid L \in \mathbf{RAT}(X^{[*]})\}$$

Then the following theorem holds:

THEOREM 8.2.

$$|L| \cap X^* \in \mathbf{RAT}(X^*)$$

PROOF. We only sketch the proof of this theorem, the reader may give the proof formally.

Let q_1, q_2 be vertices of the graph $G = (V, E)$ of the automaton $\mathfrak{D}$, and let $\pi : q_1 \to q_2$ be a path with label $\alpha(\pi) \equiv 1 \pmod{X^{[*]}}$.

In this case, a new edge $e : q_1 \xrightarrow{\epsilon} q_2$ will be added to the graph. This will be done for all pairs $(q_1, q_2) \in V \times V$ which are connected by such a path.

We get a new automaton $\mathfrak{D}'$ with $|L_{\mathfrak{D}}| = |L_{\mathfrak{D}'}|$.

If one removes all edges e from the graph of $\mathfrak{D}'$ with label $\alpha(e) \in X^{-1}$, one gets a finite automaton $\mathfrak{A}$ accepting the language $|L| \cap X^*$ (exercise). □

Remark: A similar result can be proved for the free group $F(X)$ (exercise).

THEOREM 8.3 (Rabin, Scott).

$$<L_{\mathfrak{D}}> \in \mathbf{REG}(X^*)$$

PROOF. The proof gets clearer if we don't use the state graph but the automaton model used in the construction of $\mathfrak{D}$.

A figure is missing here!

On the input tape we have $w = x_1 \cdots x_n,\ x_i \in X$, the automaton shall be in state $(q, y),\ y \in \{L_2, R_2\}$) and the reading head shall be located to the right of the kth cell.

We set $w_k := x_1 \cdots x_k$ and ask in which states the automaton will return to its current position for $y = L_2$.

We denote the set of these states as the **echo** of state q in w_k.

We don't assume that the automaton has read the tape from the left, but think of the head to be placed arbitrarily at this position in state (q, L_2).

We define the echo formally:

Define for $v \in X^*,\ a \in X$ the relation $\mathrm{Echo}_v \subset V \times \mathfrak{P}(V)$ by

(1) $\mathrm{Echo}_\epsilon((q, L_i)) = \emptyset$ for $i = 1, 2$.

(2) $\mathrm{Echo}_v((q, R_i)) = (q, R_i)$ for $i = 1, 2$.

(3) $\mathrm{Echo}_{v \cdot a}((q, L_1)) = a \cdot \mathrm{Echo}_v((a^{-1}(q, L_1)))$

(4) $\mathrm{Echo}_{v \cdot a}((q, L_2)) = (a \cdot \mathrm{Echo}_v(a^{-1}))^+(qL_2)$

□

CHAPTER 3

Finite Automata with Storage

1. The finite automaton with pushdown store

We investigate now a second example (after the 2-way automaton) that represents a generalization of the finite automaton in our concept, the **finite automaton with pushdown store**.

For preparation we remember theorem 2 from chapter II.8

THEOREM 1.1. *For a finite automaton* $\mathfrak{A} = (G, X^{[*]}, S, F, \alpha)$ *it holds:*

$$X^* \cap |L_{\mathfrak{A}}| \in \mathbf{REG}(X^*)$$

Remark: From this theorem the following claims can be derived (exercise):

(1) With the same assumptions it holds $|L_{\mathfrak{A}}| \in \mathbf{REG}((X \cup X^{-1})^*)$

(2) Corresponding theorems hold for the free group $F(X)$ and the polycyclic monoid $X^{(*)}$ instead of $X^{[*]}$.

We want to introduce now the finite automaton with pushdown store. We define

DEFINITION 1.1. $\mathcal{K} = (X, Y, \delta)$ *is called* **simple pushdown store** *with input* X^* *and stack-monoid* Y^* *if*

(K1) $\delta : X \times Y^* \to Y^*$ *is a mapping with*

$$\begin{aligned} \delta(X \times Y) &\subset Y^2 \cup \{\epsilon\} \\ \delta(X \times \{\epsilon\}) &\subset Y \end{aligned}$$

(K2) $\delta(x, u \cdot v) = u \cdot \delta(x, v)$ *for* $v \in Y^+, u \in Y^*$.

Obviously the simple pushdown store $\mathcal{K}$ is uniquely determined by $\delta' = \delta|_{X \times (Y \cup \{\epsilon\})}$. Each such mapping can be continued using (K2) onto $X \times Y^*$ such that one gets a simple pushdown store.

We extend δ onto $X^* \times Y^*$ by defining

$$\begin{aligned} \delta(\epsilon, u) &= u \\ \delta(xv, u) &= \delta(v, \delta(x, u)) \end{aligned}$$

The following lemma holds:

LEMMA 1.1. *For* $w \in X^{[*]}$ *there exists a pushdown store*

$$\mathcal{K} = (X \cup X^{-1}, X \cup X^{-1}, \delta)$$

with $|w| = \delta(w, \epsilon)$.

PROOF. Define

$$\begin{aligned} (1) \quad \delta(x, \epsilon) &= x, \ x \in X \cup X^{-1} \\ (2) \quad \delta(x, y) &= yx, \ x \neq y^{-1} \\ (3) \quad \delta(x^{-1}, x) &= \epsilon, \ x \in X \end{aligned}$$

Let $w \in (X \cup X^{-1})^*$ and $w = x_1^{\epsilon_1} \cdots x_n^{\epsilon_n}$ with $x_i \in X, \ \epsilon_i \in \{1, -1\}$.

For the sequence $y_1 = \delta(x_1^{\epsilon_1}, \epsilon), \ y_2 = \delta(x_2^{\epsilon_2}, y_1), \ldots, \ y_n = \delta(x_n^{\epsilon_n}, y_{n-1})$ it holds: $y_n = |w|$ (exercise). □

(The following example was not included in the original book.)

Example: Let $X = \{a, b\}$ and $w = abb^{-1}baa^{-1} \in X^{[*]}$. It is easy to see that the reduced word $|w| = ab$. The computation of $\delta(w, \epsilon)$ is

$$\begin{aligned}
\delta(abb^{-1}baa^{-1}, \epsilon) &= \delta(bb^{-1}baa^{-1}, \delta(a, \epsilon)) \\
&= \delta(bb^{-1}baa^{-1}, a) \\
&= \delta(b^{-1}baa^{-1}, \delta(b, a)) \\
&= \delta(b^{-1}baa^{-1}, ab) \\
&= \delta(baa^{-1}, \delta(b^{-1}, ab)) \\
&= \delta(baa^{-1}, a\delta(b^{-1}, b)) \\
&= \delta(baa^{-1}, a\epsilon) \\
&= \delta(baa^{-1}, a) \\
&= \delta(aa^{-1}, \delta(b, a)) \\
&= \delta(aa^{-1}, ab) \\
&= \delta(a^{-1}, \delta(a, ab)) \\
&= \delta(a^{-1}, a\delta(a, b)) \\
&= \delta(a^{-1}, aba) \\
&= \delta(\epsilon, \delta(a^{-1}, aba)) \\
&= \delta(\epsilon, ab\delta(a^{-1}, a)) \\
&= \delta(\epsilon, ab\epsilon) \\
&= \delta(\epsilon, ab) \\
&= ab
\end{aligned}$$

Remark:

(1) The lemma holds also for the free group $F(X)$.

(2) The lemma does not hold for the polycyclic monoid $X^{(*)}$ (exercise).

The reader should extend the definition of the pushdown store to a pushdown store with 0 such that the lemma holds also for the polycyclic monoid $X^{(*)}$.

We can now define the finite automaton with pushdown store.

DEFINITION 1.2 (finite automaton with pushdown store, PDA). *$\kappa = (\mathfrak{A}, \mathcal{K})$ is called* **finite automaton with pushdown store** *if $\mathfrak{A} = (G, X, S, F, \alpha)$ is a finite automaton with graph $G = (V, E)$ and $\mathcal{K} = (E, Y, \delta)$ is a simple pushdown store.*

Two sets are assigned to the PDA κ, the **pushdown store language** $PSL(\kappa)$ which is the set of possible words on the pushdown store when $\mathfrak{A}$ accepts a word, and the **accepted language** L_κ which is the set of input words accepted with empty pushdown store.

Formally:

$$\begin{aligned}
PSL(\kappa) &:= \{\delta(\pi, \epsilon) \mid \pi \in \mathfrak{W}(G)(S, F)\} \\
L_\kappa &:= \{\alpha(\pi) \mid \pi \in \mathfrak{W}(G)(S, F),\ \delta(\pi, \epsilon) = \epsilon\}
\end{aligned}$$

(In the original book the pushdown store language $PSL(\kappa)$ was named "Kellerklasse" (cellar class) and denoted by $KKL(\kappa)$.)

DEFINITION 1.3.

$$\mathbf{ALG}(\mathbf{X}_\infty^*) := \{L \subset X_\infty^* \mid \textit{there exists a PDA } \kappa \textit{ with } L = L_\kappa\}$$

is called the class of **algebraic languages**.

The pushdown store language $PSL(\kappa)$ is an example for our concept of generalizing finite automata by changing the monoid.

Let $\mathcal{K} = (X, Y, \delta)$ be a pushdown store. If to each $x \in X$ we assign a mapping $\delta_x : Y^* \to Y^*,\ \delta_x(u) := \delta(x, u)$, then we can represent a PDA κ by the finite automaton

$$\mathfrak{B} = (G, M, S, F, \beta), \quad G = (V, E)$$

with monoid $M = (\{\delta_e \mid e \in E\}, \circ)$ and labelling $\beta(e) = \delta_e$.

It holds:

$$L_{\mathfrak{B}}(\epsilon) := \{\beta(\pi)(\epsilon) \mid \pi \in \mathfrak{W}(G)(S, F)\} = PSL(\kappa)$$

As our main result we get

THEOREM 1.2 (Pushdown store languages are regular).

$$PSL(\kappa) \in \mathbf{REG}(Y^*)$$

Before starting with the formal proof we want to explain the construction of the automaton for $PSL(\kappa)$ using an example.

Let $\pi = (e_1e_2e_3e_4e_5e_6e_7e_8)$ a path from $\mathfrak{W}(G)$ and the pushdown store be empty initially.

δ shall be defined as follows:

$$\begin{array}{lll} \delta(s_1, \epsilon) = y_1 & \delta(s_2, y_1) = y_1y_2 & \delta(s_3, y_2) = y_2y_3 \\ \delta(s_4, y_3) = \epsilon & \delta(s_5, y_2) = y_2y_4 & \delta(s_6, y_4) = \epsilon \\ \delta(s_7, y_2) = \epsilon & \delta(s_8, y_1) = y_1y_5 & \end{array}$$

We assign to the computation of the pushdown store a word $w' \in (Y \cup Y^{-1})^*$. The reduced word is the current content of the pushdown store, hereby we have to guess in w' the last symbol of the reduced word (corresponds to the symbol on top of the stack).

Let a path $\pi \in \mathfrak{W}(G),\ G = (V, E)$, be of the form

$$\xrightarrow{e_1}\xrightarrow{e_2}\xrightarrow{e_3}\xrightarrow{e_4}\xrightarrow{e_5}\xrightarrow{e_6}\xrightarrow{e_7}\xrightarrow{e_8}$$

In the automaton to be constructed we will have the following marking

$$\xrightarrow{y_1}\xrightarrow{y_1^{-1}y_1y_2}\xrightarrow{y_2^{-1}y_2y_3}\xrightarrow{y_3^{-1}y_2^{-1}y_2}\xrightarrow{y_2^{-1}y_2y_4}\xrightarrow{y_4^{-1}y_2^{-1}y_2}\xrightarrow{y_2^{-1}y_1^{-1}y_1}\xrightarrow{y_1^{-1}y_1y_5}$$

The first inverse on each edge label is used to "guess" the top symbol on the stack.

The corresponding stack contents are:

$$y_1 \quad y_1y_2 \quad y_1y_2y_3 \quad y_1y_2 \quad y_1y_2y_4 \quad y_1y_2 \quad y_1 \quad y_1y_5$$

By multiplying the edge labels and applying the cancellation rules one gets the stack contents.

We want to execute this idea formally:

For the proof we construct an automaton $\mathfrak{C}$ over the H-group $Y_0^{[*]}$ where $Y_0 := Y \cup \{y_0\},\ y_0 \notin Y$ and the property

$$Y_0^* \cap |L_{\mathfrak{C}}| = y_0 \cdot PSL(\kappa)$$

From theorem 1 then follows that $y_0 \cdot PSL(\kappa)$ is regular and from this follows immediately $PSL(\kappa) \in \mathbf{REG}(Y^*)$.

(The readability of the proof in the book was improved by using other names and by adding comments and figures.)

PROOF. Let $\kappa = (\mathfrak{A}, \mathcal{K})$ be a PDA with finite automaton

$$\mathfrak{A} = (G, X, S, F, \alpha)$$

with graph $G = (V, E)$ and pushdown store $\mathcal{K}$.

We define a finite automaton $\mathfrak{C}$ with the H-group over Y_0 as its storage monoid:

$$\mathfrak{C} = (G', Y_0^{[*]}, \{s_0\}, F, \gamma)$$

The graph $G' = (V', E')$ of $\mathfrak{C}$ is defined as follows:

The vertices are the same as in G plus a new vertex s_0 which becomes the start state of $\mathfrak{C}$:

$$V' = V \cup \{s_0\},\ s_0 \notin V$$

The edge set E' contains three different types of edges:

$$E' = \underbrace{\{s_0\} \times S}_{\text{start edges}} \quad \cup \quad \underbrace{E \times Y_0}_{\text{push-edges}} \quad \cup \quad \underbrace{E \times Y \times Y_0}_{\text{pop-edges}}$$

Start edges: These edges provide the initial stack content (bottom-of-stack symbol y_0).

For each start state $s \in S$ of the PDA κ we define an edge

$$e' : s_0 \xrightarrow{y_0} s$$

formally:

$$Q(e') = s_0, \quad Z(e') = s, \quad \gamma(e') = y_0$$

Edges for simulating push-operations: These edges simulate a push-operation on the stack. The top-most stack symbol y is "guessed" and if the guess was correct, y is restored and z is pushed on the stack. A wrong guess results in a combination of symbols on the stack that cannot be reduced to the empty word (no accepting computation possible) anymore.

For each pair $(e, y) \in E \times Y$ there is an edge

$$e' : Q(e) \xrightarrow{y^{-1}yz} Z(e)$$

but only if the pushdown store does not compute the monoid unit for the edge, formally $|\delta(e, y)| \neq \epsilon$. That means, the pushdown store indeed puts a symbol on the stack when executing the transition e.

Edges for handling empty stack: These edges simulate computations with empty stack. The bottom-of-stack symbol is guessed and if the guess was correct, the corresponding stack operation is simulated.

For each edge $e \in E$ we define an edge

$$e' : Q(e) \xrightarrow{y_0^{-1} y_0 \delta(e,\epsilon)} Z(e)$$

Edges for simulating pop-operations: These edges simulate popping of symbols from the stack. The top-most stack symbol y is guessed and if the guess was correct, each possible symbol z that could be on the stack after popping y is provided as a possible continuation. If some symbol can never be on the stack after popping y the computation can never be successful.

For each triple $(e, y, z) \in E \times Y \times Y_0$ we define an edge

$$e' : Q(e) \xrightarrow{y^{-1} z^{-1} z} Z(e)$$

but only if the pushdown store pops the symbol y when executing the transition e, formally $\delta(e, y) = \epsilon$.

This completes the definition of the finite automaton $\mathfrak{C}$.

We prove first that each accepting computation of the pushdown store may be simulated with the finite automaton $\mathfrak{C}$:

$$y_0 \cdot PSL(\kappa) \subset |L_{\mathfrak{C}}|$$

To prove this, we construct for each accepting path $\pi \in \mathfrak{W}(G)(S, F)$ an accepting path $\pi' \in \mathfrak{W}(G')(\{s_0\}, F)$ with $|y_0 \cdot \delta(\pi, \epsilon)| = |\gamma(\pi')|$ which means that the computation of the pushdown store is equivalent modulo the H-group to the label of the computation path in $\mathfrak{C}$.

Given an accepting computation π in the PDA, we consider all prefixes ψ of π by increasing length. The corresponding prefix of path π' in graph G' will be denoted by ψ'.

Induction base: For prefix ψ of length 0, that is $\psi = (Q(\pi), Q(\pi))$, we set $\psi' := s_0 \xrightarrow{y_0} Q(\pi)$, that is an edge from the start state s_0 to the start vertex of π.

We get $\delta(\psi, \epsilon) = \delta(\epsilon, \epsilon) = \epsilon$ and $\gamma(\psi') = y_0$ which proves the claim for paths of zero length.

Induction step: We assume that the two prefixes ψ and ψ' have the same target vertex, $Z(\psi) = Z(\psi')$, and that the computations for these prefixes fulfill the induction assumption, namely $|\gamma(\psi')| = |y_0 \cdot \delta(\psi, \epsilon)|$.

Consider the path $\pi = \psi \circ e$ where e is an edge of G and let $|y_0 \cdot \delta(\psi, \epsilon)| = u \cdot y$ for some symbol $y \in Y_0$.

We have two cases.

Case 1: $y \in Y$ and $\delta(e, y) = yz$:

By construction of G' there exists an edge $e' : Q(e) \xrightarrow{y^{-1} yz} Z(e) \in E \times Y$.

Therefore we have $|y_0 \cdot \delta(\psi \circ e, \epsilon)| = |y_0 \cdot u \cdot y \cdot z = |\gamma(\psi') \cdot e'|$.

The case $y = y_0$ and $\delta(e, \epsilon) = y$ can be handled similarly.

Case 2: $y \in Y$ and $\delta(e, y) = |yy^{-1}| = \epsilon$:

In this case it is $u = u'z \neq \epsilon$ and we can write $|y_0 \cdot \delta(\psi, \epsilon)| = u'zy$.

In E' there exists an edge $e' : Q(e) \xrightarrow{y^{-1}z^{-1}z} Z(e) \in E \times Y \times Y_0$.

We define $\psi' \circ e'$ as the path corresponding to $\psi \circ e$ and get $|y_0 \cdot \delta(\psi \circ e, \epsilon)| = |\gamma(\psi' \circ e')| = u$.

This proves inductively the inclusion $y_0 \cdot PSL(\kappa) \subset |L_{\mathfrak{C}}|$.

We prove now:

$$y_0 \cdot Y^* \cap |L_{\mathfrak{C}}| \subset y_0 \cdot PSL(\kappa)$$

Let $\pi' \in \mathfrak{W}(G')(\{s_0\}, F)$ and $|\gamma(\pi')| \in y_0 \cdot Y^*$.

From the definition of γ on E' it follows that also for each path prefix ψ' of π it holds: $|\gamma(\psi')| \in y_0 \cdot Y^*$.

We construct for each path ψ' a path ψ with $|\gamma(\psi)| = y_0 \cdot \delta(\pi, \epsilon) \in \mathfrak{W}(G)$.

To a path of length 1 we assign the empty path $1_{Z(\psi')}$.

For a path $\psi' = e_1 \circ \psi''$ where $e_1 \in E'$ we assign to ψ'' the projection π'' of ψ'' onto $\mathfrak{W}(G)$. This projection is a functor that maps paths into paths.

For ψ' of length 1, our claim $|\gamma(\psi')| = y_0 \cdot \delta(\pi, \epsilon)$ is obviously true.

Assume the claim to be true for ψ' and $\psi' \circ e'$ to be a subpath of π'.

Case 1:

$\gamma(e') = y^{-1}yz$ so $e' = (e, y) \in E'$. From $|\gamma(\psi' \cdot e')| \in y_0 \cdot Y^*$ it follows $|\gamma(\psi')| = u \cdot y$, so we have $\delta(\psi' \circ e')| = y_0 \cdot |\delta(\psi \circ e, \epsilon)|$.

Case 2:

$\gamma(e') = y^{-1}z^{-1}z$, so $e' = (e, y, z) \in E'$.

As before it follows $\gamma(\psi')| = uyz = \delta(\pi, \epsilon)$. Therefore it holds also in this case $|\gamma(\psi' \circ e')| = y_0 \cdot |\delta(\psi \circ e, \epsilon)|$.

By induction follows the inclusion $y_0 \cdot Y^* \cap |L_{\mathfrak{C}}| \subset y_0 \cdot PSL(\kappa)$.

Both inclusions give the proof of our theorem. □

Exercise:

Let $L = \{w \cdot w^R \mid w \in X^*\}$. Define a PDA accepting L and prove its correctness.

2. Closure properties of $\mathbf{ALG}(\mathbf{X}^*)$

In the last section we defined $\mathbf{ALG}(\mathbf{X}^*)$ as the class of languages over an alphabet X accepted by a pushdown automaton.

The name "algebraic languages" remembers the fact that these languages can be defined as the solution of equation systems with non-commutative variables using formal power series. See for example [**SS78**] or [**Ber77**].

We will prove for $\mathbf{ALG}(\mathbf{X}^*)$ similar closure properties as for $\mathbf{REG}(X^*)$.

In this section we assume without loss of generality that the pushdown automata have the following properties:

$\mathrm{card}(S) = \mathrm{card}(F) = 1$ and no edge ends in a start state and no edge leaves a final state. Additionally, all edges labels have length 0 or 1, $\alpha(e) \leq 1,\ e \in E$.

In a very easy way one gets

THEOREM 2.1 (Closure under monoid homomorphism). *Let $\phi : X^* \to Y^*$ be a monoid homomorphism.*

$$L \in \mathbf{ALG}(\mathbf{X}^*) \Rightarrow \phi(\mathbf{L}) \in \mathbf{ALG}(\mathbf{Y}^*)$$

PROOF. $L \in \mathbf{ALG}(\mathbf{X}^*) \Rightarrow$ there exists a PDA $\kappa = (\mathfrak{A}, \mathcal{K})$ with $L_\kappa = L$. Replace α by $\alpha' := \alpha \circ \phi$ in the definition of $\mathfrak{A}$.

For

$$\mathfrak{A}' = (G, Y, S, F, \alpha')$$

the PDA $\kappa' = (\mathfrak{A}', \mathcal{K})$ accepts $\phi(L_\kappa) = \phi(L)$. □

THEOREM 2.2. $\mathbf{ALG}(\mathbf{X}^*)$ *is closed under union, concatenation and Kleene-star.*

Closure under union:

$$L_1, L_2 \in \mathbf{ALG}(\mathbf{X}^*) \Rightarrow \mathbf{L_1} \cup \mathbf{L_2} \in \mathbf{ALG}(\mathbf{X}^*)$$

PROOF. Let

$$\kappa_1 = (\mathfrak{A}_1, \mathcal{K}_1), \quad \mathfrak{A}_1 = (G_1, X, \{s_1\}, \{f_1\}, \alpha_1), \quad G_1 = (V_1, E_1), \quad \mathcal{K}_1 = (X, Y_1, \delta_1)$$

and

$$\kappa_2 = (\mathfrak{A}_2, \mathcal{K}_2), \quad \mathfrak{A}_2 = (G_2, X, \{s_2\}, \{f_2\}, \alpha_2), \quad G_2 = (V_2, E_2), \quad \mathcal{K}_2 = (X, Y_2, \delta_2)$$

be pushdown automata accepting L_1 resp. L_2.

We may assume that $Y_1 \cap Y_2 = \emptyset,\ E_1 \cap E_2 = \emptyset,\ V_1 \cap V_2 = \{s_1, f_2\}$ where $s_1 = s_2$ and $f_1 = f_2$.

Define the PDA $\kappa_1 \cup \kappa_2$ by

$$\mathfrak{A}_1 \cup \mathfrak{A}_2 := (G_1 \cup G_2, X, \{s_1\}, \{f_2\}, \alpha_1 \cup \alpha_2)$$

and pushdown store

$$\mathcal{K}_1 \cup \mathcal{K}_2 := (E_1 \cup E_2, Y_1 \cup Y_2 \cup \{\infty\}, \delta_1 \cup \delta_2)$$

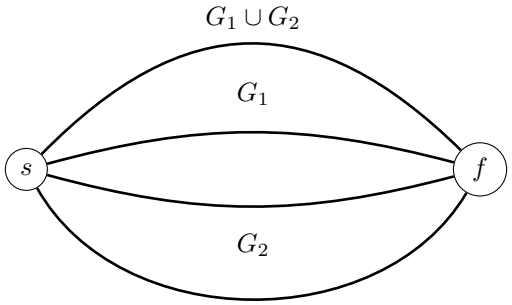
where

$$(\delta_1 \cup \delta_2)(e, y) = \begin{cases} \delta_1(e, y) & e \in E_1,\ y \in Y_1 \cup \{\epsilon\} \\ \delta_2(e, y) & e \in E_2,\ y \in Y_2 \cup \{\epsilon\} \\ \infty\infty & \text{else} \end{cases}$$

The symbol ∞ is only introduced to make $\delta_1 \cup \delta_2$ complete. If ∞ is on-top of the pushdown store, then this cannot become empty anymore. In that way it will become impossible to "mix" words from both languages.

It holds: $L_{\kappa_1 \cup \kappa_2} = L_{\kappa_1} \cup L_{\kappa_2}$ (exercise).

The following figure shows our construction:



□

Closure under complex product:

$$L_1, L_2 \in \mathbf{ALG}(\mathbf{X}^*) \Rightarrow \mathbf{L_1} \cdot \mathbf{L_2} \in \mathbf{ALG}(\mathbf{X}^*)$$

PROOF. We construct the PDA $\kappa = (\mathfrak{A}, \mathcal{K})$ with

$$\mathfrak{A} = (G, X, \{s_1\}, \{f_2\}, \alpha)$$

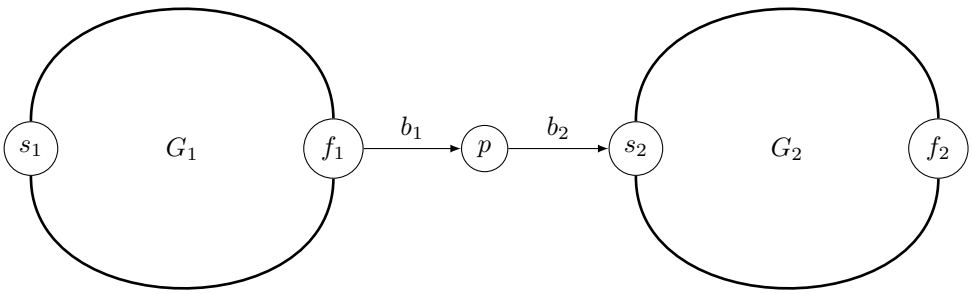
where

$$\begin{aligned} G &= (V, E) \\ V &= V_1 \cup V_2 \cup \{p\},\ p \notin V_1 \cup V_2 \\ E &= E_1 \cup E_2 \cup \{b_1, b_2\}, b_1, b_2 \notin E_1 \cup E_2 \end{aligned}$$

For the new "bridge" edges b_1, b_2 it holds $b_1 : f_1 \xrightarrow{\epsilon} p$ and $b_2 : p \xrightarrow{\epsilon} s_2$.

The labelling of the graph G is defined by

$$\alpha(e) = \begin{cases} \alpha_1(e) & e \in E_1 \\ \alpha_2(e) & e \in E_2 \\ \epsilon & e \in \{b_1, b_2\} \end{cases}$$



The pushdown store $\mathcal{K} = (E, Y, \delta)$ is defined by

$$Y = Y_1 \cup Y_2 \cup \{\bowtie, \infty\} \text{ with } \{\bowtie, \infty\} \cap (Y_1 \cup Y_2) = \emptyset$$

$$\delta(e,y) = \begin{cases} \delta_1(e,y) & e \in E_1,\ y \in Y_1 \cup \{\epsilon\} \\ \delta_2(e,y) & e \in E_2,\ y \in Y_2 \cup \{\epsilon\} \\ y \cdot \bowtie & e = b_1,\ y \in Y \cup \{\epsilon\} \\ \epsilon & e = b_2,\ y = \bowtie \\ \infty\infty & \text{else} \end{cases}$$

We prove now that the pushdown automaton κ defined in that way accepts the complex product $L_1 \cdot L_2$.

"$L_1 \cdot L_2 \subset L_\kappa$":

Let $w \in L_1 \cdot L_2$. Then there exists a factorization $w = w_1 \cdot w_2,\ w_1 \in L_1, w_2 \in L_2$.

For w_1 there exists an accepting path $\pi_1 \in \mathfrak{W}(G_1)(s_1, f_1)$ with label $\alpha_1(\pi_1) = w_1$ and pushdown store computation $\delta_1(w_1, \epsilon) = \epsilon$.

For w_2 there exists an accepting path $\pi_2 \in \mathfrak{W}(G_2)(s_2, f_2)$ with label $\alpha_2(\pi_2) = w_2$ and pushdown store computation $\delta_2(w_2, \epsilon) = \epsilon$.

Consider the path $\pi = \pi_1 \circ b_1 \circ b_2 \circ \pi_2 \in \mathfrak{W}(G)(s_1, f_2)$.

It holds

$$\begin{aligned} \delta(\pi) &= \delta(\pi_1 \circ b_1 \circ b_2 \circ \pi_2) \\ &= \delta(b_1 \circ b_2 \circ \pi_2, \delta(\pi, \epsilon)) \\ &= \delta(b_1 \circ b_2 \circ \pi_2, \epsilon) \\ &= \delta(b_2 \circ \pi_2, \delta(b_1, \epsilon)) \\ &= \delta(b_2 \circ \pi_2, \epsilon \cdot \bowtie) \\ &= \delta(\pi_2, \delta(b_2, \bowtie)) \\ &= \delta(\pi_2, \epsilon) \\ &= \delta_2(\pi_2, \epsilon) \\ &= \epsilon \end{aligned}$$

This means, the pushdown store computation for π is accepting.

The label of path π is

$$\begin{aligned} \alpha(\pi) &= \alpha(\pi_1 \circ b_1 \circ b_2 \circ \pi_2) \\ &= \alpha(\pi_1) \circ \alpha(b_1) \circ \alpha(b_1 \circ \alpha(\pi_2) \\ &= w_1 \cdot \epsilon \cdot \epsilon \cdot w_2 \\ &= w_1 \cdot w_2 \\ &= w \end{aligned}$$

This proves $w \in L_\kappa$ and therefore $L_1 \cdot L_2 \subset L_\kappa$.

"$L_1 \cdot L_2 \supset L_\kappa$":

Let $w \in L_\kappa$ be a word accepted by the PDA κ. Then there exists a path $\pi \in \mathfrak{W}(G)(s_1, f_2)$ with label $\alpha(\pi) = w$ and a pushdown store computation $\delta(\pi, \epsilon) = \epsilon$.

The path π has a unique factorization $\pi = \pi_1 \circ b_1 \circ b_2 \circ \pi_2$ with $\pi_1 \in \mathfrak{W}(G_1)(s_1, f_1)$ and $\pi_2 \in \mathfrak{W}(s_2, f_2)$.

Consider $\delta(\pi_1, \epsilon)$, the computation of the pushdown store on the path prefix π_1:

Assume $\delta(\pi_1, \epsilon) \neq \epsilon$. Then $\delta(\pi_1 \circ b_1 \circ b_2 \circ \pi_2, \epsilon) = \delta_1(\pi, \epsilon)$

TODO clarify

For the path labels we have $\alpha_1(\pi_1) = w_1 \in L_1, \alpha_2(\pi_2) = w_2 \in L_2$ and

$$\begin{aligned} \alpha(\pi) &= \alpha(\pi_1 \cdot b_1 \cdot b_2 \cdot \alpha_2) \\ &= \alpha(\pi_1) \cdot \epsilon \cdot \epsilon \cdot \alpha(\pi_2) \\ &= w_1 \cdot w_2 \\ &= w \end{aligned}$$

So $w \in L_1 \cdot L_2$ and $L_1 \cdot L_2 \supset L_\kappa$. □

Closure under Kleene-star:

$$L \in \mathbf{ALG}(\mathbf{X}^*) \Rightarrow \mathbf{L}^* \in \mathbf{ALG}(\mathbf{X}^*)$$

Proof. Let $\kappa = (\mathfrak{A}, \mathcal{K})$ be a PDA accepting L. We construct a PDA κ' accepting L^* as follows:

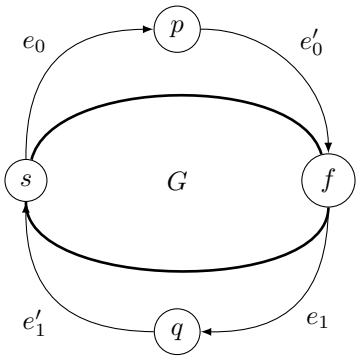
$$\mathfrak{A}' = (G, X, \{s\}, \{f\}, \alpha')$$

with

$$\begin{aligned} G' &= (V', E') \\ V' &= V \cup \{p, q\},\ \{p, q\} \cap E = \emptyset \\ E' &= E \cup \{e_0, e_0', e_1, e_1'\},\ \{e_0, e_0', e_1, e_1'\} \cap E = \emptyset \end{aligned}$$

The new edges have the following source, target states and labelling:

$$\begin{aligned} e_0 &: s \xrightarrow{\epsilon} p \\ e_0' &: p \xrightarrow{\epsilon} f \\ e_1 &: f \xrightarrow{\epsilon} q \\ e_1' &: q \xrightarrow{\epsilon} s \end{aligned}$$



The pushdown store $\mathcal{K}' = (E', Y', \delta')$ is defined by $Y' = Y \cup \{y', y'', \infty\}$ where $\{y', y'', \infty\} \cap Y = \emptyset$ and δ' defined by

$$\delta'(e,y) = \begin{cases} \delta(e,y) & e \in E,\ y \in Y \cup \{\epsilon\} \\ y' & e = e_0,\ y = \epsilon \\ \epsilon & e = e_0',\ y = y' \\ y'' & e = e_1,\ y = \epsilon \\ \epsilon & e = e_1',\ y = y'' \\ \infty\infty & \text{else} \end{cases}$$

For the so defined PDA κ' we show $L^* = L_{\kappa'}$.

"$L^* \subset L_\kappa$":

Let $w \in L^*$, then we have two cases:

(1) $w = \epsilon$: Then there exists the path $\pi = e_0 \circ e_0' \in \mathfrak{W}(G')(s,f)$ with label $\alpha(\pi) = \epsilon$ and pushdown computation $\delta'(\pi) = \epsilon$.

(2) $w \neq \epsilon$: Then there exists a factorization $w = w_1 \cdots w_k,\ w_i \in L$ and paths $\pi \in \mathfrak{W}(G)(s,f)$ with labels $\alpha(\pi_i) = w_i$ and pushdown computations $\delta(\pi, \epsilon) = \epsilon,\ i = 1, \ldots, k$.

Consider the path $\pi = \pi_1 \circ e_1 \circ e_1' \circ \pi_2 \circ e_1 \circ e_1' \ldots \circ e_1 \circ e_1' \circ \pi_k \in \mathfrak{W}(G')(s,f)$.

It holds

$$\begin{aligned} \alpha(\pi) &= \alpha(\pi_1) \cdot \alpha(e_1 \circ e_1') \cdot \alpha(\pi_2) \cdots \alpha(e_1 \circ e_1') \cdot \alpha(\pi_k) \\ &= \alpha(\pi_1) \cdot \epsilon \cdot \alpha(\pi_2) \cdots \epsilon \cdot \alpha(\pi_k) \\ &= w_1 \cdots w_k = w \end{aligned}$$

For the pushdown computation holds:

$$\begin{aligned} \delta'(\pi, \epsilon) &= \delta'(e_1 \circ e_1' \circ \pi_2 \ldots \pi_k, \delta(\pi_1, \epsilon)) \\ &= \delta'(e_1' \circ \pi_2 \ldots \pi_k, y'') = \delta'(\pi_2 \ldots \pi_k, \epsilon) \\ &\cdots \\ &= \delta'(\pi_k, \epsilon) = \delta(\pi_k, \epsilon) = \epsilon \end{aligned}$$

From this follows $w \in L_{\kappa'}$.

"$L^* \supset L_\kappa$":

Let $w \in L_{\kappa'}$ be a word accepted by the PDA κ'. Then there exists a path $\pi \in \mathfrak{W}(G')(s,f)$ with label $\alpha'(\pi) = w$ and pushdown computation $\delta'(\pi, \epsilon) = \epsilon$.

(1) $\pi = e_0 \circ e_0',\ \alpha'(\pi) = \epsilon,\ \delta'(\pi, \epsilon) = \epsilon \Rightarrow \epsilon \in L^*$.

(2) Given a factorization $\pi = \pi_1 \circ e_1 \circ e_1' \circ \pi_2 \circ e_1 \circ e_1' \ldots \circ e_1 \circ e_1' \circ \pi_k$ with $\pi_i \in \mathfrak{W}(G)(s,f)$ and edges e_1, e_1' as constructed.

Claim: $\delta'(\pi, \epsilon) = \epsilon \Rightarrow \delta(\pi_1, \epsilon) = \epsilon$

Assumption: $\delta(\pi, \epsilon) \neq \epsilon$. Then by definition of δ' it holds $\delta'(e_1, \delta(\pi_1, \epsilon)) = \infty\infty$ and because $\delta'(e, \infty) = \infty\infty$ the pushdown store will never become empty, therefore $\delta'(\pi, \epsilon) \neq \epsilon$.

This holds inductively for the paths $\pi_i,\ i = 1, \dots k$. We get

$$\begin{aligned} \delta'(\pi, \epsilon) &= \delta'(\pi_1 \circ e_1 \circ e_1' \circ \pi_2 \dots \pi_k, \epsilon) \Rightarrow \\ \delta'(\pi_1, \epsilon) &= \epsilon \\ \delta'(\pi_1 \circ e_1 \circ e_1', \epsilon) &= \delta'(e_1 \circ e_1', \delta(\pi_1, \epsilon)) = \delta'(e_1', y'') = \epsilon \end{aligned}$$

and for all $i = 1, \dots, k:\ \delta'(\pi_i, \epsilon) = \epsilon$.

It holds $\alpha(\pi_i) = w_i \in L,\ i = 1, \dots, k$.

$\alpha'(\pi_1 \circ e_1 \circ e_1' \circ \pi_2 \dots \pi_k) = \alpha(\pi_1) \cdot \epsilon \dots \epsilon \cdot \alpha(\pi_k) = w_1 \cdots w_k = w \in L^*$.

This completes the proof of our theorem. □

THEOREM 2.3 (Closure under intersection with regular language).

$$L \in \mathbf{ALG}(\mathbf{X}^*),\ \mathbf{R} \in \mathbf{REG}(\mathbf{X}^*) \Rightarrow \mathbf{L} \cap \mathbf{R} \in \mathbf{ALG}(\mathbf{X}^*)$$

PROOF. Let $\kappa = (\mathfrak{A}, \mathcal{K})$ be a PDA with $L_\kappa = L$ and $\mathfrak{B}$ a complete, deterministic finite automaton accepting R. At each vertex, a loop labeled with ϵ should be attached.

We construct a finite automaton $\mathfrak{C} = \mathfrak{A} \times \mathfrak{B}$ as we did for the intersection of regular languages in chapter II.1. The graph of $\mathfrak{C}$ is $G = (V, E)$ where the vertex set V is the cartesian product $V_\mathfrak{A} \times V_\mathfrak{B}$ and the edges are defined by

$$E = \{(e, e') \in E_\mathfrak{A} \times E_\mathfrak{B} \mid \alpha(e) = \beta(e')\}$$

Because $\mathfrak{B}$ is complete and deterministic, it holds $L_\mathfrak{C} = L_\mathfrak{A} \cap L_\mathfrak{B}$.

Now let $\kappa' = (\mathfrak{C}, \mathcal{K}')$ be a PDA with $\mathcal{K}' = (E, Y, \delta')$ and $\delta'((e, e'), y) = \delta(e, y)$. For a word $w \in L_{\kappa'}$ there exists a path $\pi \in \mathfrak{W}(G)(S, F)$ with label $\alpha(\pi) = w$ and pushdown store computation $\delta'(\pi, \epsilon) = \delta(proj_1(\pi), \epsilon) = \epsilon$. Here $proj_1(\pi)$ denotes the projection onto the first component of the edge-pairs constituting the path π.

From this follows $w \in L_\kappa \cap L_\mathfrak{B}$.

The opposite direction is clear because of the construction of PDA κ'. □

THEOREM 2.4 (Closure under inverse homorphism). *Let $\phi : X^* \to Y^*$ be a monoid homomorphism and $\phi(x) \in Y \cup \{\epsilon\}$. Then*

$$L \in \mathbf{ALG}(\mathbf{Y}^*) \Rightarrow \phi^{-1}(\mathbf{L}) \in \mathbf{ALG}(\mathbf{X}^*)$$

PROOF. Let $\kappa = (\mathfrak{A}, \mathcal{K})$ be a PDA accepting L. For the language

$$\phi^{-1}(L) = \{w \in X^* \mid \phi(w) \in L\}$$

we construct a PDA $\kappa' = (\mathfrak{A}', \mathcal{K}')$ as follows (for simplicity, we construct the graph and the pushdown store in parallel):

Let $\mathfrak{A}$ be the finite automaton

$$\mathfrak{A} = (G, Y, S, F, \alpha), \quad G = (V, E)$$

Define

$$\mathfrak{A}' = (G', X, S, F, \alpha')$$

with graph $G' = (V', E')$.

1. Vertex set V': In addition to the vertices of V, new vertices $r(v,x),\ x \in X$ are added which will be used to create cycles simulating the erasure of symbols from X by ϕ.

$$V' = V \cup \{r(v,x) \mid v \in V\}$$

2. Edge set E': For each symbol $x \in X$ which is mapped by ϕ to a symbol $y \in Y$, an edge is added to simulate the computation on y. For each symbol $x \in X$ which is erased by ϕ, two edges are added that form a cycle simulating the computation on the empty word. Formally:

(1) Let $X[\to Y] := \{x \in X \mid \phi(x) \in Y\}$. For each edge $e \in E$ labelled $y \in Y$ and each $x \in X[\to Y]$ an edge $e' : Q(e) \xrightarrow{x} Z(e)$ is added to E'.

(2) Let $X[\to \epsilon] := \{x \in X \mid \phi(x) = \epsilon\}$. For each vertex $v \in V$ and each $x \in X[\to \epsilon]$ two edges e' and e'' are added where

$$e' : v \xrightarrow{x} r(v,x), \qquad e' : r(v,x) \xrightarrow{\epsilon} v$$

The pushdown store computation for e', e'' is defined by

$$\delta'(e', y) = y\$, \qquad \delta'(e'', \$) = \epsilon$$

This means, when traversing edge e', the symbol \$ is pushed onto the stack and popped again when e'' is traversed.

The PDA $\kappa' = (\mathfrak{A}', \mathcal{K}')$ with pushdown store $\mathcal{K}' = (E', Y \cup \{\$\}, \delta')$ with δ' defined as above accepts the language $\phi^{-1}(L)$ (exercise). □

Remark: The theorem also holds for arbitrary homomorphisms ($\phi(x) \in Y^*$).

Corollary 2.1. *Let $\tau : X^* \to Y^*$ be a rational transduction. Then*

$$L \in \mathbf{ALG}(\mathbf{X}^*) \Rightarrow \tau(\mathbf{L}) \in \mathbf{ALG}(\mathbf{Y}^*)$$

Proof. As shown in chapter II.5 we can write $\tau(L)$ as

$$\tau(L) = \beta(\alpha^{-1}(L) \cap R)$$

where $\alpha : X^* \to Z^*, \beta : X^* \to Y^*$ are monoid homomorphisms and $R \in \mathbf{REG}(Z^*)$ is a regular language.

$$\begin{aligned} L \in \mathbf{ALG}(\mathbf{X}^*) &\Rightarrow \alpha^{-1}(L) \in \mathbf{ALG}(\mathbf{Z}^*) \text{ by theorem 2.4} \\ &\Rightarrow \alpha^{-1}(L) \cap R \in \mathbf{ALG}(\mathbf{Z}^*) \text{ by theorem 2.3} \\ &\Rightarrow \beta(\alpha^{-1(L)} \cap R) \in \mathbf{ALG}(\mathbf{Y}^*) \text{ by theorem 2.1} \end{aligned}$$

□

We want to prove another important closure property of the algebraic languages. To do so, we introduce the notion of **substitution**.

Definition 2.1. *Let X_∞ be a countably infinite alphabet (see chapter II.4), let $X \subset X_\infty$ and $\sigma : X \to \mathbf{ALG}(\mathbf{X}_\infty^*)$ be a mapping. $\mathbf{ALG}(\mathbf{X}_\infty^*)$ is a monoid with the complex product as its operation.*

The continuation of σ to a monoid homomorphism

$$\sigma : X^* \to \mathbf{ALG}(\mathbf{X}_\infty^*)$$

is called **algebraic substitution**.

It holds

THEOREM 2.5. *Let $L \in \mathbf{ALG}(\mathbf{X}^*)$ and $\sigma : X^* \to \mathbf{ALG}(\mathbf{X}^*_\infty)$ an algebraic substitution. Then*

$$\sigma(L) := \{\sigma(w) \mid w \in L\} \in \mathbf{ALG}(\mathbf{X}_\infty *)$$

PROOF. Let $\kappa = (\mathfrak{A}, \mathcal{K})$ be a PDA accepting L,

$$\mathfrak{A} = (G, X, \{s\}, \{f\}, \alpha), \quad G = (V, E), \quad \mathcal{K} = (E, Y, \delta)$$

Let $\kappa_x = (\mathfrak{A}_x, \mathcal{K}_x)$ be a PDA accepting $\sigma(x)$,

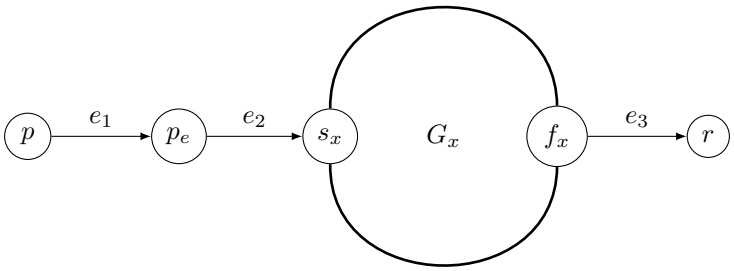
$$\mathfrak{A}_x = (G_x, A_x, \{s_x\}, \{f_x\}, \alpha_x), \quad G_x = (V_x, E_x)$$

and

$$\mathcal{K}_x = (E_x, Y_x, \delta_x)$$

We construct a PDA κ' accepting $\sigma(L)$:

Informal idea: In the PDA for L each edge labelled with x is substituted by a separate copy of the graph G_x together with 3 auxiliary edges. These edges are used to ensure that the resulting graph cannot accept more words that those from $\sigma(L)$.



Consider an edge $e : p \xrightarrow{x} r \in E,\ \alpha(e) = x \in X$.

Add new edges e_1, e_2, e_3 to E' and a new vertex p_e to V' with

$$e_1 : p \xrightarrow{\epsilon} p_e$$
$$e_2 : p_e \xrightarrow{\epsilon} s_x$$
$$e_3 : f_x \xrightarrow{\epsilon} r$$

Additionally, add a disjoint copy of the graph G_x to the graph G' and keep the labels, i.e. $\alpha'(e) = \alpha_x(e),\ e \in E_x$.

The pushdown computation is defined as follows:

$$\begin{aligned}
\delta'(e_1, y) &= \delta(e, y),\ y \in Y \cup \{\epsilon\} \\
\delta'(e_2, y) &= y \cdot \$_e,\ y \in Y \cup \{\epsilon\} \\
\delta'(e_3, \$_e) &= \epsilon \\
\delta'(e', \$_e) &= \$_e \cdot \delta_x(e', \epsilon),\ e' \in E_x,\ Q(e') = s_x \\
\delta'(e', y) &= \delta_x(e', y),\ e' \in E_x,\ Q(e') \neq s_x,\ y \in Y_x \cup \{\epsilon\} \\
\delta'(e', y) &= \infty\infty, \text{ else}
\end{aligned}$$

We get a PDA $\kappa' = (\mathfrak{A}', \mathcal{K}')$ with

$$\mathfrak{A}' = (G', X', \{s\}, \{f\}, , \alpha'), \quad G' = (V', E')$$

where the vertex set is

$$V' = V \cup \bigcup_{x \in X} V_x \cup \{p_e \mid e \in E\}$$

and the edge set is

$$E' = \bigcup_{x \in X} E_x \cup \{e_1, e_2, e_3 \mid e \in E\} \cup \{e \in E \mid \alpha(e) = \epsilon\}$$

The start and final state is the same as in PDA κ.

The edge labels are defined by

$$\alpha'(e) = \begin{cases} \alpha_x(e) & e \in E_x,\ x \in X \\ \epsilon & \text{else} \end{cases}$$

The pushdown store is defined as

$$\mathcal{K}' = (E', Y', \delta')$$

where the pushdown alphabet Y' is

$$Y' = \bigcup_{x \in X} Y_x \cup \{\$_e \mid e \in E,\ \alpha(e) \neq \epsilon\} \cup \{\infty\}$$

δ' is defined as shown above, additionally $\delta'(e, y) = \delta(e, y)$ for all edges with $\alpha(e) = \epsilon$.

Claim: It holds

$$L_{\kappa'} = \sigma(L)$$

"$\subset$":

Let $w \in L_{\kappa'}$ be a word accepted by PDA κ'. Then there exists a path $\pi \in \mathfrak{W}(G')(s, f)$ with label $\alpha'(\pi) = w$ and pushdown store computation $\delta'(\pi, \epsilon) = \epsilon$.

The path π can be decomposed into $\pi = \pi_1 \circ \ldots \pi_k$ and each subpath π can be decomposed into $\pi_i = e_1^{(i)} \circ e_2^{(i)} \circ \pi_i' \circ e_3^{(i)}$ where $\pi_i' \in \mathfrak{W}(G_{x_i})(s_{x_i}, f_{x_i})$ and $\alpha(e_i) = x_i$.

By construction it holds $\alpha'(\pi_i') \in \sigma(x_i),\ x_i \in X.\ \alpha'(e_i) = \epsilon \Rightarrow \alpha'(\pi_i') \in \sigma(x_i) \Rightarrow$

$$\begin{aligned} w &= \alpha'(\pi) \\ &= \alpha'(\pi_1) \cdots \alpha'(\pi_k) \in \sigma(x_1) \cdots \sigma(x_k) \\ &= \sigma(x_1 \cdots x_k),\ x_1 \cdots x_k \in X^* \end{aligned}$$

It remains to show $x_1 \cdots x_k \in L$.

Let by construction $e_1 \circ \ldots \circ e_k \in \mathfrak{W}(G)(s, f)$ be an accepting path in the pushdown automaton for L, so $\alpha(e_1 \circ \ldots \circ e_k) = x_1 \cdots x_k$.

We claim that the pushdown store computation for this path is the same as the one of the corresponding path in the original PDA:

$$\delta'(\pi_1 \circ \ldots \circ \pi_k, \epsilon) = \delta(e_1 \circ \ldots \circ e_k, \epsilon)$$

It holds

$$\begin{aligned}\delta'(\pi_1,\epsilon) &= \delta'(e_1^{(1)} \circ e_2^{(1)} \circ \pi_1' \circ e_3^{(1)}, \epsilon)\\ &= \delta'(e_1^{(2)} \circ \pi_1' \circ e_3^{(1)}, \delta(e_1,\epsilon))\\ &= \delta'(e_1^{(3)}, \delta(e_1,\epsilon)\cdot \$_{e_1})\\ &= \delta(e_1,\epsilon)\end{aligned}$$

By induction, the claim follows.

By assumption it holds $\delta'(\pi_1 \circ \ldots \pi_k, \epsilon) = \epsilon \Rightarrow \delta(e_1 \circ \ldots \circ e_k) = \epsilon$

$$\Rightarrow x_1 \cdots x_k \in L_\kappa \Rightarrow \alpha'(\pi) \in \sigma(L_\kappa) = \sigma(L)$$

"$\supset$":

Let $w \in \sigma(L)$ be a word in the target language of the substitution σ. Then there exists a word $v \in L$ with $w \in \sigma(v)$. For v there exists an accepting path $\pi \in \mathfrak{W}(G)(s,f)$ labelled $\alpha(\pi) = v$ and with pushdown computation $\delta(\pi,\epsilon) = \epsilon$.

Let $\pi = \pi_1 \circ \ldots \circ \pi_k$ be a decomposition of path π into single edges. To each edge π_i with label $\alpha(\pi_i) = x_i$ there exists a PDA κ_{x_i} accepting $\sigma(x_i)$.

For each edge π_i there exists in κ' a path

$$\pi_i' = {\pi'}_i^{(1)} \circ {\pi'}_i^{(2)} \circ \pi_{x_i} \circ {\pi'}_i^{(3)}$$

where $\delta_{x_i}(\pi_{x_i},\epsilon) = \epsilon$ and $\alpha'(\pi_i') \in \sigma(x_i)$ such that $\alpha'({\pi'}_1) \cdots \alpha'({\pi'}_k) = u$.

By construction of δ' it holds

$$\delta'(\pi_1' \circ \ldots \circ \pi_k', \epsilon) = \delta(\pi_1 \circ \ldots \circ \pi_k', \epsilon)$$

and from that follows $\delta'(\pi_1' \circ \ldots \circ \pi_k', \epsilon) = \epsilon$.

Together: $\alpha'(\pi_1' \circ \ldots \circ \pi_k') = u$ and $\delta'(\pi_1' \cdots \pi_k', \epsilon) = \epsilon \Rightarrow u \in L_{\kappa'}$.

This finishes the proof $L_{\kappa'} = \sigma(L)$. □

We have proved now the most important closure properties for algebraic languages.

In the next section we investigate the theorem of Chomsky-Schützenberger, another important theorem which gives a characterization of the algebraic languages.

EXERCISE 2.1. (1) *Let* $L \in \mathbf{ALG}(\mathbf{X}^*)$, $\mathbf{R} \in \mathbf{REG}(\mathbf{X}^*)$. *Show:*

$$LR^{-1} \in \mathbf{ALG}(\mathbf{X}^*) \text{ and } \mathbf{R^{-1}L} \in \mathbf{ALG}(\mathbf{X}^*)$$

(2) *For* $L \in X^*$ *let*

$$\begin{aligned}INIT(L) &:= \{u \mid uv \in L \text{ for } v \in X^*\}\\ FIN(L) &:= \{u \mid vu \in L \text{ for } v \in X^*\}\\ SUBWORD(L) &:= \{v \mid uvw \in L \text{ for } u,w \in X^*\}\end{aligned}$$

Show: $L \in \mathbf{ALG}(\mathbf{X}^*) \Rightarrow \mathbf{INIT(L)}, \mathbf{FIN(L)}, \mathbf{SUBWORD(L)} \in \mathbf{ALG}(\mathbf{X}^*)$.

(3) *Show: There exist algebraic languages* $L_1, L_2 \in \mathbf{ALG}(\mathbf{X}^*)$ *such that* $L_1^{-1}L_2 \notin \mathbf{ALG}(\mathbf{X}^*)$.

Hint: Set $L_1 = a\{b^i a^i \mid i \geq 1\}^*$ *and* $L_2 = \{a^i b^{2i} \mid i \geq 1\}^*$. *Prove that* L_1 *and* L_2 *are algebraic and take the intersection* $(L_1^{-1}L_2) \cap b^+$. *Then prove the following theorem: If* X *contains a single element, then each algebraic language over* X *is regular. From this the claim immediately follows.*

3. The theorem of Chomsky-Schützenberger

The Chomsky-Schützenberger theorem is one of the fundamental theorems in the theory of formal languages, especially the context-free languages [**CS63**].

In the literature, the proof of this theorem is always based on grammars.

We will give an automata-theoretic access to this theorem. We present this theorem at this point in the book because from a proof technical point of view it can very well be appended to the section on pushdown languages (chapter III.1). A similar access to this theorem has been given by Goldstine [**Gol77, Gol79, Gol80**].

The Chomsky-Schützenberger theorem characterizes the context-free languages by a very simple subclass of context-free languages with the use of a homomorphism.

In this respect the theorem is an analogon to lemma 7 in chapter II.1 where we proved that each regular language can be presented as the homomorphic image of a local language.

We want to formulate the theorem now.

Let $D(\Sigma)$ be the Dyck language over the alphabet Σ (see chapter I.3, page 19).

Remark: In the literature, often the language

$$\{w \in (\Sigma \cup \Sigma^{-1})^* \mid |w| = \epsilon \text{ in the free group } F(\Sigma)\}$$

is called Dyck language. We denote that language as the *symmetric* Dyck language.

With our notations the following theorem holds:

THEOREM 3.1 (Chomsky-Schützenberger). *For each language $L \in \mathbf{ALG}(\mathbf{X}^*)$ there exists*

an alphabet $\Sigma \subset X_\infty$,

a regular set $R \in \mathbf{REG}((\Sigma \cup \Sigma^{-1})^*)$ and

a monoid homomorphism $\phi : (\Sigma \cup \Sigma^{-1})^* \to X^*$ such that

$$L = \phi(D(\Sigma) \cap R)$$

(The proof in the original book was somewhat hard to read and has been reformulated therefore.)

PROOF. For each language $L \in \mathbf{ALG}(\mathbf{X}^*)$ there exists a PDA $\kappa = (\mathfrak{A}, \mathcal{K})$ with $L = L_\kappa$. Here, $\mathfrak{A} = (G, X, S, F, \alpha)$ is a finite automaton with graph $G = (V, E)$ and $\mathcal{K} = (E, Y, \delta)$ is a simple pushdown store.

For the proof we use the graph $G' = (V', E')$, $V' = V \cup \{s_0\}$ we constructed in theorem 2 of section III.1 and the mapping $\gamma : E' \to Y^{[*]}$ which encodes the pushdown computations by elements from the H-group over Y.

The graph G' has the following types of edges:

- A start edge $e'_s \in \{s_0\} \times S$ labelled by y_0

  $$e'_s : s_0 \xrightarrow{y_0} s$$

  for each start vertex $s \in S$.

- "PUSH(z)"-edges $e' = (e, y) \in E \times Y$ of the form

$$e' : Q(e) \xrightarrow{y^{-1}yz} Z(e)$$

for each edge $e \in E$ with $\delta(e, y) = yz$ and $|yz| \neq \epsilon$.

The prefix $y^{-1}y$ realizes the non-deterministic guess of the topmost stack symbol y.

- "Empty-stack"-edges $e' = (e, y) \in E \times Y$ of the form

$$e' : Q(e) \xrightarrow{y_0^{-1}y_0 \cdot \delta(e,\epsilon)} Z(e)$$

for each edge $e \in E$.

The prefix $y_0^{-1}y_0$ realizes the non-deterministic guess of the empty-stack symbol y_0.

- "POP(z)"-edges $e' = (e, z, y) \in E \times Y \times Y_0$ of the form

$$e' : Q(e) \xrightarrow{z^{-1}y^{-1}y} Z(e)$$

for each edge $e \in E$ with $\delta(e, z) = \epsilon$.

The suffix $y^{-1}y$ realizes the non-deterministic guess of the topmost stack symbol y after the symbol z has been popped.

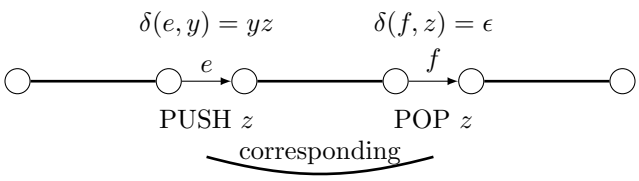
Our goal is to construct from G' a graph $\tilde{G}$ whose edges will constitute the alphabet $\Sigma \cup \Sigma^{-1}$ of the Dyck language.

First we define a relation $\rho' \subset E' \times E'$ on the edges of the graph G'. ρ' relates two edges if they represent a pair of corresponding push/pop-operations.

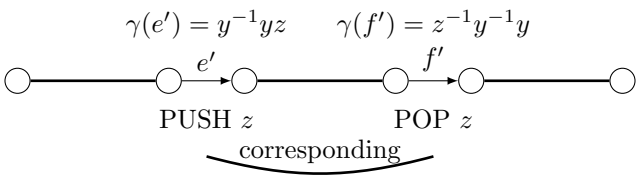
For edges $e', f' \in E'$ define

$$\begin{aligned} (e', f') \in \rho' \quad \Longleftrightarrow \quad & e' = (e, y) \text{ with } \delta(e, y) = yz \\ & f' = (f, z, y) \text{ with } \delta(f, z) = \epsilon \\ & e, f \in E,\ y, z \in Y \end{aligned}$$

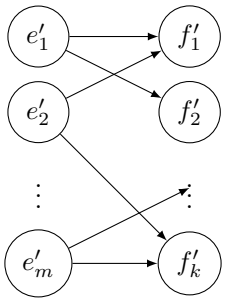
Consider a path in the graph G of the original pushdown automaton:



The corresponding path in the graph G':

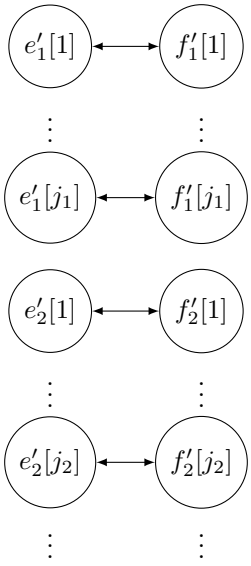


The relation ρ' between the edges of G' can be visualized as a bipartite graph:



The edges of this relation connect each edge from E' representing an "opening bracket" (push-operation) with those edges from E' representing the corresponding "closing bracket" (pop-operation).

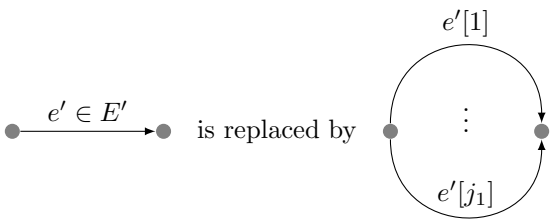
To get a one-to-one relation, the edges of the relation ρ' are multiplied as needed. The result is a new relation $\tilde{\rho}$ in which each opening bracket has a unique closing bracket:



The edges that have been created by multiplying an edge $e' \in E'$ are called the *parallel edges* of e'.

For edges $(\tilde{e}, \tilde{f}) \in \tilde{\rho}$ we also write $\tilde{f} = \tilde{e}^{-1}$ and call $\tilde{f}$ the *inverse edge* to $\tilde{e}$.

In the graph $\tilde{G}$ we replace the edges of G' by their parallel edges:



A parallel edge $e'[i] \in \tilde{E}$ has the same source and target as the original edge $e' \in G'$:

$$Q(e'[i]) = Q(e'), \quad Z(e'[i]) = Z(e')$$

By our construction it holds: If $(e', f') \in \rho'$ then there exists exactly one pair of parallel edges $(\tilde{e}, \tilde{f}) \in \tilde{\rho}$ such that $\tilde{f} = \tilde{e}^{-1}$.

From G' we construct the graph $\tilde{G} = (\tilde{V}, \tilde{E})$ by setting

$$\begin{aligned} \tilde{V} &:= V' \ (= V \cup \{s_0\}) \\ \tilde{E} &:= \{\tilde{e} \mid \tilde{e} \text{ is parallel edge for some } e' \in E'\} \end{aligned}$$

If $\tilde{e} \in \tilde{E}$ is a parallel edge for some $e' \in E'$ we write:

$$e' = \mathbf{p}(\tilde{e}) \quad (e' \text{ is the projection of } \tilde{e}$$

For each path $\tilde{\pi} \in \mathfrak{W}(\tilde{G})$ then holds $\mathbf{p}(\tilde{\pi}) \in \mathfrak{W}(G')$, i.e. the paths of parallel edges are projected to paths in G'.

We define the set of *valid paths* in $\tilde{G}$ by

$$\begin{aligned} \mathrm{VP} \ = \ & \{\tilde{e}_1 \cdots \tilde{e}_k \in \mathfrak{W}(\tilde{G}) \mid \tilde{e}_i \in \tilde{E} \\ & \text{and for each pair } (\tilde{e}_i, \tilde{e}_{i+1}) \text{ of consecutive edges holds:} \\ & \gamma(\mathbf{p}(\tilde{e}_i) \cdot \mathbf{p}(\tilde{e}_{i+1})) \not\equiv 0 \text{ in the polycyclic monoid } Y^{(*)} \} \end{aligned}$$

We define the set of *valid accepted* paths in $\tilde{G}$ by

$$\mathrm{VAP} = \{\tilde{\pi} \in \mathrm{VP} \mid Q(\tilde{\pi}) \in S, \ Z(\tilde{\pi}) \in F\} \subset \mathrm{VP}$$

Note that VP and VAP both are *local* sets.

We define the (positive) alphabet Σ as the set of those parallel edges that are projected to edges which execute a push-operation:

$$\Sigma := \tilde{E}^+ := \{\tilde{e} \in \tilde{E} \mid \mathbf{p}(\tilde{e}) = (e, y) \in E \times Y\}$$

Then the edge set of the graph $\tilde{G}$ is divided into $\tilde{E} = \tilde{E}^+ \cup \tilde{E}^-$.

For the proof of the theorem we need two lemmata.

A path in $\tilde{G}$ is called *well-nested* if it can be reduced to the empty path using the relations

$$\tilde{e} \cdot \tilde{e}^{-1} = \epsilon, \quad \tilde{e}, \tilde{e}^{-1} \in E'$$

(The lemma from the original book has been reformulated.)

LEMMA 3.1. *If $\tilde{\pi} \in \mathrm{VP}$ is a non-empty valid path and $\tilde{\pi}$ is well-nested, then for the projection $\pi' = \mathbf{p}(\tilde{\pi})$ holds:*

$$|\gamma(\pi')| = yy^{-1} \text{ for some } y \in Y$$

PROOF. The proof uses induction over the path length. Observe that a well-nested path has even length, therefore a well-nested, non-empty path has length at least two.

Remember the definition of the Dyck language $D(\Sigma)$ over an alphabet Σ:

$$\begin{aligned} d \in D(\Sigma) \quad &\Rightarrow \quad \sigma \cdot d \cdot \sigma^{-1} \in D(\Sigma) \text{ for each } \sigma \in \Sigma \\ d_1, d_2 \in D(\Sigma) \quad &\Rightarrow \quad d_1 \cdot d_2 \in D(\Sigma) \end{aligned}$$

Induction base: $\text{length}(\tilde{\pi}) = 2$

$$\begin{array}{rcl} \tilde{\pi} = \tilde{e} \cdot \tilde{e}^{-1} & \Rightarrow & (\tilde{e}, \tilde{e}^{-1}) \in \tilde{\rho} \\ & \Rightarrow & (\mathbf{p}(\tilde{e}), \mathbf{p}(\tilde{e}^{-1})) \in \rho' \text{ where} \\ & & \mathbf{p}(\tilde{e}) \in E \times Y \text{ is a "push"-edge and} \\ & & \mathbf{p}(\tilde{e}^{-1}) \in E \times Y \times Y_0 \text{ is the corresponding "pop"-edge} \end{array}$$

$\mathbf{p}(\tilde{\pi}) = \mathbf{p}(\tilde{e}) \cdot \mathbf{p}(\tilde{e}^{-1})$

If

$$\begin{array}{rcll} \gamma(\mathbf{p}(\tilde{e})) & = & y^{-1}yz & \text{("guess } y\text{; push } z\text{")} \\ \gamma(\mathbf{p}(\tilde{e}^{-1})) & = & z^{-1}y^{-1}y & \text{("pop } z\text{; guess } y\text{")} \\ \text{then} & & & \\ |\gamma(\mathbf{p}(\tilde{\pi}))| & = & |\gamma(\mathbf{p}(\tilde{e})) \cdot \gamma(\mathbf{p}(\tilde{e}^{-1}))| & \\ & = & |y^{-1}yz \cdot z^{-1}y^{-1}y| & \\ & = & |y^{-1}y(z \cdot z^{-1})y^{-1}y| & \\ & = & |y^{-1}(yy^{-1})y| & \\ & = & |y^{-1}y| & \\ & = & y^{-1}y & \end{array}$$

Induction step:

Case 1: Let the claim be true for a well-nested path $\tilde{\pi}$. Then $|\tilde{\pi}| = \epsilon$ and for the projection $\mathbf{p}(\tilde{\pi})$ holds

$$|\gamma(\mathbf{p}(\tilde{\pi}))| = {y_1}^{-1} \cdot y_1 \text{ for some } y_1 \in Y$$

Consider the path $\tilde{e} \cdot \tilde{\pi} \cdot \tilde{e}^{-1}$. It is well-nested because $\tilde{\pi}$ is well-nested.

For the projection

$$\mathbf{p}(\tilde{e} \cdot \tilde{\pi} \cdot \tilde{e}^{-1}) = \mathbf{p}(\tilde{e}) \cdot \mathbf{p}(\tilde{\pi}) \cdot \mathbf{p}(\tilde{e}^{-1}) \in \mathfrak{W}(G')$$

where

$$\begin{array}{rcll} \mathbf{p}(\tilde{e}) & = & (e_1, y) & \text{"push } z\text{-edge"} \\ \mathbf{p}(\tilde{e}^{-1}) & = & (e_2, z, y) & \text{"pop } z\text{-edge"} \end{array}$$

we get:

$$\begin{array}{rcl} |\gamma(\mathbf{p}(\tilde{e}) \cdot \mathbf{p}(\tilde{\pi}) \cdot \mathbf{p}(\tilde{e}^{-1}))| & = & |\gamma(\mathbf{p}(\tilde{e})) \cdot \gamma(\mathbf{p}(\tilde{\pi})) \cdot \gamma(\mathbf{p}(\tilde{e}^{-1}))| \\ & = & |\gamma((e_1, y)) \cdot ({y_1}^{-1}y_1) \cdot \gamma((e_2, z, y))| \\ & = & |(y^{-1} \cdot y \cdot z) \cdot ({y_1}^{-1} \cdot y_1) \cdot (z^{-1} \cdot y^{-1} \cdot y)| \end{array}$$

Because $\tilde{e} \cdot \tilde{\pi} \cdot \tilde{e}^{-1}$ is well-nested it must hold $z = y_1$. Otherwise from $z \cdot {y_1}^{-1} \equiv 0$ would follows $e \cdot \tilde{\pi} \cdot e^{-1} \equiv 0$ which would be a contradiction to our assumptions.

Therefore

$$\begin{aligned}|(y^{-1}\cdot y\cdot z)\cdot ({y_1}^{-1}\cdot y_1)\cdot (z^{-1}\cdot y^{-1}\cdot y)| &= |(y^{-1}\cdot y\cdot z)\cdot (z^{-1}\cdot z)\cdot (z^{-1}\cdot y^{-1}\cdot y)|\\ &= |y^{-1}\cdot (y\cdot (z\cdot z^{-1})\cdot (z\cdot z^{-1})\cdot y^{-1})\cdot y|\\ &= |y^{-1}\cdot (y\cdot y^{-1})\cdot y|\\ &= |y^{-1}\cdot y|\\ &= y^{-1}\cdot y\end{aligned}$$

Case 2: Let $\tilde{\pi} = \tilde{\pi}_1 \cdot \tilde{\pi}_2$ be a valid path which is the product of two well-nested paths, i.e. $|\tilde{\pi}_1| = |\tilde{\pi}_2| = \epsilon$.

Consider

$$\begin{aligned}|\gamma(\mathbf{p}(\tilde{\pi}))| &= |\gamma(\mathbf{p}(\tilde{\pi}_1\cdot\tilde{\pi}_2))|\\ &= |\gamma(\mathbf{p}(\tilde{\pi}_1))\cdot\gamma(\mathbf{p}(\tilde{\pi}_2))|\\ &= ({y_1}^{-1}\cdot y_1)\cdot ({y_2}^{-1}\cdot y_2),\ \text{for some } y_1, y_2 \in Y \text{ by induction hypothesis}\end{aligned}$$

Because $\tilde{\pi}_1 \cdot \tilde{\pi}_2$ is valid, it must hold $y_1 = y_2$. Otherwise it would be $y_1 \cdot {y_2}^{-1} \equiv 0$ which would violate our assumptions.

Therefore we get

$$\begin{aligned}|\gamma(\mathbf{p}(\tilde{\pi}))| &= ({y_1}^{-1}\cdot y_1)\cdot ({y_2}^{-1}\cdot y_2)\\ &= {y_1}^{-1}\cdot y_1\cdot {y_1}^{-1}\cdot y_1\\ &= {y_1}^{-1}\cdot y_1\end{aligned}$$

□

The lemma of course also holds for well-nested *accepted paths*. For the projection of such a path $\tilde{\pi} \in \text{VAP},\ \tilde{\pi} \equiv \epsilon$, holds

$$\pi = \mathbf{p}(\tilde{\pi}) \in \mathfrak{W}(G')(S,F) \Rightarrow |\gamma(\pi)| = {y_0}^{-1} y_0$$

where y_0 is the special symbol in the pushdown alphabet.

LEMMA 3.2. *For each path* $\pi' \in \mathfrak{W}(G')$ *with*

$$|\gamma(\pi')| = y^{-1}y \quad \textit{for some } y \in Y$$

there exists exactly one path $\tilde{\pi} \in \text{VP}$ *with*

$$|\tilde{\pi}| = \epsilon \textit{ and } \pi' = \mathbf{p}(\tilde{\pi})$$

PROOF. TODO □

Similar as after lemma 1 one can conclude here that for each path $\pi' \in \mathfrak{W}(G')(S,F)$ with $|\gamma(\pi')| = {y_0}^{-1}y_0$ there exists exactly one well-nested path $\tilde{\pi} \in \text{VAP}$ with $\mathbf{p}(\tilde{\pi}) = \pi'$ and $|\tilde{\pi}| = \epsilon$.

Using both lemmata we can now prove the theorem:

$$\begin{aligned}L_\kappa &= \{\alpha(\pi) \mid \pi \in \mathfrak{W}(G)(S,F),\ \delta(\pi,\epsilon) = \epsilon\} \text{ (by definition of PDA)}\\ &= \{\alpha(\pi') \mid \pi' \in \mathfrak{W}(G')(s_0,F),\ |\gamma(\pi')| = y_0\} \text{ (by theorem 2, chapter III.1)}\\ &= \{\alpha(\pi') \mid \pi' \in \mathfrak{W}(G')(S,F),\ |\gamma(\pi')| = {y_0}^{-1}y_0\} \text{ (by construction of } G')\end{aligned}$$

If we define $R := \text{VAP}$, we get $R \in \mathbf{REG}(\Sigma \cup \Sigma^{-1})^*$ because every local language is regular.

Together with lemma 1 and lemma 2 it follows

$$\mathbf{p}(R \cap D(\underbrace{\tilde{E}^+}_{\Sigma})) = \{\pi' \in \mathfrak{W}(G')(s_0, F) \mid |\gamma(\pi')| = y_0\}$$

With the homomorphism

$$\phi := (\mathbf{p} \circ \alpha) : (\Sigma \cup \Sigma^{-1})^* \to X^*$$

we get

$$L_\kappa = \phi(D(\Sigma) \cap R)$$

□

We even proved slightly more: Let

$$<\kappa, w> := \mathrm{card}(\{\pi \in \mathfrak{W}(G)(S, F) \mid \alpha(\pi) = w,\ \delta(\pi, \epsilon) = \epsilon\})$$

be the number of accepting computations of the PDA κ for the word w.

COROLLARY 3.1. *For each PDA κ there exists a Dyck language $D(\Sigma)$ and a local set R over $\Sigma \cup \Sigma^{-1}$ and a monoid homomorphism $\phi : (\Sigma \cup \Sigma^{-1})^* \to X^*$ such that*

(1) $\phi(D(\Sigma \cap R)) = L_\kappa$

(2) *For $w \in X^*$ it holds $<\kappa, w> = \mathrm{card}(\phi^{-1}(w))$*

PROOF.

(1) Chomsky-Schützenberger theorem

(2) Follows from the construction in the proof. (The inverse homomorphism gives the different accepting computations for a word.)

□

Problem: Is it possible to always find a PDA κ accepting $L \in \mathbf{ALG}(\mathbf{X}^*)$ such that for each word w it holds $<\kappa, w> < \infty$?

If the definition of the pushdown store is generalized such that $\delta(X \times Y) \subset Y^*$ is allowed, is then the condition $<\kappa, w> < \infty$ always satisfiable?

This problem will be investigated in a later chapter.

At the end of this section, for interested readers we want to mention the following:

The Dyck language $D(X) \in \mathbf{ALG}(\mathbf{X}^*_\infty)$ is algebraic.

By theorem 1 in section III.2 it holds that the class of algebraic languages is closed under monoid homomorphism, and by theorem 3 in chapter III.2 also closed under intersection with regular sets.

From the Chomsky-Schützenberger theorem therefore follows that the class of algebraic languages over the alphabet X_∞ is exactly the class of context-free languages.

4. The finite automaton with storage

In the following we slightly modify the notion of a pushdown-store by admitting arbitrary monoids instead of the free monoid Y^*.

Doing so, the given definition of the pushdown-store cannot be used any longer because for arbitrary monoids the notion of a "topmost" factor cannot be defined in a meaningful way.

In our finite automaton with storage the control unit puts symbols onto the store without caring about its current content which gives an even stronger separation between control unit and pushdown-store as we had for pushwon automata.

With the finite automaton with storage we have a universal model of automata.

DEFINITION 4.1. $\mathfrak{P} = (\mathfrak{A}, \mathcal{P})$ *is called* **finite automaton with storage** $\Longleftrightarrow$

$$\begin{gathered}\mathfrak{A} = (G, X, S, F, \alpha),\ (G = (V, E),\ \textit{is a finite automaton}\\ \mathfrak{P} = (E, M, D, C, \gamma)\ \textit{is a storage with monoid } M\\ \gamma : E^* \to M\ \textit{is a monoidhomomorphism}\\ D, C \subset M\ \textit{(D=domain, C= codomain)}\end{gathered}$$

$$L_{\mathfrak{P}} = \{\alpha(\pi) \mid \pi \in \mathfrak{W}(G)(S, F),\ D \cdot \gamma(\pi) \cap C \neq \emptyset\}$$

is the **language accepted by** $\mathfrak{P}$.

Remark: In the following we will also just write "finite automaton with storage monoid M" if this does not lead to any misunderstandings.

The following theorem gives a relation between the pushdown automaton and the finite automaton with storage.

THEOREM 4.1. *If $\kappa = (\mathfrak{A}, \mathcal{K})$ is a pushdown automaton, then there exists a finite automaton with storage $\mathfrak{P}$ with storage monoid $Y^{[*]}$ such that $L_\kappa = L_{\mathfrak{P}}$.*

PROOF. We remember the proof of the Chomsky-Schützenberger theorem in the previous section.

Let

$$\mathfrak{A}' = (G', Y^{[*]}, S_0, F, \gamma),\ G' = (V', E')$$

be the finite automaton used in the proof with accepted language

$$L_\kappa = \{\alpha(\pi) \mid \pi \in \mathfrak{W}(G')(S_0, F),\ |\gamma(\pi)| = y_0\}$$

We set

$$M := Y^{[*]}, \qquad E' \to Y^{[*]}$$

Then for the automaton with storage $\mathfrak{P} = (\mathfrak{A}, \mathcal{P})$ with

$$\mathcal{P} = (E', Y^{[*]}, \{\epsilon\}, \{y_0\}, \gamma)$$

it holds $L_\kappa = L_{\mathfrak{P}}$. □

It is now possible to prove similar closure properties for our new type of automaton as we did before for pushdown automata, in the corresponding constructions one has to complement these for the used monoids.

We do not want to execute these constructions here.

THEOREM 4.2. *Let $\mathfrak{P} = (\mathfrak{A}, \mathcal{P})$ be a finite automaton with storage monoid $Y^{(*)}$, $C = D = \{\epsilon\}$. Then there exists a pushdown automaton $\kappa = (\mathfrak{A}, \mathcal{K})$ with $L_{\mathfrak{P}} = L_\kappa$.*

PROOF. Let $\mathfrak{P} = (\mathfrak{A}, \mathcal{P})$ with

$$\mathfrak{A} = (G, X, S, F, \alpha),\ G = (V, E), \quad \mathcal{P} = (E, Y^{(*)}, \{\epsilon\}, \{\epsilon\}, \gamma)$$

Without loss of generality let $\gamma(e) \in Y \cup Y^{-1} \cup \{\epsilon\}$ (this can be achieved by breaking the graph edges).

We define the pushdown automaton $\kappa = (\mathfrak{A}, \mathcal{K})$ as follows:

$$\mathcal{K} = (E, Y_1^*, \delta) \text{ with } Y_1 = Y \cup \{0\}$$

The pushdown computation δ is

$$\delta : E \times Y_1 \to (Y_1 \times Y_1) \cup Y_1 \cup \{\epsilon\}$$

defined by

$$\delta(e, y) = \begin{cases} |y \cdot \gamma(e)| & \text{if } \gamma(e) \in Y \cup \{\epsilon\} \text{ or } \gamma(e) = y^{-1} \\ y \cdot 0 & \text{else} \end{cases}$$

Claim: $L_\kappa \stackrel{!}{=} L_{\mathfrak{P}}$

"$\supset$": Let $w \in L_{\mathfrak{P}}$, then there exists a path $\pi \in \mathfrak{W}(G)(S, F)$ with label $\alpha(\pi) = w$ and storage computation $\gamma(\pi) = \epsilon$ and for each prefix path π' of π holds:

$$|\gamma(\pi')| = \delta(\pi', \epsilon)$$

This is easily proved by induction and left to the reader.

Therefore $w \in L_\kappa$.

"$\subset$": Let $w \in L_\kappa$, then there exists a path $\pi \in \mathfrak{W}(G(S, F))$ with label $\alpha(\pi) = w$ and pushdown computation $\delta(\pi, \epsilon) = \epsilon$.

Wit part 1 the claim holds again from which follows $w \in L_{\mathfrak{P}}$.

From both parts follows the claim. □

Remark: The theorem also holds if we use the H-group $X^{[*]}$ as storage monoid (exercise).

From theorem 1 and theorem 2 it follows the equivalence of the automata models with storage monoid $X^{[*]}$.

As an example we give a finite automaton with storage which accepts the language L over the alphabet $X = \{a, b, c\}$ defined as follows:

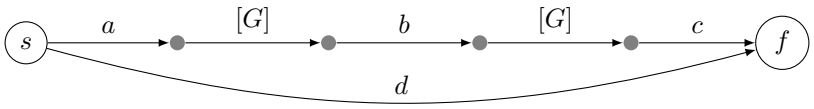
(1) $d \in L$

(2) $x, y \in L \Rightarrow a \cdot x \cdot b \cdot y \cdot c \in L$

(3) L is minimal with 1) and 2)

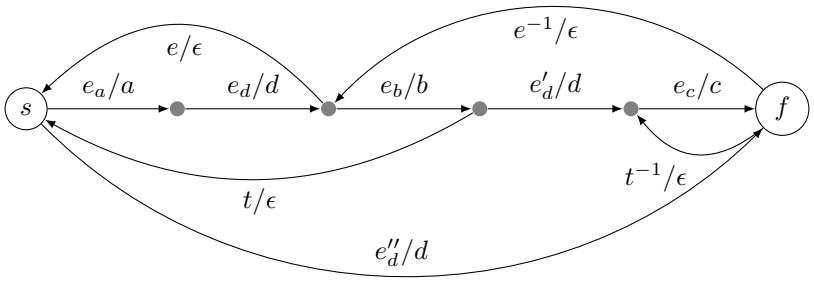
L can be understood as the language of the bracketed arithmetic expressions.

We create the following preliminary (infinite) graph G for L:



We want to remove each edge labeled with $[G]$ by adding an ϵ-edge from the source of this edge to s and from f to the target of the $[G]$- edges.

We get the following graph G':



The graph $G' = (V', E')$ is finite but the corresponding finite acceptor would also accept words which are not elements of L. This has to be corrected by defining a suitable storage.

Define $\gamma : E' \to \{a, b, s, t\}^{(*)}$ by

$$\begin{aligned}
\gamma(e_a) &= a \\
\gamma(e) &= s \\
\gamma(e_d) &= \epsilon \\
\gamma(t) &= t \\
\gamma(e_b) &= a^{-1} \cdot b \\
\gamma(e_c) &= b^{-1} \\
\gamma(e'_d) &= \epsilon \\
\gamma(e''_d) &= \epsilon \\
\gamma(e^{-1}) &= s^{-1} \\
\gamma(t^{-1}) &= t^{-1}
\end{aligned}$$

Domain and codomain contain the empty word only, the storage monoid M is the H-group over X:

$$D = C = \{\epsilon\}, \quad M = \{a, b, s, t\}^{[*]}$$

For

$$\mathfrak{A} = (G', \{a, b, cd\}, \{s\}, \{f\}, \alpha)$$

and storage

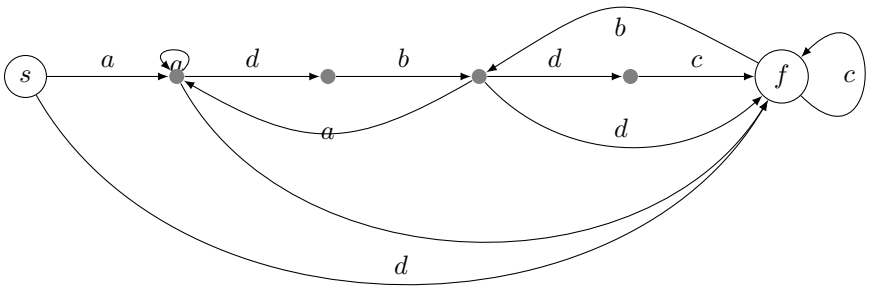
$$\mathcal{P} = (E', M, D, C, \gamma)$$

the finite automaton with storage $\mathfrak{P} = (\mathfrak{A}, \mathcal{P})$ accepts L:

$$L_{\mathfrak{P}} = \{\alpha(\pi) \mid \pi \in \mathfrak{W}(G')(s, f),\ D \cdot \gamma(\pi) \cap C \neq \emptyset\} = L$$

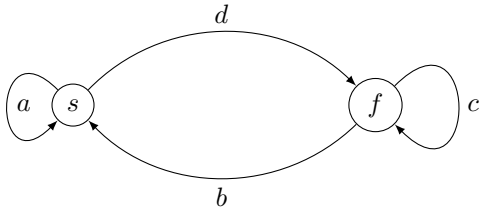
Proof: exercise.

Now we want to remove the ϵ-edges from the graph G'. To do so, we use the algorithm from chapter II, lemma 2 and get:



The storage is given by

We can simplify the graph even further and get



This graph is deterministic and the storage has the following form:

$$\gamma(e_a) = a,\ \gamma(e_d) = \epsilon,\ \gamma(e_b) = a^{-1}b,\ \gamma(e_c) = b^{-1}$$

We want to look at an example motivated by practical applications: we define parts of the syntax of the PASCAL programming language using finite automata with storage.

Every programming language is based on some vocabular. Programs are constructed by linear composition of these basic symbols in a way defined by the syntax rules.

The syntax rules are formulated such that is is easy to verify if a sequence of basic symbols is legal.

The syntax rules are most often (with advantage over other ways) represented by flow diagrams which are called *syntax diagrams*.

The possible paths in these syntax diagrams represent all possible symbol sequences.

Starting with a diagram labeled "program", along the path one either switches to another diagram if a rectangle is met, one appends a basic symbol w to the program text if a circle labelled with w is met. A complete example for this kind of syntac definition can be found in [**Wir72**].

We will present syntax diagrams slightly different. We use kind of a dual graph by making the vertices of the syntax diagrams to edges of our graphs.

In the following example we always show first the syntax diagram as given in [**Wir72**] and afterwards show the graph of our finite automaton with storage that accepts the language defined by the syntax diagram.

A figure is missing here!

Remark: A program in PASCAL by this definition is a block followed by a dot.

A figure is missing here!

[Block] may be interpreted as a procedure call of a procedure "Block" (corresponds to the substitution of chapter III.2).

Storage:

$$\gamma(e_1) = s, \quad \gamma(e_2) = s^{-1}, \qquad D = C = \{\epsilon\}$$

In this very simple example no difficulties arise. We want to consider now a diagram in which the same diagram's name occurs again. This corresponds to a recursive procedure call. We have to copy the diagram at this location.

A figure is missing here!

This is the syntax diagram which describes a "block" in PASCAL. It consists of a declaration section and a statement section where the declaration section may be empty.

G_{block} is shown below:

A figure is missing here!

For simplicity the nonterminal symbols are shortened to single characters.

e and e^{-1} are edges inverse to each other. In the storage we have to take care that the nesting depth of the block is correct. The edge e takes care that in a function or procedure-declaration the body appears correctly and in case of multiple nesting the end of the block is always correct.

In the way we described now every syntax diagram can be tranlated to a finite automaton with storage.

For each edge whose label is enclosed by the brackets [ ] there exists a finite automaton with storage accepting the language associated with the edge.

By substitution of these graphs into the original graph we can then produce graphs without bracket labels.

With this construction we could construct a finite automaton wit storage for the complete syntax of the PASCAL programming language that accepts the syntactically correct PASCAL programs. We get exactly the syntactically correct programs as the labels of paths in our graph which additionally fulfill the codomain-condition of the storage.

By the insertion of ϵ-edges when resolving the recursion the automaton becomes non-deterministic.

In the next section we want to transform the non-deterministic finite automaton with storage into a deterministic one.

EXERCISE 4.1. *Given the alphabet* $X_k = \{a_1, \ldots, a_K\}$ *and the language* $L_k := \{a_1, \ldots, a_k\}^{4^k}$.

Define a finite automaton with storage accepting L_k *and prove its correctness.*

EXERCISE 4.2. *Let* $X = \{(,), [,], \$, c\},\ d \notin X$.

$$\begin{aligned} L_0 &= \{\epsilon\} \\ &\cup \{x_1 c y_1 c z_1 d \cdots d x_n c y_n c z_n d \mid n \geq 1,\ y_1 \cdots y_n \in \$D_2 \\ &\quad x_i, z_i \in X^*,\ i = 1, \ldots, n\} \end{aligned}$$

is called the Greibach language. *(D_2 denotes the Dynck language over* $\{(,), [,]\}$*).*

Show: There exists a finite automaton with storage accepting L_0.

5. The deterministic finite automaton with storage

In this section we will prove a very important theorem about the relation between non-determinism and determinism of finite automata with storage.

To be able to construct the deterministic automaton we need some definitions.

DEFINITION 5.1. *A finite automaton with storage* $\mathfrak{P} = (\mathfrak{A}, \mathcal{P})$ *is called*

- ϵ**-free** *if* $\mathfrak{A}$ *is* ϵ*-free,*
- **deterministic** *if* $\mathfrak{A}$ *is deterministic,*
- **complete** *if* $\mathfrak{A}$ *is complete.*

DEFINITION 5.2. $(S, +, \cdot, 0, 1)$ *is called a* **semi-ring** *if*

(1) $(S, +, 0)$ *is a commutative monoid*

(2) $(S, \cdot, 1)$ *is a monoid*

(3) *The distributive laws hold:*

$$\begin{aligned} a \cdot (b + c) &= a \cdot b + a \cdot c \\ (a + b) \cdot c &= a \cdot c + b \cdot c \end{aligned}$$

(4) $0 \cdot a = a \cdot 0 = 0$ *for all* $a \in S$

DEFINITION 5.3. *Let* M *be a monoid and* S *a semi-ring.*

$$S(M) := \{\lambda : M \to S \mid \mathrm{card}(\{m \mid \lambda(m) \neq 0\}) < \infty\}$$

is called the **monoid-ring** *of the monoid* M *over the semi-ring* S*.*

Addition and multiplication on $S(M)$ are defined as follows: For $\lambda_1, \lambda_2 \in S(M)$

$$\begin{aligned} (\lambda_1 + \lambda_2)(m) &= \lambda_1(m) + \lambda_2(m) \\ (\lambda_1 \cdot \lambda_2)(m) &= \sum_{m_1 \cdot m_2 = m} \lambda_1(m) \cdot \lambda_2(m) \end{aligned}$$

One can see that $(S(M), +, \cdot)$ again is a semi-ring. We also write

$$\lambda := \sum_{\substack{m \in M \\ \lambda(m) \neq 0}} m \cdot \lambda(m)$$

or also

$$\sum_{\substack{m \in M \\ \lambda(m) \neq 0}} m \cdot <\lambda, m>$$

The operations are defined in a way that one can calculate with these expressions as usual.

In the following we will most often consider the semi-ring $S = \mathbb{B}$, the **Boolean semi-ring**.

Before proving our main theorem we will show some other results on finite automata with storage.

First we want to give a representation for graphs in the polycyclic monoid $Y^{(*)}$.

Let $G = (V, E)$ be a finite graph with $\text{card}(E) = n$, let $\text{card}(Y) = 2$ and $2^k \geq n$.

Let $\beta : V \to Y^*$ be an injective mapping and let $\text{length}(\beta(v)) = k$ for each vertex $v \in V$.

Further let $G' = (\{v_0\}, E')$ be a single-vertex graph and $\phi : E \to E'$ a bijection on the edges, and let $Q(e') = Z(e') = v_0$ for each edge $e' \in E'$.

The mapping ϕ can be uniquely continued to a functor $\phi : \mathfrak{W}(G) \to \mathfrak{W}(G')$. As a functor, ϕ is also injective.

We complement G' by a storage $\gamma : E' \to Y^{(*)}$ by setting

$$\gamma(e') := u^{-1} \cdot v \text{ where } u = \beta(Q(\phi^{-1}(e'))) \text{ and } v = \beta(Z(\phi^{-1}(e')))$$

In this way the storage "characterizes" the paths $\phi(\mathfrak{W}(G))$.

Obviously it then holds

Lemma 5.1.

$$\begin{array}{rcl} \pi \in \phi(\mathfrak{W}(G)) & \Leftrightarrow & \gamma(\pi) \neq 0 \\ \gamma(\pi) \neq 0 & \Rightarrow & \gamma(\pi) = u^{-1} \cdot v \\ & & \textit{with } u = \beta(Q(\phi^{-1}(e'))) \\ & & \textit{and } v = \beta(Z(\phi^{-1}(e'))) \\ & & \textit{and } \pi = e'_1 \circ \pi' \circ e'_k \end{array}$$

Proof. "⇒":

Let $\pi \in \mathfrak{W}(G)$ be a path, $\pi = e'_1 \dots e'_k \Rightarrow Z(e'_i) = Q(e'_{i+1})$.

There exists a path $\psi \in \mathfrak{W}(G)$ such that $\psi = e_1 \dots e_k$ with $\phi(\psi) = \pi$.

$$\begin{array}{rl} \Rightarrow & \beta(Z(\phi^{-1}(e'_i))) = \beta(Q(\phi^{-1}(e'_{i+1}))) \\ \Rightarrow & \beta(Z(\phi^{-1}(e'_i)))^{-1} \cdot \beta(Q(\phi^{-1}(e'_{i+1}))) = 1 \\ \Rightarrow & \gamma(e'_1 \dots e'_k) = \beta(Q(\phi^{-1}(e'_1)))^{-1} \cdot \beta(Z(\phi^{-1}(e'_k))) \end{array}$$

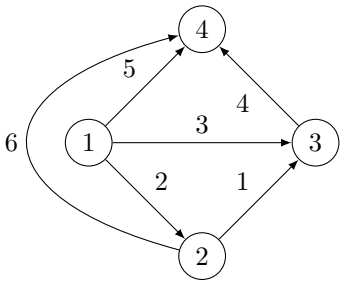
"⇐":

Let $\pi \notin \phi(\mathfrak{W}(G))$, $\pi = e'_1 \dots e'_k$ and there exists an index i such that $Z(\phi^{-1}(e'_i)) \neq Q(\phi^{-1}(e'_{i+1}))$

$$\begin{array}{rl} \Rightarrow & \beta(Z(\phi^{-1}(e'_i))) \neq \beta(Q(\phi^{-1}(s'_{i+1}))) \\ \Rightarrow & \beta(Z(\phi^{-1}(s'_i)))^{-1} \cdot \beta(Q(\phi^{-1}(e'_{i+1}))) = 0 \end{array}$$

□

We want to clarify the construction using an example. Let $G = (V, E)$ be the following graph:



It is $\text{card}(V) = 4 \Rightarrow k = 2\ (= \log_2(4))$.

The mapping $\beta : V \to Y^*,\ Y = \{(, [\}$ shall be defined by

$$\beta(1) = [[, \quad \beta(2) = [(, \quad \beta(3) = ([, \quad \beta(4) = (($$

Let $\phi : E \to E'$ be a bijection and $\gamma : E' \to Y^{(*)}$ with for example $\gamma(1) =)](.$

As a consequence to our construction we get

THEOREM 5.1. *For each regular language $L \in \mathbf{REG}(X^*)$ there exists a homomorphism $h : X^* \to \mathbb{B}(Y^{(*)})$ with*

$$L = h^{-1}(u^{-1} \cdot v + \sum), \quad \sum \textit{ finite sum in } \mathbb{B}(Y^{(*)})$$

$h(x),\ x \in X$ contains only monomials of length $2 \cdot \log_2(n)$ where $n = \text{card}(V)$.

PROOF. $L \in \mathbf{REG}(X^*)$, then there exists a deterministic finite automaton

$$\mathfrak{A} = (G, X, S, F, \alpha)$$

with $\text{card}(S) = \text{card}(F) = 1$ which accepts L.

To $G = (V, E)$ we construct the single-vertex graph $G' = (V', E')$ and the mapping $\phi : E \to E'$ as above. The labelling α is carried over to E' by the bijection ϕ. γ shall also be defined as above.

We merge all edges with the same label into a single edge e' an define

$$\gamma'(e') := \sum_s \gamma(s) \quad \text{the edges } e \text{ have the same label and are merged to } s'$$

and continue γ' to a homomorphism.

We define our homomorphism h as

$$h : X^* \to \mathbb{B}(Y^{(*)}) \text{ defined by } h(x) := \gamma'(e'),\ \alpha'(e') = x$$

Then it holds: $w \in L_{\mathfrak{A}} \Leftrightarrow h(w) = u^{-1} \cdot v + \sum$ with $u = \beta(S)$ and $v = \beta(F)$ (exercise) and the length-condition is also fulfilled. □

Now we want to use our graph embedding construction to transform special finite automata with storage and to approach our goal, the deterministic finite automaton with storage.

THEOREM 5.2. *Let $\mathfrak{P} = (\mathfrak{A}, \mathcal{P})$ be an ϵ-free automaton with storage monoid $Y^{(*)}$, let the domain and co-domain be given by $D = C = \{\epsilon\}$.*

Then there exists a deterministic finite automaton $\mathfrak{P}' = (\mathfrak{A}', \mathcal{P}')$ *with storage* $\mathbb{B}(Y^{(*)})$ *(if* $\operatorname{card}(Y) \geq 2$*) and with*

$$D' = \{\epsilon\}, \qquad C' = \{\beta(s)^{-1} \cdot \beta(f) + \mathbb{B}(Y^{(*)}) \mid s \in S,\ f \in F\}$$

such that $L_{\mathfrak{P}} = L_{\mathfrak{P}'}$.

If $\operatorname{card}(Y) = 1$ *we add a second element to* Y.

PROOF. In the first step we make $\mathfrak{A} = (G, X, S, F, \alpha)$ deterministic as we have seen in chapter II.1. We also can assume without loss of generality that $\alpha(e) \in X$ and $\operatorname{card}(S) = 1$ (exercise).

Let now

$$\mathfrak{A}' = (G', X, S', F', \alpha')$$

be the deterministic finite automaton where

$$G' = (V', E'), \quad V' = \mathfrak{P}(V)$$

□

CHAPTER 4

Context-Free Languages, $\mathbf{CF}(X^*)$

1. Normal forms of grammars

In chapter 1.6 we learned about the notion of a grammar, especially about context-free grammars. Now we want to define normal-forms for these grammars which are given by certain restrictions on the form of the grammar rules.

DEFINITION 1.1. *Let $G = (N, X, P, S)$ be a context-free grammar. G is in **Chomsky normal form** or CNF for short, if*

$$P \subset N \times N^2 \ \cup \ N \times X.$$

Additionally we require $\epsilon \in L(G) \iff S \to \epsilon \in P$. Here, $L(G)$ is the language generated by grammar G, see chapter 1.6.

We define

$$CF(X^\star) = \{L \mid L = L(G),\ G = (N, P, X, S) \text{ context-free grammar}$$

as the class of **context-free languages** over the alphabet X.

We want to show now that in the definition of $CF(X^*)$ we also can require that G is in Chomsky normal form. It holds

THEOREM 1.1. *For each context-free grammar G there exists a CNF grammar G' with $L(G) = L(G')$.*

PROOF. Let without loss of generality $\epsilon \notin L(G)$. We will construct G' from G in several steps.

Step 1:

Let $p \in P(G)$ with

$$p : Y \to t_1 Y_1 t_2 \dots t_k Y_k t_{k+1},$$

$t_i \in X,\ Y_i \in N^*,\ i = 1, \dots, k$ (or $k+1$ resp.).

We introduce new nonterminals $Y_{t_j},\ j = 1, \dots, k+1$, and replace rule p by new rules

$$Y \to Y_{t_1} Y_1 Y_{t_2} \dots\ Y_k Y_{t_{k+1}}$$

and

$$Y_{t_j} \to t_j,\ j = 1, \dots, k+1.$$

If $Y \to Z \in P,\ Y, Z \in N$, add a rule $Y \to w$ to the new production system for each $Z \to w \in P$ and $w \in X$ or $length(w) \geq 2$.

Let N'' be our nonterminal alphabet resulting from this construction.

We get a production system P'' where only terminal productions or productions of the form $Y \to Y_1 \dots Y_m,\ Y - i \in N'',\ m \geq 2$ exist.

Step 2:

Let $p : Y \to Y_1 \dots Y_m \in P'',\ m \geq 3$.

We introduce new nonterminals $Z_i,\ i = 1, \dots, m-2$ and replace p by

$$\begin{aligned} p_1 &: Y \to Y_1 Z_1 \\ p_2 &: Z_1 \to Y_2 Z_2 \\ &\vdots \\ p_{m-1} &: Z_{m-2} \to Y_{m-1} Y_m \end{aligned}$$

We do this for all rules p of the form above and obtain a grammar

$$G' = (N', X, P', s)$$

in Chomsky normal form. It is easy to show that $L(G') = L(G)$ (exercise). $\square$

2. The relationship between $\mathbf{CF}(X^*)$ and $\mathbf{ALG}(\mathbf{X}^*)$

Final remarks

Within the scope of this book, we could not handle efficient deterministic algorithms for deciding the word problem of context-free languages though our approach strongly would suggest this.

As a follow-up to Shamir's theorem one can easily derive the reduction of the word problem to the matrix multiplication as given by Valiant.

Our treatment of the 2-way finite automaton suggest the corresponding generalization for automata with (monoid-) storage.

Also for linear-bounded automata and other machine models, our approach can be applied without problems.

This composition of the theory shows in a new way the universality of the syntactic monoid of the Dyck-language. This can also be manifested by some algebraic representation theorems [**Hot81**].

It seems to be very promising to develop the whole theory of formal languages under this point of view as has already been done by Goldstine [**Gol77, Gol79, Gol80**] at different places.

Bibliography

[AHU76] A.V. Aho, J. E. Hopcroft, and J. D. Ullmann, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1976.

[Ber77] J. Berstel, *Series Formelles en variable non commutatives et applications*, Tech. report, LITP and ENSTA, 1977.

[Ber79] ———, *Transductions and Context-Free Languages*, Teubner, 1979, An online version of the first four chapters of this book is available.

[Boo80] R. Book, *Formal Languages Theory, Perspectives and Open Problems*, Academic Press, New York, 1980.

[BW77] H. Becker and H. Walter, *Formale Sprachen*, Vieweg, 1977.

[CF77] R.H. Crowell and R.H. Fox, *Introduction to Knot Theory*, Springer, New York, 1977.

[Com75] B. Commentz, *Platzkomplexität des Äquivalenzproblems endlicher Automaten*, Master's thesis, Universität des Saarlandes, Saarbrücken, 1975.

[CS63] N. Chomsky and M.P. Schützenberger, *The algebraic theory of context-free languages*, Computer Programming and Formal Systems (P. Brafford and D. Hirschberg, eds.), North Holland, 1963, pp. 118–161.

[EM45] S. Eilenberg and S. MacLane, *General Theory of Natural Equivalences*, Trans. Am. Math. Soc. (1945).

[FS74] G. Fischer and R. Sacher, *Einführung in die Algebra*, Teubner, 1974.

[Gin66] S. Ginsburg, *The Mathematical Theory of Context-Free Languages*, McGraw-Hill, 1966.

[Gol77] J. Goldstine, *Automata with Data Storage*, A conference on theoretical computer science (Waterloo Ontario), 1977, pp. 239–246.

[Gol79] ———, *A Rational Theory of AFLs*, Automata, Languages and Programming (Graz) (H. Maurer, ed.), Springer, 1979, pp. 271–280.

[Gol80] ———, *Formal Languages and Their Relation to Automata: What Hopcroft and Ullmann didn't tell us*, pp. 109–140, Academic Press, 1980.

[Gre73] S.A. Greibach, *The hardest context-free language*, SIAM Journal of Computing (1973), no. 2, 304–310.

[Gre75] ———, *Erasable context-free languages*, Information and Control (1975), no. 29, 301–326.

[Har78] M.A. Harrison, *Introduction To Formal Languages Theory*, Addison-Wesley, Reading, MA, 1978.

[HC72] G. Hotz and V. Claus, *Automatentheorie und Formale Sprachen III, Formale Sprachen*, BI, Mannheim, 1972.

[HM75] G. Hotz and J. Messerschmidt, *Dycksprachen sind in Bandkomplexität log n analysierbar*, Tech. report, Universität des Saarlandes, Saarbrücken, 1975.

[Hot74] G. Hotz, *Sequentielle Analyse kontextfreier Sprachen*, Acta Informatica (1974), no. 4, 55–75.

[Hot77] ———, *Space-Komplexität von klammerfreien Ausdrücken mit Typen*, EIK (1977), no. 13, 655–657.

[Hot78] ———, *Normal-Form Transformations of Context-Free Grammars*, Acta Cybernetica **4** (1978), no. 1, 65–84.

[Hot81] ———, *Ein Darstellungssatz für Algebren und Anwendungen in der Theorie der Formalen Sprachen*, Tech. report, Universität des Saarlandes, 1981.

[HR79] G. Hotz and R. Ross, *LL(k) und LR(k)-Invarianz von kontextfreien Grammatiken unter einer Transformation auf Greibach-Normalform*, EIK (1979), no. 15, 73–86.

[HU79] J.E. Hopcroft and J.D. Ullmann, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

[Kam09] Mark Kambites, *Formal Languages and Groups as Memory*, Communications in Algebra **37** (2009), no. 1, 193–208.

[Niv68] M. Nivat, *Transductions des languages de Chomsky*, Annales de l'institut Fourier (1968), no. 18, 339–456.

[Per77] J.F. Perrot, *Monoides syntactiques des languages algébriques*, Acta Informatica (1977), no. 7, 393–413.

[RK09] Elaine Render and Mark Kambites, *Rational subsets of polycyclic monoids and valence automata*, Information and Computation **207** (2009), no. 11, 1329 – 1339.

[Ros67] D.J. Rosenkrantz, *Matrix equations and normal forms for context-free grammars*, Journal of the ACM (1967), no. 14, 501–507.

[RS59] M.O. Rabin and D. Scott, *Finite automata and their decision problems*, IBM Journal of Research and Development (1959), no. 3, 114–125.

[Sak76] J. Sakarovitch, *Monoides syntactiques et languages algébriques*, Ph.D. thesis, Thèse 3ème cycle math, Paris, 1976.

[Sal73] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.

[Sha67] E. Shamir, *A representation theorem for algebraic and context-free power series in noncommutating variables*, Information and Control (1967), no. 11, 239–254.

[SS78] A. Salomaa and M. Soittola, *Automata Theoretic Aspects of Formal Power Series*, Springer, New York, 1978.

[Wag70] K. Wagner, *Graphentheorie*, BI, Mannheim, 1970.

[Wir72] N. Wirth, *Systematisches Programmieren*, Teubner, Stuttgart, 1972.

[Zet16] Georg Zetzsche, *Monoids as storage mechanisms*, Bulletin of the EATCS **120** (2016), 237–249, Invited abstract due to EATCS Distinguished Dissertation Award.