Formal Languages
An automata-theoretic introduction

Günter Hotz
Universität des Saarlandes

Klaus Estenfeld
Universität des Saarlandes

Armin Reichert
Alumnus Universität des Saarlandes
English Translation

Preface

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

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

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

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

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

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

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

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

Saarbrücken, August 1981

Günter Hotz

Contents

Introduction

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Then it holds:

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

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

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

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

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

TODO: Figure

If we consider all labelings of paths from S to F we get of course also words not contained in $D(X_k)$, for example $x_1 x_2 \bar{x}_k$ or $x_1 \bar{x}_1 \bar{x}_2$ etc.

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

Let us consider the word $x_1 x_2 \bar{x}_2 \bar{x}_1 x_2 \bar{x}_2 \in D(X_2)$, then we have paths $s_{x_1} s_2 s_{x_2} s_{\bar{x}_2} s_3 s_{\bar{x}_1} s_1 s_{x_2} s_{\bar{x}_2}$ and $s_{x_1} s_2 s_{x_2} s_{\bar{x}_2} s_3 s_{\bar{x}_1} s_3 s_2 s_{x_2} s_{\bar{x}_2}$ which both have this word as their labeling and we can easily define a homomorphism in the sense above.

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

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

CHAPTER 1

Mathematical foundations

1. Notations, basic notions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

It is easy to see that

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

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

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

We define

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

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

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

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

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

The following example should clarify that: Let $A = \{a, b, c\}$ and let (a, b, a) and $(b, a) \in A^*$.

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

.

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

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

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

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

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

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

We define **left** and **right quotient**:

$$B^{-1} \cdot A = \{v \mid \exists u \in B, w \in A : u \cdot v = w\}$$

and $A \cdot B^{-1}$ in the same way.

Because of the reduction rules it holds:

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

If $\{w\}^{-1} \cdot \{v\}$ is not empty we call w a **prefix** of v. If $\{w\} \cdot \{v\}^{-1} \neq \emptyset$, we call v a **suffix** of w. In the future we will always write just w instead of $\{w\}$ and also w is prefix of v if $w^{-1} \cdot v \neq \emptyset$.

2. Monoid homomorphisms and congruence relations

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

(1) $\Phi(m_1 \cdot m_2) = \Phi(m_1) \cdot \Phi(m_2), \quad m_1, m2 \in M$

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

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

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

monomorphism: if Φ is injective

epimorphism: if Φ is surjective

isomorphism: if Φ is bijective

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

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

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

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

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

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

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

(1) If M_1 is a generating system of M and $\Phi 1, \Phi 2 : M \to S$ both are monoid homomorphisms which coincide on M_1, then $\Phi 1 = \Phi_2$.

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

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

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

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

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

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

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

If $\Phi : M \to S$ is a monoid homomorphism, then the sets $\{\Phi^{-1}(s) | s \in S\} \subset Pot(M)$ is a monoid isomorphic to $\Phi(M)$.

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

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

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

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

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

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

$$a \equiv b(L) \Leftrightarrow \textit{For all } u, v \in M : uav \in L \Leftrightarrow ubv \in L$$

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

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

(2) If we define $[a]_L \cdot [b]_L := [ab]_L$ (complex product), then $synt_M(L) = \{[a]_L \mid a \in M$ becomes a monoid and the mapping $\Psi_L : M \to synt_M(L)$ with $\Psi_L(a) = [a]_L$ is a monoidepimorphism.

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

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

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

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

3. Special monoids and the free group

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Definition 3.2.

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

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

Now we introduce a special absorbing element 0 by defining:

Definition 3.3.

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

is called the **polycyclic monoid**.

Using the naming of the previous section we get:

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

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

Definition 3.4.

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

is the **free group** over X.

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

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

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

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

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

Lemma 3.2.

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

Proof:

"$\Leftarrow$":

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

"$\Rightarrow$":

Let $[w] = [w']$. We may assume that w' is created from w by application of an equation $xx^{-1} = 1$. Let $w = w_1 x x^{-1} w_2$ and $w' = w_1 w_2$.

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

Proof by induction over k:

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

$k > 0$: It holds $\rho(w) = \rho(w_1 x x^{-1} w_2), \quad \rho(w') = \rho(w_1 w_2)$. The reduction length of $\rho(w)$ by induction proposition is $k - 1$ and $\rho^k \rho(w) = \rho^{k-1}\rho(w') \Rightarrow$ the reduced word of w and w' is the same so $|w| = |w'|$.

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

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

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

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