Tommi J. M. Lehtinen

# Numbers and Languages

Turku Centre *for* Computer Science

TUCS Dissertations
No 158, March 2013

# Numbers and Languages

Tommi J. M. Lehtinen

*To be presented, with the permission of the Faculty of Mathematics and Natural Sciences of the University of Turku, for public criticism in Auditorium Cal4 on March 15, 2013, at 12 noon.*

University of Turku
Department of Mathematics and Statistics
FI-20014 Turku
Finland

2013

## Supervisors

Docent Alexander Okhotin
Department of Mathematics and Statistics
University of Turku
FI-20014 Turku
Finland

## Reviewers

Professor Martin Kutrib
Institut für Informatik
Universität Gießen
Gießen
Germany

Professor Lila Kari
Department of Computer Science
University of Western Ontario
London, Ontario
Canada

## Opponent

Docent Juha Kortelainen
Department of Information Processing Science
University of Oulu
Linnanmaa
FIN-90570 Oulu
Finland

ISBN 978-952-12-2849-0
ISSN 1239-1883

# Acknowledgements

Alexander Okhotin for supervising my graduate studies and providing me nice problems to think about.
Juhani Karhumäki for answering yes on my question whether there is a possibility for me to continue my studies.
Lila Kari and Martin Kutrib for pre-examination of the thesis.
Anne Kivelä for checking the language of the thesis and other stuff.
Personnel of the department for making it a pleasant working environment.
My family for always being there for me and especially my parents for giving me my life.
My friends for all the good times.

# Contents

# Introduction

This thesis presents theoretical results published during the graduate studies of the author. It is divided in two parts: Numbers and Languages.

Although the first part is written in terms of numbers, the historical background is in formal language theory. Formal languages are sets of words, where words are sequences of letters from some fixed alphabet. Language equations are equations $\varphi(X_1, \dots X_m) = \psi(X_1, \dots X_m)$, where $X_i$ are variables over languages and $\varphi$ and $\psi$ are expressions containing variables, constants and language-theoretic operations. A system of language equations

$$\begin{aligned} \varphi_1(X_1, \dots X_n) &= \psi_1(X_1, \dots X_n) \\ &\vdots \\ \varphi_n(X_1, \dots X_n) &= \psi_n(X_1, \dots X_n) \end{aligned}$$

is a set of language equations. Language equations have been a part of formal language theory since Seymour Ginsburg and H. Gordon Rice defined the semantics for context-free grammars in 1962 [6] by the least solutions of systems of language equations of the resolved form

$$\begin{aligned} X_1 &= \varphi_1(X_1, \dots X_n) \\ &\vdots \\ X_n &= \varphi_n(X_1, \dots X_n), \end{aligned}$$

where the operations of concatenation and union are allowed. Furthermore, Alexander Okhotin has introduced generalizations of the context-free grammar, the so-called conjunctive and Boolean grammars, that have their semantics defined through language equations using concatenation, union and intersection in the case of conjunctive grammars and, in addition, complementation in the case of Boolean grammars.

It was proved in 1985 by Parikh et al. [33], that the problem of solvability of systems of language equations with concatenation and Boolean operations is undecidable. This undecidability result was made more specific by Okhotin [27, 31, 29] in 2003, who proved that these systems are computationally complete and the problem of solution existence co-r.e.-complete. A

further completeness result was achieved in 2005, when Michal Kunc [18, 19] proved that there exists a finite language $L$ such that the greatest solution of the commutation equation

$$XL = LX$$

is co-r.e.-complete. This suprising result answered the question by John Conway [3] from 1971, asking if the greatest solution is regular if $L$ is. The proof relies upon the non-commutativity of concatenation of languages. A recent survey was written by Kunc and Okhotin [20], for a more thorough overview on language equations.

In the first part of this thesis, computational universality of simple systems of equations over sets of natural numbers or, equivalently, of systems of language equations over a unary alphabet is presented. It is the final step in a quest for finding simpler systems of equations over the unary alphabet that are computationally universal. That quest began after Artur Jeż answered negatively to the question proposed by Okhotin, asking if conjunctive grammars can generate non-regular unary languages. Jeż gave an example of a grammar generating the language $\{a^{4^n} \mid n \in \mathbb{N}\}$ [12], showing that the conjunctive grammars are more powerful than context-free grammars also in the unary case. The technique used by Jeż was further developed by him and Okhotin, who proved that systems of equations over unary alphabet using operations of concatenation and union and finite constants are computationally complete [13]. It is based on the representation of computation histories of Turing machines as $k$-ary representations of natural numbers. They further improved this result by showing that computational universality can be achieved with systems using only concatenation and regular constants [14]. In this thesis, this development is taken to its conclusion. It is shown that computational universality can be achieved by systems of language equations over a unary alphabet

$$\begin{aligned} XK &= L \\ XXM &= XXN, \end{aligned}$$

where $K, L, M$ and $N$ are unary regular languages.

This is done in Chapter 2, where the result is presented in terms of natural numbers. Jeż and Okhotin also used this terminology, and considered systems of equations over sets of natural numbers as the primary object rather than systems over unary alphabet. This is quite natural, as the length of unary words characterizes them and it is presented with a natural number. In Section 2.2 it is proved that systems of equations over sets of numbers of the form

$$\begin{aligned} X + A &= B \\ X + X + C &= X + X + D, \end{aligned}$$

where $A, B, C$ and $D$ are eventually periodic constant sets of numbers and the addition of sets is defined elementwise, are computationally universal. This follows from the presentation of all recursive sets in an encoded form as unique solutions of that kind of systems. This result was first presented in DLT 2010 conference [24]. However, they cannot represent all recursive sets as shown in Section 2.3, which is based on the article [25].

The second part of the thesis concerns languages, the results falling within the sphere of closure properties of language families. In Chapter 4, it is proved that the family of languages generated by Boolean grammars is closed under injective and inverse gsm-mappings. The proof for this is just a generalization of the argument used by Ginburg and Ullian [8], who proved that unambiguous context-free languages are closed under inverse gsm-mappings. However, this is more of a tribute for the argument than an underration of the result presented here. Closure properties of language families are among the most important theoretical results in formal language theory, so this is a crucial step in the development of the theory of Boolean languages. The proof also applies for unambiguous Boolean languages and for the conjunctive languages and their unambiguous variant. It uses injective gsm-mappings, and closure of the families mentioned under them is achieved on the side. The chapter is based on the article [23].

The last chapter of the thesis takes another view on closure properties. The question is not if a family is closed under a class of some operations, but under which operations in the class. There is a beautiful correspondence between codes that can be deciphered deterministically and context-free languages that can be recognized deterministically. The family of deterministic context-free languages is closed under a code, an injective morphism, if and only if it is a so-called code of bounded deciphering delay. This is, at least after reading the proofs, not a suprising result, and the most suprising thing about it is that it was not known before. This result was presented at DLT 2012 conference [26].

# Part I

# Numbers

# Chapter 1

# Numbers everywhere

Numbers are everywhere. If you look at your hands, you will probably see two. There is one and there is the other one, so there are two. In sports competitions, the results are expressed by numbers so that the winner is labeled with one, followed by two, three, four, five, six, seven, eight, nine, ten, eleven, twelve, thirteen, and so on. In stores, products are labeled with numbers expressing their prices. Products also very often have numbers in their names like Commodore 64, for example. Even songs have numbers in them, like the song *When I'm sixty-four* by The Beatles. You see, there is no escape from numbers. New examples of using numbers in different ways could be given one after another to the point of exhaustion of both the reader and the writer.

Numbers are very convenient to use. Imagine 64 elephants. They are heavy. Then imagine another 64 elephants. Now you have 128 elephants. That is a lot of elephants! But since they are expressed by numbers, they can be imagined almost without effort. Now sell 23 of those $2 \cdot 64 = 128$ elephants. You get an amount of money and still have 105 elephants left. In the real world, dividing a group of 128 elephants in two groups, 23 elephants and 105 elephants, would require quite a lot of work. But there are no elephants around, just the numbers. And it is fairly easy to say $128 = 23 + 105$ or $128 - 23 = 105$. Hence, a lot of things can be expressed in a nice and convenient way with a little help from numbers.

Numbers also lie in the core of mathematics, as reasoning about natural numbers is the starting point for anybody becoming familiar with the science. They are taken for granted in the elementary school, because there is no need to define them formally. They are just numbers and people become familiar with them without even noticing it. If you are reading this, you probably know the natural numbers up to some extent. If you wouldn't, you couldn't have read this far. However, there is always something new to learn, something more.

The set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ is the profound structure of mathematics. Everything else, at least numberwise, is built on top of them. Starting from the integers $\mathbb{Z}$ with negative numbers, continuing to rationals $\mathbb{Q}$ with fractions and then to reals $\mathbb{R}$, complex numbers $\mathbb{C}$ and beyond. At a glance, the set of natural numbers seems simple and easy to understand. However, they have been under a very active study for thousands of years, and no ending to that quest of understanding different aspects of $\mathbb{N}$ is at sight.

The study of natural numbers, known as number theory, has led to the development of heavy theoretical machinery used in almost all other areas of mathematics, and consequently, in other sciences using mathematics as a tool. Natural numbers have been used in calculations as obvious or given through the ages. Still, a precise logical definition for them was established only as late as in the end of the 19th century. This was done by Giuseppe Peano, who built upon the insights of preceding mathematicians such as Richard Dedekind. The Peano axioms, as they are called, are essentially so-called second-order sentences. The induction axiom, stating that a subset of natural numbers including zero and the following number of every number in it equals the whole set of natural numbers, needs to be expressed through quantification over subsets. Any attempt to axiomatize natural numbers by axioms of first-order is doomed to fail, as any set of first-order axioms has so-called non-standard models besides the standard that is the familiar set of natural numbers.

The existence of these non-standard models of natural numbers, or actually of any infinite logical structure expressed by first-order axioms, is a consequence of the Löwenheim-Skolem theorem. The Löwenheim-Skolem theorem states that if a countable first-order theory has an infinite model, then it has infinite models of cardinality of any infinite cardinal. The non-standard models form the foundation of non-standard analysis, introduced by Abraham Robinson around 1960. Non-standard analysis gives a firm ground to the so-called infinitesimal numbers that already Leibniz, who provided some of the foundations of analysis, used intuitively. Infinitesimal numbers have also been used in computations here and there more or less rigorously.

Somewhat related to the impossibility of describing the natural number by first-order sentences is Gödel's incompleteness theorem, stating that any finitely describable set of first-order axioms is incapable of having all true first-order statements about natural numbers as logically derivable consequences. This result by Kurt Gödel is one of the most significant results in mathematics in the 1900s, or even in the history of mathematics overall. It gave a death punch to David Hilbert's program to prove everything, which in a way is a drawback. On the other hand, it has the comforting consequence that there will always be new mathematical results to prove. It also gives insight on why some problems concerning the familiar and simple set

of natural numbers are so hard to solve.

Another example of the difficulty of questions related to natural numbers is Hilbert's 10th problem presented in 1900. It is about the so-called Diophantine equations, which are equations of the form

$$p(x_1, \ldots, x_n) = q(x_1, \ldots, x_n),$$

where $p$ and $q$ are polynomials with integer coefficients. One of the most famous Diophantine equations is

$$a^n + b^n = c^n.$$

Fermat's last theorem, proposed by Pierre Fermat in 1673 claiming he had a proof too long to fit in the margin, states that the equation does not have a solution where $a, b, c > 0$ are natural numbers and $n$ is a natural number larger than two. Fermat's last theorem was escaping all efforts to prove it right or wrong for centuries, finally proved to be true in 1995 by Andrew Wiles. Hilbert's 10th problem asked for an algorithm solving if a given Diophantine equation has an integer solution. The problem is stated in terms of integers, but it can be shown using Lagrange's four-square theorem that the problem can be phrased equivalently in terms of natural numbers as well. The problem was answered by Yuri Matiyasevich in 1970, who proved that there actually is no algorithm solving it. So the answer is that there is no answer, and the problem is impossible to solve.

Similarly, this thesis also presents a problem in terms of natural numbers. A problem easy to state, but impossible to solve. It is the problem of solving a system of equations

$$\begin{aligned} X + A &= B \\ X + X + C &= X + X + D \end{aligned}$$

where $A, B, C, D \subseteq \mathbb{N}$ are eventually periodic sets of natural numbers, $X$ is a variable with subsets of natural numbers as valuations, and the addition of sets is defined elementwise. The hardness of the problem is a consequence of the possibility to encode every computable set in the unique solution of a system of the above kind. This means that solving this kind of a simple system of equations is as hard as solving any computational problem in general.

## 1.1 Equations, algebra and topology

Let $(\mathcal{A}, \mathcal{O})$ be an algebra. This means that $\mathcal{A}$ is a set and $\mathcal{O}$ a set of operations from $\mathcal{A}$ to itself. The constants of algebraic structures are usually defined as 0-ary operations. For example, a monoid would be an algebraic structure

$(\mathcal{M}, \cdot, e)$, where $e$ is the unit element with respect to the product $(\cdot)$. However, in this thesis, the constants are dropped, so the monoid would be denoted just by $(\mathcal{M}, \cdot)$ with an implicit assumption about the existence of a unit element.

If $A, B \in \mathcal{A}$, then the equation

$$A = B$$

states that $A$ and $B$ are the same element of $\mathcal{A}$. Usually equations also contain variables $\mathcal{X}$ that are not elements of the algebra. If $X$ and $Y$ are variables, then the equation

$$X = Y$$

has a solution $X = A$, $Y = B$ if and only if $A = B$. The mapping $X \to A$, $Y \to B$ is called a valuation of variables. In general, a valuation of variables is a mapping from $\mathcal{X}$ to $\mathcal{A}$, or an element of the power $\mathcal{A}^{\mathcal{X}}$ defined as a collection of elements from $\mathcal{A}$ indexed with elements from $\mathcal{X}$:

$$\mathcal{A}^{\mathcal{X}} = \{(S_X)_{X \in \mathcal{X}} \mid S_X \in \mathcal{A}\}.$$

The power is accompanied with projections $\Pi_X \colon \mathcal{A}^{\mathcal{X}} \to \mathcal{A}$ defined by

$$\Pi_X(\mathcal{S}) = S_X \text{ for } \mathcal{S} \in \mathcal{A}^{\mathcal{X}}.$$

The set of $n$ variables $\{X_1, \ldots, X_n\}$ is denoted by $\mathcal{X}_n$. In this case, notation

$$\mathcal{A}^n = \{(S_1, \ldots, S_n) \mid S_i \in \mathcal{A}\}$$

is used for the product and $\Pi_i$ for the projection $\Pi_{X_i}$. In the one variable case, the index is dropped and the notation $\mathcal{X}_1 = \{X\}$ is used. Other letters for variables besides $X$ are used also and in this case one can write, for example, $\mathcal{X}_3 = \{X, Y, Z\}$. It should always be clear from the context which names are used for the variables, usually they are capital Latin letters from the end of the alphabet. The sets of variables $\mathcal{X}$ are assumed to be finite.

The use of elements of $\mathcal{A}$ can be limited by fixing a set $\mathcal{C} \subseteq \mathcal{A}$ of constants that are allowed in the equations. For a more detailed description of equations, there are the following definitions.

**Definition 1.1.1.** *Expressions over the algebra $(\mathcal{A}, \mathcal{O})$ with variables $\mathcal{X}$ and constants $\mathcal{C}$, denoted $\mathcal{E}[\mathcal{X}, \mathcal{C}, (\mathcal{A}, \mathcal{O})]$, are defined recursively:*

- *$A$ is an expression for all $A \in \mathcal{C}$*
- *$X$ is an expression for all $X \in \mathcal{X}$*
- *If $\varphi_i(\mathcal{X})$ for $i = 1, \ldots, m$ are expressions and $f \in \mathcal{O}$ is an $m$-ary operation, then $f(\varphi_1(\mathcal{X}), \ldots, \varphi_m(\mathcal{X}))$ is an expression.*

Every expression $\varphi \in \mathcal{E}[\mathcal{X}, \mathcal{C}, (\mathcal{A}, \mathcal{O})]$ with variables $\mathcal{X}$ defines a mapping

$$\varphi \colon \mathcal{A}^{\mathcal{X}} \to \mathcal{A}$$

For every $\mathcal{S} \in \mathcal{A}^{\mathcal{X}}$, the value of $\varphi(\mathcal{S})$ is defined in the following way:

**Definition 1.1.2.** *Let $\varphi \in \mathcal{E}[\mathcal{X}, \mathcal{C}, (\mathcal{A}, \mathcal{O})]$ be an expression and $\mathcal{S} \in \mathcal{A}^{\mathcal{X}}$. Then value of $\varphi(\mathcal{S})$ is defined recursively:*

- *If $\varphi = A \in \mathcal{C}$, then*
$$\varphi(\mathcal{S}) = A$$
- *If $\varphi(\mathcal{X}) = X \in \mathcal{X}$, then*
$$\varphi(\mathcal{S}) = \Pi_X(\mathcal{S})$$
- *If*
$$\varphi = f(\varphi_1, \ldots, \varphi_n)$$
*for some $\varphi_i \in \mathcal{E}[\mathcal{X}, \mathcal{C}, (\mathcal{A}, \mathcal{O})]$ with the values $\varphi_i(\mathcal{S}) = S_i \in \mathcal{A}$ and an $n$-ary operation $f \in \mathcal{O}$, then*
$$\varphi(\mathcal{S}) = f(S_1, \ldots, S_n)$$

Equations are statements on the equality of expressions:

**Definition 1.1.3.** *Let $\varphi, \psi \in \mathcal{E}[\mathcal{X}, \mathcal{C}, (\mathcal{A}, \mathcal{O})]$ be expressions. Then*

$$e \colon \quad \varphi = \psi$$

*is the equation $e$.*

*The expression $\varphi$ is the left-hand and $\psi$ is the right-hand side of the equation. If $\mathcal{S} \in \mathcal{A}^{\mathcal{X}}$ and $\varphi(\mathcal{S}) = \psi(\mathcal{S})$, then $\mathcal{S}$ is a solution of $e$.*

The set of equations over the algebra $(\mathcal{A}, \mathcal{O})$ with variables $\mathcal{X}$ and constants $\mathcal{C}$ is denoted by $\mathbb{E}[\mathcal{X}, \mathcal{C}, (\mathcal{A}, \mathcal{O})]$.

Systems of equations are sets of equations:

**Definition 1.1.4.** *A system of equations in $\mathfrak{S}[\mathcal{X}, \mathcal{C}, (\mathcal{A}, \mathcal{O})]$ contains equations in $\mathbb{E}[\mathcal{X}, \mathcal{C}, (\mathcal{A}, \mathcal{O})]$. If $\mathcal{S} \in \mathcal{A}^{\mathcal{X}}$ is a solution of all the equations $e \in \mathfrak{S}$, then it is a solution of the system $\mathfrak{S}$.*

If the set $\mathcal{A}$ is accompanied with an order $\leqslant$, then one can talk about the least and greatest solutions of a system. A solution $\mathcal{S} \in \mathcal{A}^{\mathcal{X}}$ of a system is the least solution, if for all solutions $\mathcal{S}' \in \mathcal{A}^{\mathcal{X}}$ it holds that $\Pi_X(\mathcal{S}) \leqslant \Pi_X(\mathcal{S}')$ for all $X \in \mathcal{X}$. Symmetrically, $\mathcal{S}$ is the greatest solution if $\Pi_X(\mathcal{S}') \leqslant \Pi_X(\mathcal{S})$ for all solutions $\mathcal{S}' \in \mathcal{A}^{\mathcal{X}}$ and variables $X \in \mathcal{X}$.

If the algebra has an order $\leqslant$, the operations in $\mathcal{O}$ are called monotone with respect to $\leqslant$ if for every $m$-ary operation $f \in \mathcal{O}$

$$f(\mathcal{S}) \leqslant f(\mathcal{S}')$$

whenever $\mathcal{S}, \mathcal{S}' \in \mathcal{A}^{\mathcal{X}}$ and $\Pi_X(\mathcal{S}) \leqslant \Pi_X(\mathcal{S}')$ for all $X \in \mathcal{X}$.

If the order has the property that for any elements $A, B \in \mathcal{A}$ there exists a greatest element $A \wedge B$ with $A \wedge B \leqslant A, B$ and a least element $A \vee B$ with $A, B \leqslant A \vee B$, then $(\mathcal{A}, \leqslant)$ is a lattice. Lattices can be defined also algebraically through $\wedge$ and $\vee$. In this case, $(\mathcal{A}, \wedge, \vee)$ is a lattice if the operations are commutative:

$$A \wedge B = B \wedge A \text{ for all } A, B \in \mathcal{A}$$

$$A \vee B = B \vee A \text{ for all } A, B \in \mathcal{A},$$

associative:

$$(A \wedge B) \wedge C = A \wedge (B \wedge C) \text{ for all } A, B, C \in \mathcal{A}$$

$$(A \vee B) \vee C = A \vee (B \vee C) \text{ for all } A, B, C \in \mathcal{A}$$

and mutually distributive:

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C) \text{ for all } A, B, C \in \mathcal{A}$$

$$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C) \text{ for all } A, B, C \in \mathcal{A}.$$

The definitions of lattices through order and algebra are equivalent. The algebraic operations can be defined by the order as mentioned above. And if the operations are given, then the order can be defined by

$$A \leqslant B \text{ if and only if } A \wedge B = A.$$

A lattice is bounded if there are $A, B \in \mathcal{A}$ such that for every $C \in \mathcal{A}$ it holds that $A \leqslant B$. It is complete, if there is a greatest lower bound and a least upper bound for every subset of $\mathcal{A}$. A complete lattice is bounded as the greatest lower bound and the least upper bound for $\mathcal{A}$ are the least and greatest elements of $\mathcal{A}$ respectively.

A system is in resolved form, if all equations in it consist of equations of the form

$$X = \varphi_X(\mathcal{X}),$$

for $X \in \mathcal{X}$ and $\varphi_X \in \mathcal{E}[\mathcal{X}, \mathcal{C}, (\mathcal{A}, \mathcal{O})]$. If expressions are allowed on the left-hand side of the equations as well, the system is unresolved. A resolved system over an algebra with a complete lattice-order and monotone operations always has the least and greatest solutions:

**Lemma 1.1.1** (Tarski's fixed point theorem [35]). *Let $\mathfrak{S} \in \mathfrak{S}[\mathcal{X}, \mathcal{C}, (\mathcal{A}, \mathcal{O})]$ be a resolved system of equations for an algebra $(\mathcal{A}, \mathcal{O})$ with a complete lattice-order and monotone operations. The least and greatest solutions exist.*

Most algebraic stuctures in this thesis can be seen as ultra-metric spaces, defined as:

**Definition 1.1.5.** *Let $X$ be a set and $d\colon X^2 \to \mathbb{R}_{\geqslant 0}$ a mapping. Then $(X, d)$ is an ultra-metric space, if the following hold:*

$$\begin{aligned} d(x,x) &\geqslant 0 \text{ for all } x \in X \\ d(x,y) &= d(y,x) \text{ for all } x, y \in X \\ d(x,z) &\leqslant \max(d(x,y), d(y,z)) \text{ for all } x, y, z \in X \end{aligned}$$

If $(X, d)$ is an ultra-metric space and $n \in \mathbb{N}$, then $(X^n, d_{\max})$, where

$$d_{\max}((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = \max(d(x_1, y_1), \ldots, d(x_n, y_n))$$

for $(x_1, \ldots, x_n), (y_1, \ldots, y_n) \in X^n$, is also an ultra-metric space. The notation $(X^n, d_{\max}) = (X^n, d)$ without the index will be used from now on.

If $(X, d)$ and $(X', d')$ are ultra-metric spaces, then a mapping $f\colon X \to X'$ is called continuous, if for every $\varepsilon > 0$ there exists $\delta > 0$ such that $d(x, y) < \delta$ implies $d'(f(x), f(y))$ for all $x, y \in X$. An equivalent condition for continuity can be given through converging sequences: $f\colon X \to X'$ is continuous if and only if for every converging sequence $(x_k)_{k\in\mathbb{N}} \xrightarrow[k\to\infty]{} x$ of elements of $X$ it holds that

$$\lim_{k\to\infty} f(x_k) = f(x).$$

An ultra-metric space $(X, d)$ is compact if every open cover of $X$ has a finite subcover or, equivalently, if every sequence of elements of $X$ has a converging subsequence. If $(X, d)$ is compact, then $(X^n, d)$ is compact as well.

The following Lemma states an important property of compact ultra-metric spaces that is used later.

**Lemma 1.1.2.** *Let $(X, d)$ be a compact ultra-metric space and $\varphi, \psi\colon X^n \to X$ continuous mappings. For all $\varepsilon > 0$ there exists $\delta > 0$ such that if $x, y \in X^n$, $d(\varphi(x), \psi(x)) < \varepsilon$, $d(x, y) < \varepsilon$ and $d(\varphi(y), \psi(y)) < \delta$, then there exists $z \in X^n$ such that $\varphi(z) = \psi(z)$ and $d(x, z) < \varepsilon$.*

*Proof.* Let $\varepsilon > 0$. Suppose the contrary: There exists $x \in X^n$ such that $d(\varphi(x), \psi(x)) < \varepsilon$ and for every $0 < k \in \mathbb{N}$ there exists $y_k \in X^n$ such that $d(x, y_k) < \varepsilon$ and $d(\varphi(y)_k, \psi(y_k)) < \frac{1}{k}$, but there does not exist such a $z$.

Since $X$ is compact, the sequence $y_k$ has a subsequence $y_{k_i}$converging to some $z \in X^n$, so $\varphi(y_{k_i}) \xrightarrow[i\to\infty]{} \varphi(z)$ and $\psi(y_{k_i}) \xrightarrow[i\to\infty]{} \psi(z)$ as $\varphi$ and $\psi$ are continuous. Hence,

$$d(\varphi(z), \psi(z)) = \lim_{i\to\infty} d(\varphi(y_{k_i}), \psi(y_{k_i})) \leqslant \lim_{i\to\infty} \frac{1}{k_i} = 0,$$

and $\varphi(z) = \psi(z)$.

Furthermore,

$$d(x, z) \leqslant \max(d(x, y_{k_i}), d(y_{k_i}, z))$$

and since $d(x, y_{k_i}) < \varepsilon$ and $d(y_{k_i}, z) \to 0$, it follows that $d(x, z) < \varepsilon$. This is a contradiction and there exists such a $\delta$. □

## 1.2 Natural numbers

The Peano axioms defining the natural numbers are formulated as follows:

**Definition 1.2.1.** *The set of natural numbers is defined as the set with the properties*

- *Zero is a natural number:*

$$\big(0 \in \mathbb{N}\big).$$

- *If $n$ is a natural number, then $n + 1$ is a natural number:*

$$\big(n \in \mathbb{N}\big) \text{ implies } \big(n + 1 \in \mathbb{N}\big).$$

- *If $n$ is a natural number, then $n + 1$ does not equal zero:*

$$\big(n \in \mathbb{N}\big) \text{ implies } \big(n + 1 \neq 0\big).$$

- *If $m$ and $n$ are natural numbers and $m + 1$ and $n + 1$ are equal, then $m$ and $n$ are equal:*

$$\big((m, n \in \mathbb{N}) \text{ and } (m + 1 = n + 1)\big) \text{ implies } \big(m = n\big).$$

- *If $S$ is a subset of natural numbers closed under $+1$ and containing zero, then $S$ is the whole set of natural numbers:*

$$\big(0 \in S \subseteq \mathbb{N} \text{ and } (S + 1) \subseteq S\big) \text{ implies } \big(S = \mathbb{N}\big).$$

Usually a successor function is used in the place of adding one. After that addition is defined using the successor function. In this thesis, addition is taken for granted, so such formalism is not needed. In general, the use of successor function has the advantage that it makes visible the presence of natural numbers in almost every infinite structure. Whenever an operation with an infinite trace is defined, it defines an isomorphic copy of the natural numbers.

In algebraic terms, the set of natural numbers with addition and zero $(\mathbb{N}, +)$ is a commutative monoid. This means that the addition is associative

$$(n_1 + n_2) + n_3 = n_1 + (n_2 + n_3) \text{ for all } n_1, n_2, n_3 \in \mathbb{N}$$

and commutative

$$n_1 + n_2 = n_2 + n_1 \text{ for all } n_1, n_2 \in \mathbb{N}$$

and that zero is the unit element with respect to addition

$$0 + n = n + 0 = n \text{ for all } n \in \mathbb{N}.$$

The monoid of natural numbers with addition $(\mathbb{N}, +)$ is a free monoid, which means that for every monoid $(\mathcal{M}, \cdot)$, any mapping $f\colon 1 \to m \in \mathcal{M}$ can be continued in a unique way to a monoid morphism $\widehat{f}\colon \mathbb{N} \to \mathcal{M}$ as $\widehat{f}(n) = m^n$.

If $p > 0$ is a natural number, then natural numbers $m$ and $n$ are said to be equal modulo $p$ if $p$ divides $m - n$. In this case, the set consisting of numbers equal $d$ modulo $p$, or the set

$$\{np + d \mid n \in \mathbb{N}\},$$

is called the coset $d$ modulo $p$.

## 1.3 Sets of natural numbers

Subsets of natural numbers, denoted $S \subseteq \mathbb{N}$ or $S \in \mathcal{P}(\mathbb{N})$, will be discussed in the remainder of the first part of this thesis.

The addition of $S, T \subseteq \mathbb{N}$ is defined elementwise so that $n \in S + T$ if and only if there exist such $k \in S$ and $l \in T$ that $n = k + l$.

$$S + T = \{k + l \mid k \in S,\ l \in T\}$$

The addition of sets of numbers has the same properties as the addition of numbers. It is associative

$$(S_1 + S_2) + S_3 = S_1 + (S_2 + S_3) \text{ for all } S_1, S_2, S_3 \subseteq \mathbb{N}$$

and commutative

$$S_1 + S_2 = S_2 + S_1 \text{ for all } S_1, S_2 \subseteq \mathbb{N}$$

and the singleton set $\{0\}$ is the unit element with respect to addition

$$\{0\} + S = S + \{0\} = S \text{ for all } S \subseteq \mathbb{N}.$$

This makes $(\mathcal{P}(\mathbb{N}), +)$ a commutative monoid.

Because sets are on the table, there are also set-theoretic operations, such as union, intersection and complementation. The union of two sets $S_1, S_2 \subseteq \mathbb{N}$ contains all numbers in those sets:

$$S_1 \cup S_2 = \{n \in \ | \ n \in S_1 \text{ or } n \in S_2\}.$$

The intersection of two sets $S_1, S_2 \subseteq \mathbb{N}$ contains the numbers in both sets:

$$S_1 \cap S_2 = \{n \in \ | \ n \in S_1 \text{ and } n \in S_2\}$$

and the complement of a set $S \subseteq \mathbb{N}$ contains the numbers not in $S$:

$$\overline{S} = \{n \in \ | \ n \notin S\}.$$

The union is associative

$$(S_1 \cup S_2) \cup S_3 = S_1 \cup (S_2 \cup S_3) \text{ for all } S_1, S_2, S_3 \subseteq \mathbb{N}$$

commutative

$$S_1 \cup S_2 = S_2 \cup S_1 \text{ for all } S_1, S_2, S_3 \subseteq \mathbb{N},$$

it has the empty set as the unit element

$$\varnothing \cup S = S \cup \varnothing = S \text{ for all } S \subseteq \mathbb{N},$$

and it is distributive over addition

$$S_1 + (S_2 \cup S_3) = (S_1 + S_2) \cup (S_1 + S_3) \text{ for all } S_1, S_2, S_3 \subseteq \mathbb{N}.$$

So $(\mathcal{P}(\mathbb{N}), \cup, +)$ is a commutative semiring.

Furthermore, the union is distributive over intersection

$$S_1 \cup (S_2 \cap S_3) = (S_1 \cup S_2) \cap (S_1 \cup S_3) \text{ for all } S_1, S_2, S_3 \subseteq \mathbb{N}$$

and vice versa

$$S_1 \cap (S_2 \cup S_3) = (S_1 \cap S_2) \cup (S_1 \cap S_3) \text{ for all } S_1, S_2, S_3 \subseteq \mathbb{N},$$

making $(\mathcal{P}(\mathbb{N}), \cup, \cap)$ or $(\mathcal{P}(\mathbb{N}), \subseteq)$ a lattice with respect to the inclusion order

$$S_1 \subseteq S_2 \text{ if and only if } (n \in S_1 \text{ implies } n \in S_2).$$

The lattice has $\varnothing$ and $\mathbb{N}$ as the least and greatest elements respectively, so it is bounded.

De Morgan's laws

$$\overline{S_1 \cup S_2} = \overline{S_1} \cap \overline{S_2}$$

and

$$\overline{S_1 \cap S_2} = \overline{S_1} \cup \overline{S_2},$$

express the connection between these three set operations. As a result, $(\mathcal{P}(\mathbb{N}), \cup, \cap, \overline{\phantom{x}})$ is a Boolean algebra.

The set of natural numbers can be embedded into the set of sets of natural numbers by the inclusion $n \hookrightarrow \{n\}$. With this inclusion in mind, the singleton sets $\{n\}$ appearing in sums are sometimes written without the brackets:

$$\{n\} + S = n + S.$$

There are some classes of subsets referred to later. The class of finite sets is denoted by $\mathcal{P}_{\text{fin}}(\mathbb{N})$. Another important class of subsets is that of eventually periodic sets $\mathcal{P}_{\text{evp}}(\mathbb{N})$. A set $S \subseteq \mathbb{N}$ is eventually periodic with period $p > 0$ if there is a number $n_0 \in \mathbb{N}$ such that

$$n \in S \text{ if and only if } n + p \in S$$

for every $n \geqslant n_0$.

In the following, the computational properties of sets of numbers are of particular interest. One of the most natural classes of sets in computational sense is that of recursive sets, which are those that can be recognized by algorithms that terminate on every input:

**Definition 1.3.1.** *A set $S \subseteq \mathbb{N}$ is recursive if and only if there exists an algorithm deciding whether $n \in S$ for all $n \in \mathbb{N}$.*

Furthermore, a set is called recursively enumerable (r.e) if there exists an algorithm that returns a positive answer on every number that belongs to that set, while the algorithm might not terminate on inputs not in the set. A set is co-recursively enumerable (co-r.e.) if its complement is recursively enumerative. A set is both r.e. and co-r.e. if and only if it is recursive.

A problem is called decidable if there exists an algorithm solving it that terminates on all instances. If there exists an algorithm that terminates on all positive instances, but might not terminate on negative instances, a problem is called an r.e. problem. Symmetrically, co-r.e. problems are those that have an algorithm terminating on all negative instances. A problem $P$ is r.e.-complete (co-r.e.-complete) if it is r.e. (co-r.e.) and for every r.e. (co-r.e.) problem there exists an algorithm that reduces the problem to $P$.

# Chapter 2

# The hardness of simple things

Some things seem simple. But in the path to know them, one always finds new things to learn. One can discover a new perspective to look at a thing from and can get new insights into an already familiar topic. One becomes more familiar with things over time spent with them. One gets to understand things in a deeper way than before, over and over again.

Throwing and catching balls is a familiar activity to most and even children do it, but there are people devoting their lives to master the skill of juggling. Likewise, playing games like chess or go can be started after the rules have been learnt, but people still get better at them after years of practice. Writing a thesis, playing an instrument, singing, talking, dancing, walking, running, skateboarding, standing, sitting, making a list...

All of these seemingly simple things take a moment to get acquainted with, but may take a lifetime or more to master. This is the case also in questions related to natural numbers. In this chapter, a simple system of equations is presented that is not so simple to solve.

## 2.1 Equations over sets of natural numbers

The hardness of solving a simple system of equations

$$\begin{aligned} X + A &= B \\ X + X + C &= X + X + D \end{aligned}$$

is discussed later in this chapter. The hardness of solving the system depends heavily on the meaning of $X$, $A$, $B$, $C$ and $D$. If they are reals with the usual addition, the system is often written in lower case

$$\begin{aligned} x + a &= b \\ x + x + c &= x + x + d, \end{aligned}$$

and solving it is easy. In the case $c = d$ the solution to the system of equations is $x = b - a$ and in the case $c \neq d$ there is no solution. The solution to the system is similar in every algebraic structure, in which $+$ has an inverse.

If $A$, $B$, $C$ and $D$ are eventually periodic subsets of the natural numbers and $X$ gets values from $\mathcal{P}(\mathbb{N})$, then solving the system gets much more interesting. Consider the following equation

$$X + \{0, 1\} = \mathbb{N}.$$

It has the solutions

$$S \subseteq \mathbb{N} \colon 0 \in S \text{ and } S \cap \{n, n+1\} \neq \varnothing \text{ for all } n \in \mathbb{N},$$

in other words, the sets that contain zero and one of any two consecutive numbers. This is just one example of how much more complicated the addition of sets is compared to the addition of numbers.

For other examples of relations of equations over sets of numbers with addition, there is a way of expressing a result proved by Joseph Lagrange in 1770, often presented during introductory courses on number theory. It states that every natural number can be expressed as a sum of four squares:

**Example 2.1.1** (Lagrange)**.** *The set of squares*

$$\{n^2 \mid n \in \mathbb{N}\},$$

*is a solution of the equation*

$$X + X + X + X = \mathbb{N}.$$

□

It is also possible to state one of the most famous open problems in mathematics by a simple equation in $\mathbb{E}[\mathcal{X}_1, \mathcal{P}_{\mathrm{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), +)]$:

**Example 2.1.2.** *The equation*

$$X + X = \{2n \mid n \geqslant 3\}$$

*has the set of odd primes as a solution if and only if the Goldbach's conjecture, stating that every even number at least six can be represented as a sum of two odd primes, holds.* □

So, questions about the addition of sets of numbers have been considered along the history of mathematics quite frequently, and the history will probably be repeating itself ad infinitum.

Theorem 2.2.1 is perhaps the most important contribution of this thesis. It concerns sets of numbers and says that

$$\begin{aligned} X + A &= B \\ X + X + C &= X + X + D \end{aligned}$$

is as hard to solve as any system of equations in $\mathfrak{S}[\mathcal{X}_n, \mathcal{P}_{\text{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +)]$.

These systems are hard to solve as a consequence of the following proposition, stating that all recursive subsets of natural numbers can be represented as components of unique solutions of systems in $\mathfrak{S}[\mathcal{X}_n, \mathcal{P}_{\text{fin}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +)]$.

**Proposition 2.1.1** (Jeż, Okhotin [13])**.** *Let $S \subseteq \mathbb{N}$. Then $S$ is recursive if and only if there exists $n \in \mathbb{N}$ and a system of equations in $\mathfrak{S}[\mathcal{X}_n, \mathcal{P}_{\text{fin}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +)]$ with a unique solution $X_i = S_i$ such that $S = S_1$.*

This expressive power implies hardness in solving these kinds of systems. Consider the example:

**Example 2.1.3.** *Let $\mathfrak{S} \in \mathfrak{S}[\mathcal{X}_n, \mathcal{P}_{\text{fin}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +)]$ be a system of equations with a unique solution $(S_1, \ldots, S_n)$ such that $S_1$ is the set of odd primes. This system exists by Proposition 2.1.1.*

*Now, if the equation*

$$X_1 + X_1 = \{2n \mid n \geqslant 3\}$$

*is added to the system, then the resulting system has a solution if and only if the Goldbach's conjecture holds.* □

However, there is no systematic way of finding out if that kind of a system of equations has a solution: the problem of solution existence is undecidable.

**Proposition 2.1.2** (Jeż, Okhotin [13])**.** *The problem whether a system of equations $\mathfrak{S}[\mathcal{X}, \mathcal{P}_{\text{fin}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +)]$ has a solution is co-r.e.-complete.*

## 2.2 The expressive power of simple systems

In this section, a system of equations

$$\mathfrak{S} \in \mathfrak{S}[\mathcal{X}_m, \mathcal{P}_{\text{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +)]$$

with $m$ variables and operations of union and addition is simulated by a system

$$\mathfrak{S}' \in \mathfrak{S}[\mathcal{X}_1, \mathcal{P}_{\text{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), +)]$$

with only one variable and addition. This simulation means that there is a correspondence between the solutions of the two systems. For this simulation

to be possible, the solutions $(S_1, \ldots, S_m) \in \mathcal{P}(\mathbb{N})^m$ of $\mathfrak{S}$ have to be represented by the solutions $S \in \mathcal{P}(\mathbb{N})$ of $\mathfrak{S}'$. Thus, there needs to be an encoding $\pi\colon \mathcal{P}(\mathbb{N})^m \to \mathcal{P}(\mathbb{N})$.

The sets will be encoded periodically, so that there is a period $p > 0$ such that for every $i = 1, \ldots, m$ there is a natural number $0 \leqslant d_i < p$ with the property that $n \in S_i$ if and only if $pn + d_i \in S$. In other words, the set $S_i$ is stored in the coset $d_i$ modulo $p$, called the track $d_i$ in this section. So the mappings

$$\tau_d^p\colon \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$$

defined by

$$\tau_d^p(S) = \{pn + d \mid n \in S\}$$

are needed for $d = 0, 1, \ldots, p-1$. If track $d$ modulo $p$ contains $\varnothing$ or $\mathbb{N}$, it is called empty or full respectively.

The encoding has to be able to simulate the sums

$$S_i + S_j$$

to be useful in simulating the $m$-variable system. Hence, if $S = \pi(S_1, \ldots, S_m)$ for some sets $S_1, \ldots, S_m \subseteq \mathbb{N}$, then the sum $S + S$ should have the encodings of those sums.

The sum of $S_i + S_j$ is on the track $d_i + d_j$ modulo $p$ of the set $S + S$ as

$$\tau_{d_i}^p(S_i) + \tau_{d_j}^p(S_j) = \tau_{d_i+d_j}^p(S_i + S_j).$$

In $S + S$, there are this kind of sums for every pair of indices. These sums have to be compared, so their tracks must not be the same for any disjoint indices. To achieve this, the sums $d_i + d_j$ should be different for different $(i, j)$. Although, for the pairs $(i, j)$ and $(j, i)$, the sums $d_i + d_j$ and $d_j + d_i$ are equal because $(\mathbb{N}, +)$ is commutative, and so the sums $S_i + S_j$ and $S_i + S_j$ will necessarily end up on the same track. But since the addition of sets is commutative as well, the sums are the same and this is not a problem.

The simplest way to do this is to demand that $d_{i_1} + d_{i_2} < d_{i_3} + d_{i_4}$ whenever $i_1, i_2 < \max(i_3, i_4)$. In this case, $d_i + d_i < d_{i+1} + d_1$ or $2d_i - d_0 < d_{i+1}$. So at the smallest $d_{i+1} = 2d_i - d_1 + 1$. The solution of this recursion equation is $d_i = d_1 + 2^{i-1} - 1$, so fixing $d_1$ will determine the rest.

There also needs to be a way to compare these different sums: if there is an equation

$$S_{i_1} + S_{i_2} = S_{i_3} + S_{i_4}$$

in the original system, there should be sets $E_{i_1,i_2}$ and $E_{i_3,i_4}$ such that the above equation holds if and only if the equation

$$S + S + E_{i_1,i_2} = S + S + E_{i_3,i_4}$$

of the constructed system holds. This is done by a set $E_+$ with the property that

$$S + S + E_+ = \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}),$$

making all other tracks full except the track $p-1$ that remains empty. So all information about the sets $S_i$ is overwritten. Now the encoded sum $S_{i_1} + S_{i_2}$ on track $d_{i_1} + d_{i_2}$ of $S + S$ can be moved to this empty track by including $p - 1 - (d_{i_1} + d_{i_2})$ in the set $E_+$, so that

$$S + S + \big(E_+ \cup \{p - 1 - (d_{i_1} + d_{i_2})\}\big) = \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}) \cup \tau_{p-1}^p(S_{i_1} + S_{i_2}).$$

Hence, the equality of the sums can be checked by the equations

$$S + S + \big(E_+ \cup \{p - 1 - (d_{i_1} + d_{i_2})\}\big) = S + S + \big(E_+ \cup \{p - 1 - (d_{i_3} + d_{i_4})\}\big).$$

To do this overwriting, the encoding needs to have many numbers in it. A convenient way to have these numbers is to include a block of full tracks in the beginning. Consider a block of $b$ full tracks $\bigcup_{i=0}^{b-1} \tau_n^p(\mathbb{N})$. If $b = \frac{p}{4}$, then

$$\Big(\bigcup_{i=0}^{b-1} \tau_n^p(\mathbb{N})\Big) + \{0, b, 2b, 3b\} = \mathbb{N}$$

and

$$\Big(\bigcup_{i=0}^{b-1} \tau_n^p(\mathbb{N})\Big) + \{0, b, 2b, 3b - 1\} = \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}).$$

Furthermore, the data tracks $d_i$ will be after the track $b$ and their contents cannot be controlled, so they might cause trouble in the sum $S + S$. Because of this, and also in order to have encodings of the sets $S_i = S_i + \{0\}$ in $S + S$, an additional track $d_{m+1}$ containing the encoding of $\{0\}$ is added as the last data track.

It is time to define the encoding. Since adding the $b$ full tracks to $d_{m+1}$ overwrites everything between $d_{m+1}$ and $d_{m+1} + b - 1$, and the encoding of $S_1 + S_1$ should not be vanished, the inequality $d_{m+1} + b < d_1 + d_1$ should hold. This also leaves an empty track between tracks $d_{m+1} + b - 1$ and $d_1 + d_1$, which is also needed later. The earlier recursion equation with the solution $d_i = d_1 + 2^{i-1} - 1$ gives the inequality $d_1 + 2^m - 1 + b < d_1 + d_1$, and further the inequality $2^m + b - 1 < d_1$. So let $d_1 = b + 2^m$.

The number of the last data track $d_{m+1}$ should be smaller than $\frac{p}{2}$ to have its sum with itself smaller than $p$. If $d_{m+1} = \frac{p}{2} - 1$, then it is $p - 1$ modulo $b$ and causes trouble with the overwriting, so let $d_{m+1} = \frac{p}{2} - 2$. Now

$$\frac{p}{2} - 2 = d_{m+1} = d_1 + 2^m - 1 = b + 2^m + 2^m - 1 = \frac{p}{4} + 2^{m+1} - 1$$

implying $p = 2^{m+3} + 4$, $b = 2^{m+1} + 1$ and $d_i = 3 \cdot 2^m + 2^{i-1}$, so all the information needed to define the encoding has been collected:

First, there is the constant map

$$\pi_b(S_1, \ldots, S_m) = B_\pi = \bigcup_{i=0}^{b-1} \tau_i^p(\mathbb{N})$$

for the block of full tracks.

Second, there is the mapping

$$\pi_d(S_1, \ldots, S_m) = \Big( \bigcup_{j=1}^{m} \tau_{d_j}^p(S_j) \Big) \cup \{d_{m+1}\}$$

embedding the sets into the data tracks.

Third, the encoding $\pi$ is the union of these two

$$\pi(S_1, \ldots, S_m) = \Big( \bigcup_{i=0}^{b-1} \tau_i^p(\mathbb{N}) \Big) \cup \Big( \bigcup_{j=1}^{m} \tau_{d_j}^p(S_j) \Big) \cup \{d_{m+1}\},$$

so $\pi = \pi_b \cup \pi_d$.

**Example 2.2.1.** *Let $\mathfrak{S} \in \mathfrak{S}[\mathcal{X}_2, \mathcal{P}_{\mathrm{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +)]$ be a system using two variables, and union and addition as operations.*

*Now $p = 36$, $b = 9$, $d_1 = 13$, $d_2 = 14$ and $d_3 = 16$. So the two components of the encoding are*

$$\pi_b(S_1, S_2) = B_\pi = \bigcup_{i=0}^{8} \tau_i^p(\mathbb{N}),$$

*containing all numbers equivalent to $i$ modulo 36 for $0 \leqslant i \leqslant 8$, and*

$$\pi_d(S_1, S_2) = \tau_{13}^{36}(S_1) \cup \tau_{14}^{36}(S_2) \cup \{16\},$$

*containing the encoding of $S_1$ on track 13, the encoding of $S_2$ on track 14 and the encoding of $\{0\}$ on track 16. The encoding $\pi(S_1, S_2)$ is the union of these two.*

*The sum of $\pi_b(S_1, S_2)$ with itself results in a set that has all numbers equivalent to $i$ modulo 36 for $0 \leqslant i \leqslant 16$ :*

$$\pi_b(S_1, S_2) + \pi_b(S_1, S_2) = \bigcup_{i=0}^{16} \tau_i^{36}(\mathbb{N}).$$

*The sum of $\pi_d(S_1, S_2)$ with itself results in a set that has the encoding of $S_1 + S_1$ on track 26, the encoding of $S_1 + S_2$ on track 27, the encoding of*

*$S_2 + S_2$ on track 28, the encoding of $S_1$ on track 29, the encoding of $S_2$ on track 30 and the encoding of $\{0\}$ on track 32 :*

$$\pi_d(S_1, S_2) + \pi_d(S_1, S_2) =$$
$$\tau_{26}^{36}(S_1 + S_1) \cup \tau_{27}^{36}(S_1 + S_2) \cup \tau_{28}^{36}(S_2 + S_2) \cup \tau_{29}^{36}(S_1) \cup \tau_{30}^{36}(S_2) \cup \{32\}.$$

*The last component in the sum $\pi(S_1, S_2) + \pi(S_1, S_2)$ is*

$$\pi_b(S_1, S_2) + \pi_d(S_1, S_2) = \big(\bigcup_{i=0}^{8} \tau_i^p(\mathbb{N})\big) + \big(\tau_{13}^{36}(S_1) \cup \tau_{14}^{36}(S_2) \cup \{16\}\big),$$

*which equals*

$$\tau_{13}^{36}(\min(S_1) + \mathbb{N}) \cup \tau_{14}^{36}(\min(S_2) + \mathbb{N}) \cup \tau_{15}^{36}(\min(S_1, S_2) + \mathbb{N}) \cup \bigcup_{i=0}^{8} \tau_{16+i}^{p}(\mathbb{N}),$$

*where $\min(S_i)$ should be considered undefined if $S_i$ is empty and in this case $\min(S_i) + \mathbb{N} = \varnothing$. So the sum of $\pi(S_1, S_2)$ with itself equals*

$$\bigcup_{i=0}^{24} \tau_i^{36}(\mathbb{N}) \cup \tau_{26}^{36}(S_1 + S_1) \cup \tau_{27}^{36}(S_1 + S_2) \cup \tau_{28}^{36}(S_2 + S_2) \cup \tau_{29}^{36}(S_1) \cup \tau_{30}^{36}(S_2) \cup \{32\}.$$

*If the set $\{0, b+1\} = \{0, 10\}$ is added to $\pi(S_1, S_2) + \pi(S_1, S_2)$, then the sum of $\{0, b+1\}$ and $\bigcup_{i=0}^{24} \tau_i^{36}(\mathbb{N})$ equals $\bigcup_{i=0}^{34} \tau_i^{36}(\mathbb{N})$. The sums with the encoded sets on tracks 26, 27, 28, 29, 30 and 32 end up on the same tracks when added to 0 and on tracks 36, 37, 38, 39, 40 and 42 when added to 10. The most important thing is that they do not end up on the track $p - 1 = 35$, which remains empty in the sum $\pi(S_1, S_2) + \pi(S_1, S_2) + \{0, 10\}$ while all other tracks are full, so that*

$$\pi(S_1, S_2) + \pi(S_1, S_2) + \{0, 10\} = \bigcup_{i=0}^{34} \tau_i^{36}(\mathbb{N}).$$

□

The overwriting of data in $\pi(S_1, \ldots, S_m)$ and $\pi(S_1, \ldots, S_m) + \pi(S_1, \ldots, S_m)$ can be made according to the following Lemma:

**Lemma 2.2.1.** *Let $S = \pi(S_1, \ldots, S_m)$ for some $S_1, \ldots, S_m \subseteq \mathbb{N}$. Then*

$$S + \{0, b, 2b, 3b - 1\} = S + S + \{0, b + 1\} = \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}).$$

*Proof.* First consider the sum $S + \{0, b, 2b, 3b-1\}$. As previously mentioned,

$$B_\pi + \{0, b, 2b, 3b-1\} = \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N})$$

and only the emptiness of track $p-1$ is left to prove. The sums of $d_i$ with 0, $b$ and $2b$ are all less than

$$2b + d_{m+1} = \frac{p}{2} + \frac{p}{4} - 2 = \frac{3p}{4} - 2 < p - 1$$

and sums with $3b-1$ are all greater than

$$3b - 1 + d_1 = 3b - 1 + b + 2^m = 4b + 2^m > p$$

and smaller than

$$3b - 1 + d_{m+1} = \frac{3p}{4} + \frac{p}{2} - 2 = \frac{5p}{4} - 2 < 2p - 1,$$

so the data tracks added to $\{0, b, 2b, 3b-1\}$ do not end up on track $p-1$.

Then consider $S + S + \{0, b+1\}$. Since $S = B_\pi \cup \pi_d(S_1, \ldots, S_m)$ and the addition is commutative, the sum $S + S$ can be written as

$$\big(B_\pi + B_\pi\big) \cup \big(B_\pi + \pi_d(S_1, \ldots, S_m)\big) \cup \big(\pi_d(S_1, \ldots, S_m) + \pi_d(S_1, \ldots, S_m)\big).$$

The sum $B_\pi + B_\pi$ equals

$$\bigcup_{n=0}^{2b-2} \tau_n^p(\mathbb{N})$$

and adding 0 and $b+1$ to this set results in the set

$$\bigcup_{n=0}^{3b-1} \tau_n^p(\mathbb{N}).$$

It is left to prove that the tracks from $3b = \frac{3p}{4}$ to $p-2$ are full and that the track $p-1$ is empty. Adding $B_\pi$ to $d_{m+1}$ results in a block of full tracks from $d_{m+1}$ to $d_{m+1} + b - 1$, that is, from $\frac{p}{2} - 2$ to $\frac{3p}{4} - 3$. Now

$$d_{m+1} + b - 1 + b + 1 = d_{m+1} + 2b = \frac{p}{2} - 2 + \frac{p}{2} = p - 2$$

and the rest of the block $B_\pi + d_{m+1}$ ends up on tracks from $d_{m+1} + b + 1 = \frac{3p}{4} - 1$ to $p-3$, so all tracks from 0 to $p-2$ are full.

Furthermore,

$$d_1 + d_1 + b + 1 = b + 2^m + b + 2^m + b + 1 = 3b + 2^{m+1} + 1 = 3b + b - 1 + 1 = 4b = p$$

and

$$d_{m+1} + d_{m+1} + b + 1 = \frac{p}{2} - 2 + \frac{p}{2} - 2 + \frac{p}{4} + 1 = \frac{5p}{4} - 3 < 2p - 1,$$

so none of the sets encoded on tracks $d_i + d_j$ end up on track $p - 1$ and it remains empty. □

For the encoding to be usable, there needs to be a set of equations guaranteeing that a set is an encoding of some $m$ sets. This set of equations is given in the next Lemma:

**Lemma 2.2.2.** *Let $S \in \mathbb{N}$. Then $S = \pi(S_1, \ldots, S_m)$ for some $S_1, \ldots, S_m \subseteq \mathbb{N}$ if and only if $S$ is the solution to the equations*

$$\begin{aligned}
X + \{0, b, 2b, 3b - 1\} &= \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}) \\
X + E_\varnothing &= \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}) \\
X + \big(\{0, b, 2b, 3b - 1\} \cup \{p - 1 - d_{m+1}\}\big) &= \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}) \cup \{p - 1\} \\
X + \big(\{0, b, 2b, 3b - 1\} \cup \{p - 1\}\big) &= \mathbb{N} \\
X + \big(\{0, b, 2b, 3b - 1\} \cup \{p - 2\}\big) &= \mathbb{N} \\
&\vdots \\
X + \big(\{0, b, 2b, 3b - 1\} \cup \{p - b\}\big) &= \mathbb{N},
\end{aligned}$$

*where* $E_\varnothing = \{0, b, 2b, 3b - 1\} \cup \Big(\bigcup_{k=b}^{p-1}\{p - 1 - k\} \setminus \bigcup_{i=1}^{m+1}\{p - 1 - d_i\}\Big)$

*Proof.* The first equation states that

$$S + \{0, b, 2b, 3b - 1\} = \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}).$$

This is true for all $S = \pi(S_1, \ldots, S_m)$ for some $S_1, \ldots, S_m \subseteq \mathbb{N}$ by Lemma 2.2.1.

Assuming that the first equation holds, the expression

$$S + \Big(\{0, b, 2b, 3b - 1\} \cup \Big(\bigcup_{k=b}^{p-1}\{p - 1 - k\} \setminus \bigcup_{i=1}^{m+1}\{p - 1 - d_i\}\Big)\Big)$$

equals

$$\bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}) \cup \Big(S + \Big(\bigcup_{k=b}^{p-1}\{p - 1 - k\} \setminus \bigcup_{i=1}^{m+1}\{p - 1 - d_i\}\Big)\Big),$$

which implies that

$$\Big(S + \Big(\bigcup_{k=b}^{p-1}\{p-1-k\} \setminus \bigcup_{i=1}^{m+1}\{p-1-d_i\}\Big)\Big) \cap \tau_{p-1}^{p}(\mathbb{N}) = \varnothing$$

if and only if the second equation holds. So, the first and the second equations together imply that the tracks in

$$\{b,\ldots,p-1\} \setminus \bigcup_{i=1}^{m+1}\{d_i\}$$

are empty, which are exactly the always empty tracks in all $S = \pi(S_1,\ldots,S_m)$ for some $S_1,\ldots,S_m \subseteq \mathbb{N}$.

The third equation, assuming the first holds, states that the track $d_{m+1}$ contains the encoding of $\{0\}$, which is true for all $S = \pi(S_1,\ldots,S_m)$ for some $S_1,\ldots,S_m \subseteq \mathbb{N}$ by definition.

Finally, still assuming the first equation holds, the equations

$$\begin{aligned} S + \big(\{0,b,2b,3b-1\} \cup \{p-1\}\big) &= \mathbb{N} \\ S + \big(\{0,b,2b,3b-1\} \cup \{p-2\}\big) &= \mathbb{N} \\ &\vdots \\ S + \big(\{0,b,2b,3b-1\} \cup \{p-b\}\big) &= \mathbb{N}, \end{aligned}$$

state that the tracks $0,\ldots,b-1$ are full, and these are exactly the full tracks in the definition of $\pi$.

It follows that $S$ is a solution to the equations if and only if $S = \pi(S_1,\ldots,S_m)$ for some $S_i \subseteq \mathbb{N}$. □

The expressions appearing in the system $\mathfrak{S}$ to be simulated are assumed to be of the form

$$\varphi(X_1,\ldots,X_m) = \Big(\bigcup_{(i_1,i_2)\in I_\varphi} X_{i_1} + X_{i_2}\Big) \cup \Big(\bigcup_{j\in J_\varphi} X_j\Big) \cup F_\varphi,$$

where $I_\varphi \subseteq \{1,\ldots,m\} \times \{1,\ldots,m\}$, $J_\varphi \subseteq \{1,\ldots,m\}$ and $F_\varphi \in \mathcal{P}_{\mathrm{evp}}(\mathbb{N})$. This is not a serious limitation, as any system can be transformed into that form by introducing new variables.

For these expressions, the sets

$$\begin{aligned} E_\varphi &= \{0, b+1\} \cup \{p-1-(d_{i_1}+d_{i_2}) \mid (i_1,i_2) \in I_\varphi\} \cup \\ &\cup \{p-1-(d_{m+1}+d_j) \mid j \in J_\varphi\} \cup \tau_{p-1-2d_{m+1}}^{p}(F_\varphi) \end{aligned}$$

will be used to simulate them in the one-variable system:

**Lemma 2.2.3.** *Let $S_1, \ldots, S_m \subseteq \mathbb{N}$ and $S = \pi(S_1, \ldots, S_m)$. Then*

$$S + S + E_\varphi = \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}) \cup \tau_{p-1}^p(\varphi(S_1, \ldots, S_m)).$$

*Proof.* The sum of $S + S$ with $\{0, b+1\}$ makes all tracks except $p-1$ full by Lemma 2.2.1. Adding

$$\{p-1-(d_{i_1}+d_{i_2}) \mid (i_1, i_2) \in I_\varphi\} \cup \{p-1-(d_{m+1}+d_j) \mid j \in J_\varphi\} \cup \tau_{p-1-d_{m+1}}^p(F_\varphi)$$

to $S+S$ takes the sets $S_{i_1}+S_{i_2}$ for $(i_1, i_2) \in I_\varphi$, the sets $S_j$ for $j \in J_\varphi$ and the set $F_\varphi$ to the track $p-1$, so it contains the encoding of $\varphi(S_1, \ldots, S_m)$. □

**Example 2.2.2** (Example 2.2.1 continued)**.** *Let $\mathfrak{S}$ contain the equation*

$$\varphi(X_1, X_2) = \psi(X_1, X_2),$$

*where*

$$\varphi(X_1, X_2) = (X_1 + X_2) \cup X_1 \cup \{2n \mid n \in \mathbb{N}\}$$

*and*

$$\psi(X_1, X_2) = (X_1 + X_1) \cup (X_2 + X_2) \cup \{3n \mid n \in \mathbb{N}\}.$$

*Now $p - 1 = 35$ and subtracting $d_1 + d_2 = 27$, $d_1 + d_3 = 29$ and $2d_3 = 32$ from it results in 8, 6 and 3, respectively. Thus*

$$E_\varphi = \{0, 10\} \cup \{8\} \cup \{6\} \cup \tau_3^{36}(\{2n \mid n \in \mathbb{N}\})$$

*and*

$$\pi(S_1, S_2) + \pi(S_1, S_2) + E_\varphi = \bigcup_{i=0}^{34} \tau_i^{36}(\mathbb{N}) \cup \tau_{35}^{36}(\varphi(S_1, S_2)).$$

*Similarly, $d_1 + d_1 = 26$ and $d_2 + d_2 = 28$ so*

$$E_\psi = \{0, 10\} \cup \{7, 9\} \cup \tau_3^{34}(\{3n \mid n \in \mathbb{N}\})$$

*and*

$$\pi(S_1, S_2) + \pi(S_1, S_2) + E_\psi = \bigcup_{i=0}^{34} \tau_i^{36}(\mathbb{N}) \cup \tau_{35}^{36}(\psi(S_1, S_2)).$$

*The sets $\pi(S_1, S_2)+\pi(S_1, S_2)+E_\varphi$ and $\pi(S_1, S_2)+\pi(S_1, S_2)+E_\psi$ are equal on tracks from 0 to 34, and their equality depends only on equality on track 35. Since the contents of track 35 are the encodings of $\varphi(S_1, S_2)$ and $\psi(S_1, S_2)$, the sets are equal if and only if $\varphi(S_1, S_2)$ and $\psi(S_1, S_2)$ are equal as well.*

*Hence, $(S_1, S_2)$ is a solution to $\varphi(X_1, X_2) = \psi(X_1, X_2)$ if and only if $\pi(S_1, S_2)$ is a solution to $X + X + E_\varphi = X + X + E_\psi$.* □

As the expressions can be isolated, and thus compared, on track $p-1$ by the previous Lemma, it is easy to construct an equivalent equation with only one variable for the equation $\varphi = \psi$:

**Lemma 2.2.4.** *Let $S_1, \ldots, S_m \subseteq \mathbb{N}$. Then $S = \pi(S_1, \ldots, S_m)$ is a solution to the equation*

$$X + X + E_\varphi = X + X + E_\psi$$

*if and only if $\varphi(S_1, \ldots, S_m) = \psi(S_1, \ldots, S_m)$.*

*Proof.* By Lemma 2.2.3,

$$S + S + E_\varphi = \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}) \cup \tau_{p-1}^p(\varphi(S_1, \ldots, S_m))$$

and

$$S + S + E_\psi = \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}) \cup \tau_{p-1}^p(\psi(S_1, \ldots, S_m)).$$

These sets are the same if and only if $\varphi(S_1, \ldots, S_m) = \psi(S_1, \ldots, S_m)$, which proves the statement. □

Now the original system can be simulated by a system containing equations given by Lemma 2.2.2, which ensure the validity of the encoding. It also contains equations given by Lemma 2.2.4, equivalent to the equations of the original system.

**Lemma 2.2.5.** *Let $\mathfrak{S} \in \mathfrak{S}[\mathcal{X}_m, \mathcal{P}_{\mathrm{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +)]$ be a system of equations with all expressions appearing in the equations of the form*

$$\varphi(X_1, \ldots, X_m) = \Big( \bigcup_{(i_1,i_2)\in I_\varphi} X_{i_1} + X_{i_2} \Big) \cup \Big( \bigcup_{j \in J_\varphi} X_j \Big) \cup F_\varphi,$$

*where $I_\varphi \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$, $J_\varphi \subseteq \{1, \ldots, m\}$ and $F_\varphi \in \mathcal{P}_{\mathrm{evp}}(\mathbb{N})$.*

*If the system $\mathfrak{S}' \in \mathfrak{S}[\mathcal{X}_1, \mathcal{P}_{\mathrm{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), +)]$ contains the equations in Lemma 2.2.2 and an equation*

$$X + X + E_\varphi = X + X + E_\psi$$

*for every equation $\varphi = \psi \in \mathfrak{S}$, then $S \subseteq \mathbb{N}$ is a solution to $\mathfrak{S}'$ if and only if $S = \pi(S_1, \ldots, S_m)$ for some solution $(S_1, \ldots, S_m) \in \mathcal{P}(\mathbb{N})^m$ of $\mathfrak{S}$.* □

This contruction can be taken one step further. The constructed one-variable system can be simulated by a system with only two equations yielding the main result of this chapter or even the whole thesis:

**Theorem 2.2.1** ([24]). *Let $S \subseteq \mathbb{N}$ be recursive. Then there are natural numbers $p, d \in \mathbb{N}$ and eventually periodic sets $A, B, C, D \subseteq \mathbb{N}$, such that a natural number $n \in \mathbb{N}$ belongs to $S$ if and only if $pn+d$ belongs to the unique solution to*

$$\begin{aligned} X + A &= B \\ X + X + C &= X + X + D. \end{aligned}$$

*Proof.* By Proposition 2.1.1 there exists a system in $\mathfrak{S}[\mathcal{X}_m, \mathcal{P}_{\text{fin}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +)]$ with a unique solution $(S_1, S_2, \ldots, S_m)$, where $S_1 = S$. Without loss of generality, it can be assumed that all expressions appearing in the equations of the system are of the form

$$\varphi(X_1, \ldots, X_m) = \Big( \bigcup_{(i_1,i_2)\in I_\varphi} X_{i_1} + X_{i_2} \Big) \cup \Big( \bigcup_{j \in J_\varphi} X_j \Big) \cup F_\varphi,$$

where $I_\varphi \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$, $J_\varphi \subseteq \{1, \ldots, m\}$ and $F_\varphi \in \mathcal{P}_{\text{evp}}(\mathbb{N})$.

Lemma 2.2.5 gives a system in $\mathfrak{S}[\mathcal{X}_1, \mathcal{P}_{\text{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), +)]$ with a unique solution $\pi(S_1, \ldots, S_m)$. This unique solution has the property that

$$(2^{m+2} + 8)n + 2^m + 3 \in \pi(S_1, \ldots, S_m) \text{ if and only if } n \in S.$$

The constructed system has equations

$$X + A_i = B_i \text{ for } i = 1, \ldots, k_1$$

and

$$X + X + C_i = X + X + D_i \text{ for } i = 1, \ldots, k_2$$

as presented in Lemma 2.2.5.

Let $p = \max(k_1, k_2) + 1$ and define the sets

$$\begin{aligned} A &= \bigcup_{i=1}^{k_1} \tau_i^p(A_i) \\ B &= \bigcup_{i=1}^{k_1} \tau_i^p(B_i) \\ C &= \bigcup_{i=1}^{k_2} \tau_i^p(C_i) \\ D &= \bigcup_{i=1}^{k_2} \tau_i^p(D_i) \end{aligned}$$

Note that all of the constant sets $A_i, B_i, C_i, D_i$ introduced during the construction have zeros in them, and especially, they are all non-empty.

Now consider the system of equations

$$\begin{aligned} X + A &= B \\ X + X + C &= X + X + D. \end{aligned}$$

If it has a solution $T \subseteq \mathbb{N}$, then $T$ can only have numbers 0 modulo $p$ in it. Assume the contrary: there is a number $pn + d \in T$ for some $0 < d < p$.

If $p - d \leqslant k_1$, then the set $A_{p-d}$ contains 0 and the set $T + \tau^p_{p-d}(A_{p-d})$ contains $pn + d + p - d = p(n + 1)$. But the track 0 of $B$ is empty, so it is a contradiction.

If $p-d > k_1$, then the set $A_{k_1}$ contains 0 and the set $T+\tau^p_{k_1}(A_{k_1})$ contains $pn + d + k_1$. But $k_1 < d + k_1 < p$ and all tracks from $k_1 + 1$ to $p - 1$ are empty in $B$, which is again a contradiction and $T = \tau^p_0(T')$ for some $T' \subseteq \mathbb{N}$.

Now

$$\begin{aligned} \tau^p_0(T') + A &= \bigcup_{i=1}^{k_1} \tau^p_i(T' + A_i) \\ \tau^p_0(T') + \tau^p_0(T') + C &= \bigcup_{i=1}^{k_2} \tau^p_i(T' + T' + C_i) \\ \tau^p_0(T') + \tau^p_0(T') + D &= \bigcup_{i=1}^{k_2} \tau^p_i(T' + T' + D_i) \end{aligned}$$

and $T$ is a solution to

$$\begin{aligned} X + A &= B \\ X + X + C &= X + X + D \end{aligned}$$

if and only if

$$\begin{aligned} \bigcup_{i=1}^{k_1} \tau^p_i(T' + A_i) &= \bigcup_{i=1}^{k_1} \tau^p_i(B_i) \\ \bigcup_{i=1}^{k_2} \tau^p_i(T' + T' + C_i) &= \bigcup_{i=1}^{k_2} \tau^p_i(T' + T' + D_i), \end{aligned}$$

but this is the case exactly when $T'$ is a solution to the equations

$$\begin{aligned} X + A_i &= B_i \text{ for } i = 1, \ldots, k_1 \\ X + X + C_i &= X + X + D_i \text{ for } i = 1, \ldots, k_2. \end{aligned}$$

This means that $T' = \pi(S_1, \ldots, S_m)$ and $T = \tau^p_0(\pi(S_1, \ldots, S_m))$.

It follows that

$p\big((2^{m+2}+8)n+2^m+3\big) = p(2^{m+2}+8)n+p(2^m+3) \in T$ if and only if $n \in S$,

and the theorem is proved. □

One example giving insight into why these simple systems are so hard to solve is the following:

**Example 2.2.3.** *In Example 2.1.3 a system* $\mathfrak{S} \in \mathfrak{S}[\mathcal{X}_n, \mathcal{P}_{\text{fin}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +)]$ *that has a solution if and only if the Goldbach's conjecture holds was discussed.*

*By Theorem 2.2.1, there exists such eventually periodic sets* $A$, $B$, $C$ *and* $D$ *that the system of equations over sets of natural numbers*

$$\begin{aligned} X + A &= B \\ X + X + C &= X + X + D \end{aligned}$$

*has a solution if and only if the Goldbach's conjecture holds.* □

As a corollary of Theorem 2.2.1 and Proposition 2.1.2, one has the following theorem giving precise computational hardness of solving problems related to these simple systems:

**Theorem 2.2.2** ([24])**.** *The problem of existence of a solution to the system of equations*

$$\begin{aligned} X + A &= B \\ X + X + C &= X + X + D \end{aligned}$$

*is co-r.e.-complete for given eventually periodic sets* $A, B, C, D \subseteq \mathbb{N}$. □

A similar simulation can be done in the case of more general systems, where also subtraction of sets is allowed. Subtraction of sets of natural numbers is defined elementwise, so that

$$S - T = \{k - l \mid k \in S,\ l \in T,\ k \geqslant l\}$$

for $K, L \subseteq \mathbb{N}$.

The general systems using union, addition and subtraction can present the so-called hyper-arithmetical sets as unique solutions:

**Proposition 2.2.1** (Jeż, Okhotin [16])**.** *Let* $S \subseteq \mathbb{N}$. *Then* $S$ *is hyper-arithmetical if and only if there exists* $n \in \mathbb{N}$ *and a system of equations in* $\mathfrak{S}[\mathcal{X}_n, \mathcal{P}_{\text{fin}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +, -)]$ *with a unique solution* $X_i = S_i$ *such that* $S = S_1$.

And as in the case of addition only, these general systems can be simulated by systems with only one variable and two equations to get the following theorem:

**Theorem 2.2.3** (Lehtinen [21]). *Let $S \subseteq \mathbb{N}$ be hyper-arithmetical. Then there are natural numbers $p, d \in \mathbb{N}$ and eventually periodic sets $A, B, C, D \subseteq \mathbb{N}$, such that a natural number $n \in \mathbb{N}$ belongs to $S$ if and only if $pn+d$ belongs to the unique solution to*

$$\begin{aligned} X + A &= B \\ (X + X) + C &= (X - X) + D. \end{aligned}$$

## 2.3 Limitations of simple systems

As seen in the previous section, simple systems can be suprisingly powerful expressively. However, there are some limitations in their expressive power. It will be proved that a system in $\mathfrak{S}[\mathcal{X}_m, \mathcal{P}_{\mathrm{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), +)]$ cannot have certain sets as the least or greatest solution with respect to componentwise inclusion. These sets are prime and fragile, as defined below.

A set is prime (in the monoid $(\mathcal{P}(\mathbb{N}), +)$), if it cannot be represented as a sum of two sets different from the unit element $\{0\}$.

**Definition 2.3.1.** *A set $S \subseteq \mathbb{N}$ is prime if $S = S_1 + S_2$ implies $S_1 = \{0\}$ or $S_2 = \{0\}$.*

A fragile set is fragile in the sense that adding any set with at least two numbers to it results in a co-finite set, so that its structure breaks and information encoded into it is lost.

**Definition 2.3.2.** *A set $S \subseteq \mathbb{N}$ is fragile if $S + \{n_1, n_2\}$ is co-finite for all $n_1, n_2 \in \mathbb{N}$ with $n_1 \neq n_2$, while $S$ itself is not co-finite.*

A set $S$ is fragile if and only if for every $k > 0$ there exists $k_0$ such that for every pair of numbers larger than $k_0$ and missing from $S$, their difference is bigger than $k$. In other words, $m, n \notin S$ and $k_0 \leqslant m < n$ imply $n - m > k$.

Sets with these two properties cannot be represented by a system in $\mathfrak{S}[\mathcal{X}_m, \mathcal{P}_{\mathrm{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), +)]$, as is stated in the following Theorem.

**Theorem 2.3.1** ([25]). *Let $S \subseteq \mathbb{N}$ be prime and fragile. Then $S$ is not a component of the least or the greatest solution of any system of equations in $\mathfrak{S}[\mathcal{X}_m, \mathcal{P}_{\mathrm{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), +)]$.*

*Proof.* Let $\mathfrak{S} \in \mathfrak{S}[\mathcal{X}_m, \mathcal{P}_{\mathrm{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), +)]$. Assume without loss of generality that all equations in $\mathfrak{S}$ are of the form $X = Y + Z$ or $X = C$ for a constant $C \in \mathcal{P}_{\mathrm{evp}}(\mathbb{N})$.

Consider a solution, in which one of the components is $X_1 = S$. Some other components might have values of the form $S_i = S + \{i\}$ as well: let them be denoted $X_i = S_i$ with $i \in I$ for some finite index set $I \subseteq \mathbb{N}$. Furthermore, let $Y_j = T_j$ with $j \in J$ be the rest of the variables, whose values are not of

the form $S + \{i\}$ for any $i$. It can be assumed that all components of the solution are nonempty.

The smaller solution is defined as follows: Let $k$ be the maximum of the differences of the two smallest numbers in any $V$, with $|V| \geqslant 2$, that appears in the solution. Since $S$ is fragile, there is a number $l$, such that every pair of missing numbers $m, n \notin S$ with $n > m \geqslant l$ satisfies $n - m > k$. Let $l' \geqslant l$ be such that $\{l', l' + 1, \ldots, l' + 2k\} \subseteq S$. Such $l'$ exists by the fragility of $S$.

The new solution is $X_i = S'_i = (S \setminus \{l' + k\}) + \{i\}$ for all $i \in I$ and $Y_j = T_j$ for $j \in J$. As compared to $S_i$, one more number is missing from $S'_i$, but it is different by at least $k+1$ from any other missing number. Thus, the above property is inherited by $S'_i$: every pair $m, n \notin S'_i$ with $n > m \geqslant l + i$ satisfies $n - m > k$.

It has to be shown that the new solution satisfies the equations. Consider every equation in $\mathfrak{S}$. If none of the variables $X_i$ are used in the equation, then the new solution clearly satisfies it, since the variables $Y_j$ did not change their values. So, consider each equation containing an instance of some $X_i$; only three cases are possible:

**Claim 2.3.1.1.** *The equations that contain some $X_i$ are of the form*

1. $X_i = X_{i_1} + Y_j$, *with* $T_j = \{i_2\}$ *and* $i = i_1 + i_2$,
2. $Y_j = X_i + Y_{j_1}$, *with* $|T_{j_1}| \geqslant 2$ *or*
3. $Y_j = X_i + X_{i_1}$.

*Proof.* Let $X_i$ be on the left-hand side of the equation. The right-hand side of this equation cannot be a constant, since $S$ is not ultimately periodic. So, the equation is of the form $X_i = U + V$.

Then $S + \{i\}$ is factorized as $R_1 + R_2$ for some sets $R_1, R_2 \subseteq \mathbb{N}$. Let $i_1$ and $i_2$ be the smallest elements of $R_1$ and $R_2$, respectively. Then $i_1 + i_2$ is the smallest element of $S + \{i\}$, which must be $i$, as $0 \in S$ by the primality of $S$. Therefore, $S$ is factorized as $(R_1 - i_1) + (R_2 - i_2)$, and since $S$ is prime, $R_2 - i_2 = \{0\}$, that is, $R_2 = \{i_2\}$ and $R_1 = S + \{i_1\} = S_{i_1}$. Then the equation is of the form $X_i = X_{i_1} + Y_j$, where $T_j = \{i_2\}$.

If the equation is $Y_j = X_i + V$, and if $V = Y_{j_1}$, then it must be the case that $|T_{j_1}| \geqslant 2$: otherwise, $T_{j_1} = \{i_1\}$ for some $i_1 \in \mathbb{N}$ and $T_j = S_i + \{i_1\} = S_{i+i_1}$, which contradicts the assumption that $T_j \notin \{S_i \mid i \in I\}$.

The only case left is $Y_j = X_i + X_{i_1}$. □

Resuming the proof of the theorem, it has been shown that there are three cases to consider:

1. Let $X_i$ be on the left-hand side of the equation, so that it is of the form $X_i = X_{i_1} + Y_j$. Then the solution is $S_i = S_{i_1} + T_j$ with $T_j = \{i_2\}$ and

$i = i_1 + i_2$. The equality $S'_i = S'_{i_1} + \{i_2\}$ holds by definition of the sets $S'_i$.

2. Let the equation be of the form $Y_j = X_i + Y_{j_1}$, with $|T_{j_1}| \geqslant 2$. The goal is to show that $S'_i + T_{j_1} = S_i + T_{j_1}$. Clearly, $S'_i + T_{j_1} \subseteq S_i + T_{j_1}$. Consider any number $m+n$ with $m \in S_i$ and $n \in T_{j_1}$. If $m \neq i+l'+k$, then $m \in S'_i$ by the definition of $S'_i$ and thus $m + n \in S'_i + T_{j_1}$. Let $m = i + l' + k$ and let $n_1 < n_2$ be the two smallest numbers in $T_{j_1}$. Then $n_2 - n_1 \leqslant k$, by the definition of $k$.

   Consider two numbers $m + n - n_1$ and $m + n - n_2$. The former is clearly greater than $l + i$, since $n_1 \leqslant n$. For the latter number, note that $n-n_2 \geqslant n-n_1-k$ and thus $m+n-n_2 \geqslant (i+l'+k)+(n-n_1-k) = i + l' + n - n_1 \geqslant l + i$. Since both numbers are greater or equal to $l + i$ and their difference is at most $k$, it could not be the case that both of them are missing from $S'_i$. Therefore, either $(m+n-n_1)+n_1 \in S'_i+T_{j_1}$ or $(m + n - n_2) + n_2 \in S'_i + T_j$, which proves that $m + n$ is in $S'_i + T_{j_1}$.

3. Finally, consider the case $Y_j = X_i + X_{i_1}$. The solution is $T_j = S_i + S_{i_1}$. It is to be shown that $T_j = S'_i + S'_{i_1}$. Obviously, $S'_i + S'_{i_1} \subseteq S_i + S_{i_1}$. To prove the converse, assume that $m \in S_i$ and $n \in S_{i_1}$. If $m \in S'_i$, then the argument used in the previous case applies, with $S'_{i_1}$ instead of $T_{j_1}$. The case of $n \in S'_{i_1}$ is handled symmetrically.

   Suppose $m \notin S'_i$ and $n \notin S'_{i_1}$. Then $m = i + l' + k$ and $n = i_1 + l' + k$. Now $m-1 \in S'_i$ and $n+1 \in S'_{i_1}$, so $m+n = (m-1)+(n+1) \in S'_i+S'_{i_1}$. Thus $T_j = S'_i + S'_{i_1}$.

It follows that the system of equations is satisfied, and $S$ is not a component of the least solution.

It is left to prove that a prime and fragile set cannot be a component of the greatest solution. As in the proof for the least solution, let the system of equations have a solution $X_1 = S$, $X_i = S_i$ with $i \in I$, $Y_j = T_j$ with $j \in J$, where $S_i = S + \{i\}$ and $T_j \neq S + \{i\}$.

For every equation of the form $Y_j = X_i + V$, where $V$ is either a variable $X_j$ or a variable $Y_j$ with $|T_j| \geqslant 2$, the sum $S_i + V$ is co-finite because $S_i$ is fragile. Let $k$ be the least number with $l \in S_i + V$ for all $l \geqslant k$ and let $k_0$ be the maximum of all $k$'s for all such equations. Let $n_0 \geqslant k_0$ and $n_0 \notin S$.

In the new solution this number $n_0$ is included into $S$: $X_i = S'_i = (S \cup \{n_0\}) + \{i\} = S_i \cup \{n_0 + i\}$ and $Y_j = T_j$ for all applicable $i$ and $j$.

To see that this assignment is a solution, only equations where some $X_i$ occur need to be considered, since the rest stay as they were. Again, see Claim 2.3.1.1, there are three cases to consider:

1. For an equation $X_i = X_{i_1} + Y_j$ the solution is $S_i = S_{i_1} + T_j$ with $T_j = \{i_2\}$ and $i = i_1 + i_2$. The equation is satisfied for the new solution as

well: $(S \cup \{n_0\}) + \{i\} = (S \cup \{n_0\}) + \{i_1 + i_2\} = ((S \cup \{n_0\}) + \{i_1\}) + \{i_2\}$.

2. In the case $Y_j = X_i + Y_{j_1}$ with $|T_{j_1}| \geqslant 2$, note that since $n_0 \geqslant k_0$, every number greater or equal to $n_0$ is in $T_j$ by the choice of $k_0$. Then $S'_i + T_{j_1}$ equals

$$(S_i \cup \{n_0 + i\}) + T_{j_1} = (S_i + T_{j_1}) \cup (\{n_0 + i\} + T_{j_1}) = T_j \cup \underbrace{(\{n_0 + i\} + T_{j_1})}_{\subseteq T_j} = T_j.$$

3. In the final case, the equation is $Y_j = X_i + X_{i_1}$. Again, $l \in T_j$ for every $l \geqslant n_0$, and $S'_i + S'_{i_1}$ can be transformed as

$$\begin{aligned} & (S_i \cup \{n + i\}) + (S_{i_1} \cup \{n_0 + i_1\}) = \\ = \; & \underbrace{(S_i + S_{i_1})}_{=T_j} \cup \underbrace{(S_i + \{n_0 + i_1\})}_{\subseteq T_j} \cup \underbrace{(S_{i_1} + \{n_0 + i_1\})}_{\subseteq T_j} \cup \underbrace{(\{n_0 + i\} + \{n_0 + i_1\})}_{\subseteq T_j}, \end{aligned}$$

which equals $T_j$.

□

In the remaining part of this section, a prime and fragile subset of natural numbers is constructed. First, some concepts that are used in the construction are defined.

For a set $S \subseteq \mathbb{N}$ the infinite word $w(S) = x_0 x_1 x_2 \cdots \in \{0,1\}^\omega$, where

$$x_k = \begin{cases} 1, & \text{if } k \in X \\ 0, & \text{if } k \notin X, \end{cases}$$

is called the *characteristic word* of $S$.

The relation $\preccurlyeq$ between infinite words over $\{0,1\}$ defined by

$$x_0 x_1 x_2 \ldots \preccurlyeq y_0 y_1 y_2 \ldots, \text{ iff } x_k \leqslant y_k \text{ for all } k \in \mathbb{N}$$

is a partial order. It corresponds to set inclusion over subsets of $\mathbb{N}$ in the sense that $w(X) \preccurlyeq w(Y)$ if and only if $X \subseteq Y$ for $X, Y \subseteq \mathbb{N}$. A similar relation is defined for finite words of matching length as $x_1 \ldots x_n \preccurlyeq y_1 \ldots y_n$ if $x_i \leqslant y_i$ for each $i$.

A finite word $w$ is a $\preccurlyeq$-*factor* of a word $v \in \{0,1\}^* \cup \{0,1\}^\omega$, if there are words $x$, $w'$ and $y$, such that $v = xw'y$, $|w'| = |w|$ and $w \preccurlyeq w'$.

Let $u = 100011110000$ and $v_k = (1^k 0)^{2^k}$ for all $k \geqslant 2$, and consider a set $S$ defined by the characteristic word

$$w(S) = s = s_0 s_1 s_2 \cdots = u v_2 v_3 v_4 \cdots = 100011110000 \prod_{k=2}^{\infty} (1^k 0)^{2^k}.$$

It will now be proved that this set has the desired properties.

**Lemma 2.3.1.** *The set $S$ is fragile and prime.*

*Proof.* The first claim is that the length of each prefix $uv_2 \dots v_n$ of $s$ is

$$|uv_2 \dots v_{n-1}| = (n-1)2^n + 8. \tag{2.1}$$

To prove this, consider the equality

$$\sum_{k=0}^{n-1} (k+1)t^k = \frac{((t-1)n-1)t^n + 1}{(t-1)^2}, \tag{2.2}$$

which can be obtained by taking the derivative of both sides of the known equality $\sum_{k=0}^{n} t^k = \frac{t^{n+1}-1}{t-1}$. Substituting $t = 2$ in (2.2) yields

$$|uv_2 \cdots v_{n-1}| = 12 + \sum_{k=2}^{n-1} (k+1)2^k = 12 + (n-1)2^n + 1 - 5 = (n-1)2^n + 8,$$

which proves (2.1).

Getting back to the proof of the Lemma, the fragility of $S$ is obvious because starting from $v_k$, the distance between any two zeroes in $w(S)$ is at least $k+1$.

To prove the primality of $S$, suppose that $S = X + Y$ for some $X, Y \subseteq \mathbb{N}$. Let $w(X) = x = x_0x_1x_2 \dots$ and $w(Y) = y = y_0y_1y_2 \dots$ be the characteristic words of $X$ and $Y$.

Since $S = X + Y$, it holds that $X + \{k\} \subseteq S$ for all $k \in Y$. This is equivalent to $0^k x \preccurlyeq s$. The characteristic word for $S$ has 10001 as a prefix, so 0 and 4 are the two smallest numbers in $S$. One of the sets, $X$ or $Y$, must contain them both. Let it be $X$, so that $x$ begins with 10001.

If the factorization $S = X + Y$ is nontrivial, then there is the smallest nonzero number $m \in Y$. Then $m, m+4 \in S$, and it is easy to see that $m \geqslant 12$: indeed, by the form of $u$, there is no pair $s_i = s_{i+4} = 1$ for $1 \leqslant i \leqslant 11$. Consequently, $u$ is a prefix of $x$.

Now $x_i = s_i$ for $i < m$ and $0^m x = 0^m u \cdots \preccurlyeq s$. Since $u = 100011110000$, we have $s_{m+4} = s_{m+5} = s_{m+6} = s_{m+7} = 1$. Then

$$m \geqslant 52 = |uv_2v_3| - 4, \tag{2.3}$$

because $v_4$ is the first of the words $v_k$ to have 1111 as a $\preccurlyeq$-factor. Let the first $m$ symbols of $x$ be

$$uv_2 \cdots v_n v,$$

where $n \geqslant 2$ and $v_{n+1} = vv'$. It follows that $uv_2 \cdots v_n$ is a $\preccurlyeq$-factor of $v'v_{n+2}$, since $|uv_2 \cdots v_n| = n \cdot 2^{n+1} + 8 < (n+3) \cdot 2^{n+2} = |v_{n+2}|$ by (2.1). In particular, there is a factor $w$ of $v'v_{n+2}$ with $|w| = |v_n|$ and $v_n \preccurlyeq w$. Since the distance between consecutive occurences of zero in $v'v_{n+2}$ is at most $n+3$,

and $|w| = (n+1) \cdot 2^n > 2(n+2)+1$, it follows that $w$ contains at least two occurrences of zero. If $w$ has a zero in some position, then $v_n$ has to have a zero in the same position. Since the distance between consecutive zeroes is $n+1$ in $v_n$ and $n+2$ or $n+3$ in $w$, a contradiction is obtained. It follows that $Y = \{0\}$ and $S$ is prime. □

It follows by Theorem 2.3.1 that this set $S$ is not representable by systems of equations with eventually periodic constants and the operation of addition. Since $S$ is obviously recursive, and in fact computationally very easy, these equations are less powerful than the equations equipped with addition and union.

Similarly, one-variable systems cannot have fragile sets as components of least or greatest solutions. The proof is omitted, as it is just a simpler version of the proof of Theorem 2.3.1.

**Theorem 2.3.2** ([24]). *Let $S \subseteq \mathbb{N}$ be fragile. Then $S$ is not the least or greatest solution of any system of equations in $\mathfrak{S}[\mathcal{X}_1, \mathcal{P}_{\mathrm{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), +)]$.* □

A fragile set can be represented by a system in $\mathfrak{S}[\mathcal{X}_m, \mathcal{P}_{\mathrm{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), \cup, +)]$, with this set as a component of a unique solution. This system can be simulated by a system in $\mathfrak{S}[\mathcal{X}_1, \mathcal{P}_{\mathrm{evp}}(\mathbb{N}), (\mathcal{P}(\mathbb{N}), +)]$ as presented in the previous section. Say the solution is $(S_1, \ldots, S_m)$ and $S_1$ is the fragile set. Then the set

$$\pi(S_1, \ldots, S_m) + \pi(S_1, \ldots, S_m) + E_\varphi = \bigcup_{n=0}^{p-2} \tau_n^p(\mathbb{N}) \cup \tau_{p-1}^p(S_1) \text{ (cf. Lemma 2.2.3)}$$

for the expression $\varphi = X_1$ is fragile, and adding a variable $Y$ and an equation $Y = X + X + E_\varphi$ to the one-variable system with the solution $X = \pi(S_1, \ldots, S_m)$ results in a system with two variables and a fragile set as a component of the unique solution. Thus, one-variable systems with addition only are less powerful than many-variable systems with addition only. Table 2.1 gathers what is known about the expressive power of systems over sets of natural numbers.

In the table, $\Sigma_1^1$ is the class of sets that can be defined by using existential quantification of second order before any first-order sentence. A dual notion is $\Pi_1^1$, where only the use of universal second-order quantification is allowed. The hyper-arithmetical sets are in the intersection of these classes, in other words $\Delta_1^1 = \Sigma_1^1 \cap \Pi_1^1$.

The encodings of different classes mentioned in the table mean that the whole class of the corresponding sets are not representable. They can be similar to the one discussed in Section 2.2, implying that the number of variables can be limited to one. There might, depending on the meaning of similar, exist other kinds of encodings for representing sets, but the results in this section show that all sets in the considered classes are not representable.

| Operations | Unique | Least | Greatest |
|---|---|---|---|
| $\cup, +$ [13] | $\Delta_1^0$ (recursive) | $\Sigma_1^0$ (r.e.) | $\Pi_1^0$ (co-r.e.) |
| $+$ [14, 24], Sec. 2.2 | encodings of $\Delta_1^0$ | encodings of $\Sigma_1^0$ | encodings of $\Pi_1^0$ |
| $\cup, +, -$ [15, 16] | $\Delta_1^1$ (HA) | ? | $\Sigma_1^1$ |
| $+, -$ [21] | encodings of $\Delta_1^1$ | ? | encodings of $\Sigma_1^1$ |

Table 2.1: Expressive power of systems of equations over sets of natural numbers

The only unknown instances in the table are the least solutions of systems using union, addition and subtraction and the more restricted case of only addition and subtraction. However, the encoding used in [21] is order-preserving. Thus, whatever the class of representable sets is in the first case, the second can represent them in an encoded form. Symmetry would suggest that these are the sets in $\Pi_1^1$, but it is left to be seen if that really is the case.

# Part II

# Languages

# Chapter 3

# Different types of languages

Languages are another thing familiar to all people. A good example is the language used to write this thesis: it is English suplemented with mathematical notion. The text consists of words. Words are sequences of capital and lower case letters in the Latin alphabet. Besides that, one can find spaces and punctuation marks in this written text. There are also numerals $(0, 1, 2, 3, 4, \ldots)$, Greek letters $(\pi, \sigma, \gamma, \ldots$ and $\Pi, \Sigma, \Gamma, \ldots)$, blackboard bold letters $(\mathbb{N}, \mathbb{P}, \mathbb{Q}, \ldots)$, symbols for operations $(+, -, \cdot, \ldots)$, and so on.

Natural languages, such as English, Finnish, Greek, Chinese or Urdu, are not exactly defined nor static. The lack of exact definability follows from the fact that whether a word belongs in a language or not might depend on the context. For example, there are many dialects of languages and it is not always easy to decide if a word like YO! should be accepted as a word in English. The lack of staticness is a consequence of the ever ongoing process of language evolution: new words are introduced and old ones are forgotten, so natural languages and grammars defining them are under a constant change.

The most familiar and common way of defining a formal language is a formal grammar. The earliest known example of a formal grammar is Pāṇini's grammar for the Sanskrit language, from around 500 BCE. Modern research on formal grammars started after Noam Chomsky introduced context-free grammars in the late 1950s. Grammars in their most familiar form express the structure of sentences in a natural language. For example, the grammar with the rules

$$\begin{aligned}
\text{(sentence)} &\rightarrow \text{(subject)(predicate)} \\
\text{(predicate)} &\rightarrow \text{(verb)} \mid \text{(verb) (object)} \\
\text{(subject)} &\rightarrow \text{It} \mid \text{Frank} \mid \text{The dog} \\
\text{(verb)} &\rightarrow \text{is} \mid \text{has} \mid \text{eats} \\
\text{(object)} &\rightarrow \text{an apple} \mid \text{a dog} \mid \text{64 elephants}
\end{aligned}$$

prescribes that sentences consist of a subject and a predicate, or of a subject,

a predicate and an object. The grammar generates a small fraction of English, 36 different sentences to be exact. However, it should be enough to give an intuitive perception on the functionality of generative grammars. It generates the sentences

It is

Frank eats an apple

Frank has 64 elephants

and

The dog is a dog

among others.

A complete grammar totally defining the English language should of course be much more complicated and actually, due to the non-exact and non-static nature of natural languages, such a grammar does not exist. Moreover, the meaning of a sentence like "It is" is heavily dependent on its context. The word "It" can refer to almost anything and the meaning of the word "is" is not clear without knowing what is being discussed.

In mathematics, exactness is required, and it is essential, to reason about abstract and well-defined objects. Therefore, at least in theory, the syntax and semantics of a mathematical language are defined in some precise way, though in practice, natural language is more or less freely used for convenience.

The grammar with the rules

$$\begin{aligned} S &\rightarrow (S+S) \mid (S \cdot S) \mid N \\ A &\rightarrow 1B \mid 2B \mid 3B \mid 4B \mid 5B \mid 6B \mid 7B \mid 8B \mid 9B \mid 0 \\ B &\rightarrow 1B \mid 2B \mid 3B \mid 4B \mid 5B \mid 6B \mid 7B \mid 8B \mid 9B \mid 0B \mid \varepsilon \end{aligned}$$

defines expressions over the natural numbers in base 10 using addition and multiplication. Here $S$ stands for an expression and $A$ stands for a number. So, an expression is either a number or a sum or product of two expressions, and a number is a sequence of numerals not starting with a zero, unless the number is zero. The last replacement for $B$ is the empty word, denoted by $\varepsilon$, which means that replacing $B$ with $\varepsilon$ makes it disappear.

In formal language theory, only the syntax of a language is in the scope of interest and words are just sequences of symbols without any meaning attached to them. In contrast to natural languages, formal languages are exact and static, although in some cases, the question whether a word belongs to a language is impossible to solve. So, formal languages consist of words that can be any sequences of letters and do not need to mean anything. Usually the words in formal languages look like abcacababcaaacbabcacb or 10001110001011, and do not need to resemble the words in natural languages.

## 3.1 Formal languages

A formal language is a set of words over a finite alphabet $\Sigma$. Words are sequences of letters in $\Sigma$, so a word $w$ can be written as

$$w = a_0 a_1 \cdots a_{n-1} \text{ or } w = b_1 b_2 \cdots b_n,$$

where $a_i, b_j \in \Sigma$. In this case, the length of $w$ is $n$, denoted by

$$|w| = n.$$

The unique word of length zero is called the empty word and is denoted by $\varepsilon$.

The concatenation $u \cdot v = uv$ of two words

$$\begin{aligned} u &= a_1 a_2 \cdots a_m \\ v &= b_1 b_2 \cdots b_n \end{aligned}$$

is the word with letters from $u$ continuing with letters from $v$:

$$uv = a_1 a_2 \cdots a_m b_1 b_2 \cdots b_n.$$

The prefix of length $k \geqslant 0$ of a word $w = a_1 a_2 \cdots a_n$ is $\mathrm{pref}_k(w) = a_1 a_2 \cdots a_k$ if $k < n$, and $\mathrm{pref}_k(w) = w$ if $k \geqslant n$. In other words, $\mathrm{pref}_k(w)$ is the word formed by the first $k$ letters of $w$, or the whole word $w$ if $k$ is larger than the length of $w$. Symmetrically, the suffix of length $k \geqslant 0$ of a word $w$, denoted by $\mathrm{suf}_k(w)$, is the word formed by the last $k$ letters of $w$.

The set of all words over $\Sigma$ is denoted by $\Sigma^*$, so that $L$ is a language over the alphabet $\Sigma$ if

$$L \subseteq \Sigma^*$$

or

$$L \in \mathcal{P}(\Sigma^*).$$

The set of words of length at most $k \geqslant 0$ is denoted by $\Sigma^{\leqslant k}$, so that

$$\Sigma^{\leqslant k} = \{w \in \Sigma^* \mid |w| \leqslant k\}.$$

Concatenation of languages $K, L \subseteq \Sigma^*$ is defined elementwise, so that $K \cdot L = KL = \{uv \mid u \in K,\ v \in L\}$.

The star $L^*$ for $L \subseteq \Sigma^*$ is defined by

$$L^* = \bigcup_{n=0}^{\infty} L^n = \{\varepsilon\} \cup L \cup (L \cdot L) \cup (L \cdot L \cdot L) \cup \ldots$$

and, applied to $\Sigma$, this definition gives the set of all words $\Sigma^*$ as was defined.

If $\Sigma = \{a\}$ is unary, containing only one letter, then the concatenation is commutative as the length of the word is enough to characterize the word making $(\Sigma^*, \cdot)$ isomorphic to $(\mathbb{N}, +)$ discussed in the first part through the mapping $a^n \to n$. It is a monoid isomorphism, as

$$a^k \cdot a^l = a^{k+l}.$$

So, from the algebra point of view it is irrelevant which of the two structures is used. In the following, it is assumed that $|\Sigma| > 1$.

The concatenation is associative

$$(w_1 \cdot w_2) \cdot w_3 = w_1 \cdot (w_2 \cdot w_3) \text{ for all } w_1, w_2, w_3 \in \Sigma^*$$

and the empty word is the unit element

$$\varepsilon \cdot w = w \cdot \varepsilon = w \text{ for all } w \in \Sigma^*.$$

The concatenation is not commutative, so $(\Sigma^*, \cdot)$ is a non-commutative monoid.

The concatenation of languages is associative

$$(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3) \text{ for all } L_1, L_2, L_3 \subseteq \Sigma^*$$

and the singleton set containing only the empty word is the unit element

$$\{\varepsilon\} \cdot L = L \cdot \{\varepsilon\} = L \text{ for all } L \subseteq \Sigma^*.$$

Thus, $(\mathcal{P}(\Sigma^*), \cdot)$ is also a non-commutative monoid.

Furthemore, the concatenation is distributive over union

$$L_1 \cdot (L_2 \cup L_3) = (L_1 \cdot L_2) \cup (L_1 \cdot L_3) \text{ for all } S_1, S_2, S_3 \subseteq \mathbb{N},$$

so that $(\mathcal{P}(\Sigma^*), \cup, \cdot)$ is a non-commutative semiring.

As in the case of natural numbers, the set of words $\Sigma^*$ can be embedded into the set of languages by the inclusion $w \hookrightarrow \{w\}$. Accordingly, the notation

$$\{w\} \cdot L = wL$$

is used. The image of $\Sigma^*$ under this inclusion is the class of singleton sets $\mathcal{P}_1(\Sigma^*)$.

The family of regular languages is one of the most important language families.

**Definition 3.1.1.** *Regular languages over the alphabet* $\Sigma$*, denoted by* $\mathrm{Reg}(\Sigma)$*, are defined as:*

- $\varnothing$ *is a regular language.*

- *$\{a\}$ is a regular language for all $a \in \Sigma$.*
- *If $K$ and $L$ are regular languages, then $K \cup L$ and $KL$ are regular languages.*
- *If $L$ is a regular language, then $L^*$ is a regular language.*

The unary regular languages correspond to the class of eventually periodic sets of natural numbers as $L$ is regular if and only if the set

$$\{n \in \mathbb{N} \mid a^n \in L\}$$

is eventually periodic.

The set of languages $\mathcal{P}(\Sigma^*)$ accompanied with the inclusion order is a complete lattice. The inclusion order for vectors of languages is defined componentwise, so that $(K_1, \ldots, K_n) \subseteq (L_1, \ldots, L_n)$ if $K_i \subseteq L_i$ for all $i = 1, \ldots, n$. The operations of union, intersection and concatenation are monotone with respect to inclusion while complementation is not monotone.

The distance between languages can be defined by

$$d(K, L) = 2^{\min(\{|w| \mid w \in K \triangle L\})},$$

where $K \triangle L = (K \setminus L) \cup (L \setminus K)$ is the symmetric difference. With this metric, $(\mathcal{P}(\Sigma^*), d)$ is an ultra-metric space and $(\mathcal{P}(\Sigma^*), \cdot, \cup, \cap, \overline{\phantom{x}})$ is an ultra-metric algebra as the operations are continuous.

## 3.2 Language equations

Language equations are equations where variables get values from languages over a given alphabet. The expressions in them usually contain the set-theoretic operations union, intersection and complementation, and the operation of concatenation. The constants are often singletons, finite sets or regular languages. If $\mathfrak{S} \in \mathfrak{S}[\mathcal{X}, \mathcal{C}, (\mathcal{P}(\Sigma^*), \cdot, \cup, \cap)]$ is a system of language equations and it has a solution $\mathcal{S} \in \mathcal{P}(\Sigma^*)^{\mathcal{X}}$, then the components $\Pi_X(\mathcal{S})$ of the solution are languages. The systems can be used to represent languages by fixing a variable $X \in \mathcal{X}$ and a solution $\mathcal{S}$ of the system and saying that the system represents the language $\Pi_X(\mathcal{S})$. Usually, the solutions considered are unique, least or greatest, in order to be easily distinguishable from other possible solutions.

If a system of equations in $\mathfrak{S}[\mathcal{X}_n, \mathcal{C}, (\mathcal{P}(\Sigma^*), \cdot, \cup, \cap)]$ is in the resolved form with the equations

$$X_i = \varphi_i(X_1, \ldots, X_n)$$

for $n = 1, \ldots, n$, then it has a least and greatest solution by Tarski's fixed point theorem (Lemma 1.1.1). In formal language theory, the most usual

way of representing languages by language equations are by least solutions of such systems.

If also complementation is allowed, the least and greatest solutions do not need to exist. In fact, resolved systems in $\mathfrak{S}[\mathcal{X}, \mathcal{C}, (\mathcal{P}(\Sigma^*), \cdot, \cup, \cap, \overline{\ })]$ can represent all languages that can be represented by unresolved systems. Consider an equation $\varphi(\mathcal{X}) = \psi(\mathcal{X})$. It has a solution $\mathcal{S}$ if and only if the symmetric difference

$$\varphi(\mathcal{S}) \triangle \psi(\mathcal{S}) = \big(\varphi(\mathcal{S}) \cap \overline{\psi(\mathcal{S})}\big) \cup \big(\psi(\mathcal{S}) \cap \overline{\varphi(\mathcal{S})}\big)$$

equals the empty set. Now introducing a new variable $Y \notin \mathcal{X}$ makes it possible to construct the equation

$$Y = \big(\varphi(\mathcal{X}) \triangle \psi(\mathcal{X})\big) \cap \overline{Y}$$

that has the solution $Y = \varnothing$, $\mathcal{X} = \mathcal{S}$ if and only if $\varphi(\mathcal{S}) = \psi(\mathcal{S})$, and no solution if $\varphi(\mathcal{X}) = \psi(\mathcal{X})$ has no solution.

The unresolved systems can represent recursive languages as unique solutions, making them computationally hard. The following definition is useful for considerations on language equations of the general form.

**Definition 3.2.1.** *Let $\varphi = \psi \in \mathbb{E}[\mathcal{X}, \mathcal{C}, (\mathcal{P}(\Sigma^*), \cdot, \cup, \cap, \overline{\ })]$ and $n \in \mathbb{N}$. The equation is said to have a solution modulo $\Sigma^{\leqslant n}$ if there is $\mathcal{S} \in \mathcal{P}(\Sigma^*)^{\mathcal{X}}$ such that*

$$d(\varphi(\mathcal{S}), \psi(\mathcal{S})) \leqslant 2^{n+1}.$$

In other words, the equation $\varphi = \psi$ has the solution $\mathcal{S}$ if $\varphi(\mathcal{S})$ and $\psi(\mathcal{S})$ are equal on words of length at most $n$. A system of equations has a solution modulo $\Sigma^{\leqslant n}$ if there is a solution modulo $\Sigma^{\leqslant n}$ to all equations in the system. Similarly, solutions modulo $L$ for any subword-closed language $L$ can be defined by

$$\varphi(\mathcal{S}) \cap L = \psi(\mathcal{S}) \cap L.$$

Since $(\mathcal{P}(\Sigma^*), \cdot, \cup, \cap, \overline{\ })$ is ultra-metric, for every system $\mathfrak{S} \in \mathfrak{S}[\mathcal{X}, \mathcal{C}, (\mathcal{P}(\Sigma^*), \cdot, \cup, \cap, \overline{\ })]$ there exists such a natural number $n$ that $\mathfrak{S}$ has a solution if and only if it has a solution modulo $\Sigma^{\leqslant n}$. This follows from Lemma 1.1.2, as it states the existence of such a number $n$ that if a solution modulo $\Sigma^{\leqslant 0}$ can be extended to a solution modulo $\Sigma^{\leqslant n}$, then it can be extended to a full solution. As there is a finite number of classes modulo $\Sigma^{\leqslant n}$, there exists an algorithm checking solvability of a system modulo $\Sigma^{\leqslant n}$. This can be done for every $n$ starting from 0 and return not solvable, if an $n$ without a solution modulo $\Sigma^{\leqslant n}$ is encountered. If there is a solution, the algorithm goes on infinitely, this being the case for every algorithm by the following Proposition:

**Proposition 3.2.1** (Okhotin [29]). *The problem of solution existence for* $\mathfrak{S}[\mathcal{X}, \mathcal{P}_1(\Sigma^*), (\mathcal{P}(\Sigma^*), \cdot, \cup, \cap, \overline{\ })]$ *is co-r.e.-complete.*

This completeness can be achieved by much more simpler systems because Theorem 2.2.2 can be interpreted as a theorem on unary languages as follows:

**Theorem 3.2.1** (Theorem 2.2.2). *The problem of solution existence for systems*

$$\begin{aligned} XK &= L \\ XXM &= XXN \end{aligned}$$

*over* $\{a\}$*, where* $K, L, M, N \subseteq a^*$ *are regular, is co-r.e.-complete.*

Similarly, the subtraction of sets of natural numbers can be seen as the quotient of unary languages. Jeż and Okhotin have proved that the components of unique solutions of systems over sets of numbers using union, addition, subtraction, and eventually periodic constants ([16], Proposition 2.2.1) are the hyper-arithmetical sets. The encoding in [21] enables the representation of encodings of all hyper-arithmetical sets in the unique solutions of systems of unary language equations using one variable, concatenation, quotient and regular constants:

**Theorem 3.2.2** (Theorem 2.2.3). *Let* $S \subseteq \mathbb{N}$ *be hyper-arithmetical. There are natural numbers* $p, d \in \mathbb{N}$ *and regular unary languages* $K, L, M, N \subseteq a^*$*, such that a natural number* $n \in \mathbb{N}$ *belongs to* $S$ *if and only if* $a^{pn+d}$ *belongs to the unique solution to*

$$\begin{aligned} XK &= L \\ (XX)M &= (XX^{-1})N. \end{aligned}$$

This implies the following result on the complexity of solution existence for these systems:

**Theorem 3.2.3** (Lehtinen [21]). *The problem of solution existence for systems*

$$\begin{aligned} XK &= L \\ (XX)M &= (XX^{-1})N \end{aligned}$$

*over* $\{a\}$*, where* $K, L, M, N \subseteq a^*$ *are regular, is* $\Sigma^1_1$*-complete.*

Since the languages here are unary, the results have consequences also on the decidability of questions concerning language equations with other

operations than concatenation and quotient. One example is the shuffle $\sqcup\!\sqcup$, defined as

$$u \sqcup\!\sqcup v = \{u_1 v_1 u_2 v_2 \cdots u_k v_k \mid u = u_1 \cdots u_k, v = v_1 \cdots v_k, u_i, v_i \in \Sigma^*\}.$$

Although the shuffle of two words is defined to be a set of words, all of these words are equal to the concatenation of $u$ and $v$ in the unary case. These kind of operations have been studied among others by Lila Kari [17], who investigated language equations $X \Diamond K = L$, where $\Diamond$ is an invertible word operation with a right-inverse $\Box$. Here invertible means that for all words $u, v, w \in \Sigma^*$ it holds that

$$w \in (u \Diamond v) \text{ if and only if } v \in (u \Box w),$$

and in this case $\Diamond$ and $\Box$ are right-inverses of each other.

The operations considered by Kari were different insertion and deletion operations that have the common property that when restricted to the unary alphabet they coincide with the concatenation and quotient. Kari proved decidable many instances of solution existence for $X \Diamond K = L$, where $K$ and $L$ are regular constants. From the above theorems it follows that the problem of solution existence for systems of two equations

$$\begin{aligned} X \Diamond K &= L \\ X \Diamond X \Diamond M &= X \Diamond X \Diamond N \end{aligned}$$

with regular constants $K$, $L$, $M$ and $N$ and an insertion operation $\Diamond$ is undecidable, and in fact co-r.e.-hard.

Similarly, the problem of solution existence of systems of two equations

$$\begin{aligned} X \Diamond K &= L \\ (X \Diamond X) \Diamond M &= (X \Box X) \Diamond N \end{aligned}$$

with regular constants $K$, $L$, $M$ and $N$, an insertion operation $\Diamond$ and a deletion operation $\Box$ is $\Sigma^1_1$-hard.

## 3.3 Grammars and families of languages

Grammars are used to generate languages. They are defined as follows:

**Definition 3.3.1.** *Let $k, l \in \mathbb{N}$. A (Boolean) rule is an expression*

$$A \to \alpha_1 \& \ldots \& \alpha_k \& \neg\beta_1 \& \ldots \& \neg\beta_l,$$

*where $A \in N$ and $\alpha_i, \beta_j \in (N \cup \Sigma)^*$.*

*The words $\alpha_i$ and $\beta_j$ appearing on the right-hand side of the rule are called positive and negative conjuncts respectively.*

*If $l = 0$, the rule is called conjunctive, and if in addition $k = 1$, it is called context-free.*

**Definition 3.3.2.** *A grammar $G = (\Sigma, N, R, S)$ is a 4-tuple, where*

- *$N$ is the set of nonterminals*
- *$\Sigma$ is the terminal alphabet*
- *$R$ is the set of rules*
- *$S \in N$ is the initial nonterminal*

*The grammar is called Boolean, conjunctive or context-free if all the rules in the grammar are Boolean, conjunctive or context-free respectively.*

The language generated by the grammar, $L_G \subseteq \Sigma^*$, is most easily defined by the rewriting of words.

**Definition 3.3.3.** *Let $G = (\Sigma, N, R, S)$ be a conjunctive grammar.*

*If $\mu_1, \mu_2 \in (N \cup \Sigma \cup \{(,), \&\})^*$ are such that $\mu_1 = \nu_1 A \nu_2$ and $\mu_2 = \nu_1(\alpha_1 \& \ldots \& \alpha_k)\nu_2$, or $\mu_1 = \nu_1(w \& w \& \ldots \& w)\nu_2$ and $\mu_2 = \nu_1 w \nu_2$, then $\mu_2$ is derivable from $\mu_1$ in one step, denoted by $\mu_1 \to \mu_2$. If there is a sequence of one step derivations*

$$\mu_1 \to \mu_2 \to \ldots \to \mu_n,$$

*then $\mu_n$ is derivable from $\mu_1$, denoted by $\mu_1 \to^* \mu_n$.*

*The language generated by a nonterminal $A \in N$ is*

$$L_A = \{w \in \Sigma \mid A \to^* w\}.$$

*Sometimes the notation $L_G(A)$ is used to emphasize the grammar.*

*The language generated by $G$ is the language generated by the initial nonterminal $S$ :*

$$L_G = L_S.$$

The notation $L_\alpha$ is also used to denote the language corresponding to $\alpha$ according to the grammar for any $\alpha \in (N \cup \Sigma)^*$, so that $L_\alpha = L_{A_1} L_{A_2} \cdots L_{A_n}$ for $\alpha = A_1 A_2 \cdots A_n$, where $L_a = \{a\}$ for all $a \in \Sigma$.

However, this rewriting semantics is not used in this thesis because the semantics through language equations will be used instead. Ultimately, the reason for this is that Boolean grammars cannot be given a semantics through rewriting. The rule

$$A \to \alpha_1 \& \ldots \& \alpha_k \& \neg\beta_1 \& \ldots \& \neg\beta_l$$

should generate $w$ if it is in every $L_{\alpha_i}$ and not in any $L_{\beta_i}$. This would mean that something is not generated by $\beta_i$, which would make things very complicated as nonterminals in $\beta_i$'s can have negative conjuncts in rules as well.

Semantics for Boolean grammars can be defined by language equations.

**Definition 3.3.4.** *Let $G = (\Sigma, N, R, S)$ be a Boolean grammar. The corresponding system of equations $\mathfrak{S}(G) \in \mathfrak{S}[N, \mathcal{P}_1(\Sigma^*), (\mathcal{P}(\Sigma^*), \cdot, \cup, \cap, \overline{\ })]$ contains the equation*

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_k \& \neg\beta_1 \& \ldots \& \neg\beta_l \in R} \alpha_1 \cap \ldots \cap \alpha_k \cap \overline{\beta_1} \cap \ldots \cap \overline{\beta_l}$$

*for each $A \in N$.*

*If $|N| = n$, the variables of the corresponding system are usually chosen to be $\mathcal{X}_n$, with a bijection $N \to \mathcal{X}_n$ mapping $S$ to $X_1$.*

For each conjunctive grammar $G = (\Sigma, N, R, X_S)$, the corresponding system of equations $\mathfrak{S}(G)$ is in $\mathfrak{S}[\mathcal{X}_n, \mathcal{P}_1(\Sigma^*), (\mathcal{P}(\Sigma^*), \cdot, \cup, \cap)]$. The operations are monotone, so the system has a least solution $\mathcal{S} \in \mathcal{P}(\Sigma^*)^n$, and for this least solution it holds that $\Pi_1(\mathcal{S}) = L_G$.

For context-free grammars, the corresponding systems of equations are in $\mathfrak{S}[\mathcal{X}_n, \mathcal{P}_1(\Sigma^*), (\mathcal{P}(\Sigma^*), \cup, \cdot)]$ and they also have the language generated by the grammar as a component of the least solution.

**Example 3.3.1.** *Let $N = \{S\}$, $R$ contain the rules*

$$\begin{aligned} S &\to aSb \\ S &\to \varepsilon \end{aligned}$$

*and $G = (\{a, b\}, N, R, S)$. The language generated by $G$ is*

$$L_G = L_S = \{a^n b^n \mid n > 0\}.$$

*For example, the word aabb is generated by the rewriting*

$$S \to aSb \to aaSbb \to aa \cdot \varepsilon \cdot bb = aabb.$$

*Let $\mathfrak{S} \in \mathfrak{S}[\mathcal{X}_1, \mathcal{C}, (\mathcal{P}(\Sigma^*), \cup, \cdot)]$ be the system*

$$X = \{a\}X\{b\} \cup \{\varepsilon\}.$$

*It has the unique solution $X = \{a^n b^n \mid n > 0\} = L_G$.* □

The conjunctive grammars are more powerful than context-free grammars, as they can generate the following non-context-free language:

**Example 3.3.2.** *Let $N = \{S, A, B, C\}$, $R$ contain the rules*

$$\begin{aligned} S &\to AB\&CA \\ A &\to aA \\ A &\to \varepsilon \\ B &\to bBa \\ B &\to \varepsilon \\ C &\to aCb \\ C &\to \varepsilon \end{aligned}$$

*and* $G = (\{a, b\}, N, R, S)$*. The language generated by* $G$ *is*

$$L_G = \{a^n b^n a^n \mid n \in \mathbb{N}\}.$$

*The rewriting of conjuncts happens simultaneously as in the derivation*

$$\begin{aligned} S &\to AB\&CA \to aAB\&CaA \to aB\&Ca \to \\ &\to abBa\&aCba \to aba\&aba \to aba \end{aligned}$$

*of the word* $aba$*.*

*Let* $\mathfrak{S}_3 \in \mathfrak{S}[\mathcal{X}_3, \mathcal{C}, (\mathcal{P}(\Sigma^*), \cdot, \cup, \cap)]$ *be the system*

$$\begin{aligned} X_1 &= a^* X_2 \cap X_3 a^* \\ X_2 &= bX_2 a \cup \{\varepsilon\} \\ X_3 &= aX_3 b \cup \{\varepsilon\}. \end{aligned}$$

*It has the least solution* $(X_1, X_2, X_3) = (S_1, S_2, S_3)$*, where*

$$\begin{aligned} S_1 &= \{a^n b^n a^n \mid n \in \mathbb{N}\} \\ S_2 &= \{b^n a^n \mid n \in \mathbb{N}\} \\ S_3 &= \{a^n b^n \mid n \in \mathbb{N}\}. \end{aligned}$$

*Here* $S_1 = L_G$*, although it is not the system corresponding to* $G$ *because the regular components have been considered as constants.* □

The general case of systems corresponding to Boolean grammars is more complicated since complementation is not a monotone operator. For example, the grammar with only one nonterminal $S$ and rule $S \to \neg S$ does not have an intuitive meaning, and the corresponding system with the equation $X = \overline{X}$ does not have any solutions. Furthermore, the languages representable as the unique solutions of systems in $\mathfrak{S}[\mathcal{X}, \mathcal{P}_1(\Sigma^*), (\mathcal{P}(\Sigma^*), \cdot, \cup, \cap, \overline{\ })]$ are the recursive languages, so their expressive power is too big for practical applications.

The easiest, although in some senses unsatisfactory, way of defining semantics for a Boolean grammar is through the so-called strongly unique solution.

**Definition 3.3.5.** *Let* $\mathfrak{S} \in \mathfrak{S}[\mathcal{X}, \mathcal{C}, (\mathcal{P}(\Sigma^*), \cdot, \cup, \cap, \overline{\ })]$*. It is said to have a strongly unique solution if it has a unique solution modulo* $\Sigma^{\leqslant n}$ *for all* $n \in \mathbb{N}$*.*

A system with a strongly unique solution always has a unique solution.

Using this definition, Boolean grammars with the corresponding system of equations having a strongly unique solution can be given a meaning: the unique solution to the system, whereas grammars with the corresponding system without a strongly unique solution are considered ill-formed and are

not given a meaning. The drawback of this definition is that some grammars with an intuitively clear meaning are considered ill-formed. The simplest one of these is the grammar with only the rule $S \to S$. It should generate the empty language by the least solution semantics, but the corresponding system $X = X$ does not have a unique solution: all languages are solutions! However, for any conjunctive language, there exists a conjunctive grammar generating that language, with a corresponding system having a strongly unique solution.

Boolean grammars have the following normal form used in the proofs of the next chapter:

**Definition 3.3.6.** *A Boolean grammar $G = (\Sigma, N, R, S)$ is said to be in binary normal form, if all rules in $R$ are of the form*

$$A \to B_1C_1\& \ldots \&B_kC_k\&\neg D_1E_1\& \ldots \&\neg D_lE_l\&\neg\varepsilon$$

*for $k \geqslant 1$, $l \geqslant 0$ and $A, B_i, C_i, D_i, E_i \in N$,*

$$A \to a$$

*for $a \in \Sigma$ and $A \in N$ or*

$$S \to \varepsilon.$$

*The rule of the last type is allowed only if $S$ does not appear in the right-hand sides of any rules.*

Every Boolean grammar can be effectively transformed into an equivalent grammar in binary normal form generating the same language. Furthermore, every system of equations corresponding to a Boolean grammar in binary normal form has a strongly unique solution.

**Example 3.3.3.** *Let $N = \{S, A, B, C, D\}$, $R$ contain the rules*

$$\begin{aligned} S &\to AB\&\neg CA \\ A &\to aA \\ A &\to \varepsilon \\ B &\to bBa \\ B &\to \varepsilon \\ C &\to aCb \\ C &\to \varepsilon \end{aligned}$$

*and $G = (\{a, b\}, N, R, S)$. The language generated by $G$ is*

$$L_G = \{a^m b^n a^n \mid m, n \in \mathbb{N},\ m \neq n\}.$$

□

It is not known if Boolean grammars can generate more languages than conjunctive grammars, but the language $\{ww \mid w \in \{a,b\}^*\}$ is a good candidate for a counterexample as there is a Boolean grammar generating it, and no conjunctive grammar for it is known.

Grammars can be restricted in order to define subfamilies of languages. For example, if all rules in a context-free grammar are of the form

$$A \to aB$$

or

$$A \to a,$$

then the grammar is called left-linear and the language it generates is regular. If rules

$$A \to aBb$$

and

$$A \to a$$

are allowed, then the grammar is linear and it generates a linear context-free language. Similarly, if the only rules containing nonterminals on the right-hand side in a conjunctive grammar are of the form

$$A \to u_1 B_1 v_1 \& \ldots \& u_n B_n v_n,$$

where $u_i, v_i \in \Sigma^*$, then the grammar is linear conjunctive and generates a linear conjunctive language. The linear Boolean grammars could be defined accordingly, but they generate the same family of languages as the linear conjunctive grammars [28].

Some grammars are ambiguous in the sense that one word can be generated in many ways. This is sometimes undesirable and the grammars satisfying the following definition are an important subclass of grammars.

**Definition 3.3.7.** *A Boolean grammar $G = (\Sigma, N, R, S)$ is said to be unambiguous if*

- *For every nonterminal $A$ and word $w$, there exists at most one rule*

  $$A \to \alpha_1 \& \ldots \& \alpha_k \& \neg\beta_1 \& \ldots \& \neg\beta_l,$$

  *with*

  $$w \in L_{\alpha_1} \cap \ldots \cap L_{\alpha_k} \cap \overline{L_{\beta_1}} \cap \ldots \cap \overline{L_{\beta_l}}$$

  *(in other words, different rules for a single nonterminal generate disjoint languages).*

- *For every conjunct $\alpha = A_1 \cdots A_n$ or $\beta = \neg A_1 \cdots A_n$ that occurs in the grammar, and for every word $w$, there exists at most one partition $w = u_1 \cdots u_n$ with $u_i \in L_{A_i}$.*

*A (context-free, conjunctive, Boolean) language $L$ is called inherently ambiguous if there is no (context-free, conjunctive, Boolean) unambiguous grammar generating $L$.*

The following example describes a grammar generating an inherently ambiguous context-free language:

**Example 3.3.4.** *Let $N = \{S, A, B, C\}$, $R$ contain the rules*

$$\begin{aligned} S &\to AB \\ S &\to CA \\ A &\to aA \\ A &\to \varepsilon \\ B &\to bBa \\ B &\to \varepsilon \\ C &\to aCb \\ C &\to \varepsilon \end{aligned}$$

*and $G = (\{a, b\}, N, R, S)$. The language generated by $G$ is*

$$L_G = \{a^k b^l a^m \mid k = l \text{ or } l = m\}.$$

*The words $a^n b^n a^n$ are generated by both of the rules for $S$.* □

However, the language of the example is not inherently ambiguous as a conjunctive or Boolean language because there exists an unambiguous conjunctive grammar for the language.

Besides grammars of particular type, families of languages can be defined through recognition. Regular languages are recognized by finite automata, with a bounded memory represented by the states of the automaton.

Context-free languages can be recognized by pushdown automata simulating grammars in Greibach normal form, where all rules containing nonterminals on right-hand sides are of the form

$$A \to aBC.$$

A pushdown automaton has a stack, where it can push to or pull from a symbol according to the letter being read, its state and the topmost symbol of its stack. Formally, a pushdown automaton is defined as follows:

**Definition 3.3.8.** *A pushdown automaton (PDA) is a septuple $B = (\Sigma, \Gamma, Q, q_0, \delta, F, \gamma_0)$, where*

- *$\Sigma$ is a finite alphabet*

- $\Gamma$ *is the stack alphabet*
- $Q$ *is the set of states*
- $q_0 \in Q$ *is the initial state*
- *The configurations of the automaton are triples* $(q, w, x)$*, where* $q \in Q$*,* $w \in \Sigma^*$ *and* $x \in \Gamma^*$
- *the transition function* $\delta$ *maps* $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ *to the set of finite subsets of* $Q \times \Gamma^*$
- $F \subseteq Q$ *is the set of final states*
- $\gamma_0 \in \Gamma$ *is the initial stack symbol*

The relation $\vdash$ of one-step transition on the set of these configurations is defined as $(q, aw, \gamma z) \vdash (q', w, yz)$ for all $(q', y) \in \delta(q, a, \gamma)$. The language recognized by the PDA is

$$L(B) = \{w \in \Sigma^* \mid (q_0, w, \gamma_0) \vdash^* (q_F, \varepsilon, \varepsilon) \text{ for some } q_F \in F\}.$$

For a context-free grammar in Greibach normal form, the corresponding PDA reading letter $a$ from the input word pulls a nonterminal $A$ from the stack and pushes $BC$ to the stack, if there is a rule $A \to aBC$ in the grammar. In the general case, there can be many rules of that form, and consequently the automaton behaves in a nondeterministic way. This means that there are many different possibilities for the computations to emerge. The deterministic variant of PDA has just one way to continue the computation in every step. So, a PDA is deterministic if $\delta(q, a, \gamma)$ has at most one element for all $(q, a, \gamma) \in Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$, and if $\delta(q, \varepsilon, \gamma)$ is not empty, then $\delta(q, a, \gamma)$ is empty for all $a \in \Sigma$.

As an example of a context-free language not recognized by a deterministic PDA, there is the following language that is also encountered in the last chapter:

**Example 3.3.5.** *Let* $N = \{S, A, B\}$*,* $R$ *contain the rules*

$$\begin{aligned} S &\to A \\ S &\to B \\ A &\to aAb \\ A &\to ab \\ B &\to aBbb \\ B &\to abb \end{aligned}$$

*and* $G = (\{a, b\}, N, R, S)$*. The language generated by* $G$ *is*

$$L_G = \{a^n b^n \mid n > 1\} \cup \{a^n b^{2n} \mid n > 1\}$$

*Here it has to be decided in the first step of the rewriting whether a word of the form* $a^n b^n$ *or a word of the form* $a^n b^{2n}$ *is to be generated. The pushdown automaton recognizing the language has to make a nondeterministic guess at the border of a's and b's.*

*The language* $\{ba^n b^n \mid n > 1\} \cup \{bba^n b^{2n} \mid n > 1\}$ *on the other hand is deterministic context-free, as reading one or two b's in the beginning is sufficient to distinguish the two cases before reaching the border.* □

Consider a further subfamily of deterministic context-free languages defined by LL($k$) grammars. These are grammars that can be parsed with a lookahead $k$.

**Definition 3.3.9.** *A context-free grammar* $G = (\Sigma, N, R, S)$ *is called LL(k), if* $S \to^* xA\beta$*,* $A \to \alpha_1$*,* $A \to \alpha_2 \in R$*,* $w_1 \in L_G(\alpha_1\beta)$*,* $w_2 \in L_G(\alpha_2\beta)$ *and* $\mathrm{pref}_k(w_1) = \mathrm{pref}_k(w_2)$ *implies* $\alpha_1 = \alpha_2$*.*

Accordingly, for an LL($k$) grammar, there exists a partial function $T\colon N \times \Sigma^{\leqslant k} \to R$, such that whenever $S \to^* xA\beta$, $A \to \alpha \in R$ and $w \in L_G(\alpha\beta)$, the rule $A \to \alpha$ is given by $T(A, \mathrm{pref}_k(w))$.

A language is LL context-free, if there exists an LL($k$) grammar generating it for some $k$.

**Example 3.3.6.** *The grammar in Example 3.3.1 generating the language* $\{a^n b^n \mid n \in \mathbb{N}\}$ *is LL(1), since the rule used to generate a letter can be known instantly. If the word* $a^n b^n$ *is read from left to right, then reading the letter a means that the rule*

$$S \to aSb$$

*is used and at the first occurence of b, the rule*

$$S \to \varepsilon$$

*is used.* □

A non-LL language is presented in the following example:

**Example 3.3.7.** *The language* $\{a^n b^n \mid n \in \mathbb{N}\} \cup \{a^n c^n \mid n \in \mathbb{N}\}$ *is not LL context-free. The grammar defining it has to have different rules to generate words of the form* $a^n b^n$ *and of the form* $a^n c^n$*, and these cannot be distinguished reading only a's in the beginning. The language is deterministic context-free, as a pushdown automaton can decide if it should compare the number of a's to b's or c's at the first appearance of either letter.*

*The language* $\{ba^n b^n \mid n \in \mathbb{N}\} \cup \{ca^n c^n \mid n \in \mathbb{N}\}$ *on the other hand is LL(1) as the correct ending of the word is known after reading the first letter.* □

# Chapter 4

# Closure properties of language families

Closure properties of language families are among the most important theoretical results in formal language theory. Operations under interest are the set-theoretic operations like union, intersection and complementation, the algebraic operations like concatenation and different kinds of mappings.

Formally, the closure of a language family under an operation is defined as:

**Definition 4.0.10.** *If $\mathcal{F}$ is a family of languages and $f\colon \mathcal{F}^n \to \mathcal{F}$ a mapping, then $\mathcal{F}$ is closed under $f$ if*

$$L_1, \ldots, L_n \in \mathcal{F}$$

*implies*

$$f(L_1, \ldots, L_n) \in \mathcal{F}.$$

For example, the family of context-free languages is closed under concatenation and union, but it is not closed under intersection or complementation. Conjunctive languages, on the other hand, are trivially closed under intersection as it is built in their definition, and their closure under complement is not known. Boolean languages are closed under all three set-theoretic operations by definition.

Among the standard operations on languages are also morphisms that respect concatenation and the more general gsm-mappings: these are mappings $M : \Sigma^* \to \Gamma^*$ implemented by deterministic transducers (generalized sequential machines, gsm). As shown by Ginsburg and Rose [7], context-free languages are closed under gsm-mappings; an easy proof of this fact given by Harrison [9, Th. 6.4.3] is by simulating a gsm within a pushdown automaton. On the contrary, the languages generated by Boolean grammars are already not closed under morphisms: in fact, all recursively enumerable languages

can even be obtained as morphic images of languages generated by linear conjunctive grammars, because the computations of Turing machines can be represented as an intersection of two linear context-free languages, and consequently, all recursively enumerative languages appear as morphic images of linear conjunctive languages. The closure of Boolean or conjunctive languages under non-erasing morphisms remains an open problem, although the closure is not likely as it would imply $P = NP$.

For a gsm-mapping $M : \Sigma^* \to \Gamma^*$, the inverse is defined as $M^{-1}(L) = \{w \in \Sigma^* \mid M(w) \in L\}$ for every $L \subseteq \Gamma^*$. It is known from Ginsburg and Rose [7] that context-free languages are closed under inverse gsm-mappings. This argument was adapted to unambiguous context-free languages by Ginsburg and Ullian [8]. More accessible proofs of these results based upon pushdown automata were given by Harrison [9]. An examination of this argument shows that it also applies to linear context-free languages which are consequently closed under inverse gsm-mappings. Table 4.1 presents selected closure properties of language families.

| | $\cup$ | $\cap$ | $\overline{L}$ | $\cdot$ | $h$ | inj.gsm | $\text{gsm}^{-1}$ |
|---|---|---|---|---|---|---|---|
| Reg | + | + | + | + | + | + | + |
| LL | − [34] | − [34] | − [34] | − [34] | − [34] | − | − E5.2.1 |
| DetCF | − [5] | − [5] | + [5] | − [5] | − [5] | + L5.2.2 | + [5] |
| UnambCF | − [8] | − | − [10] | − [8] | + [8] | + [8] | + [8] |
| LinCF | + | − | − | − | + | + | + [9] |
| CF | + | − | − | + | + | + | + [7] |
| LinConj | + | + | + [28] | − [36] | − [28] | + [4] | + [11] |
| UnambConj | ? | + | ? | ? | − | + T4.2.1 | + T4.3.2 |
| Conj | + | + | ? | + | − | + T4.2.1 | + T4.3.2 |
| UnambBool | + | + | + | + | − | + T4.2.1 | + T4.3.2 |
| Bool | + | + | + | + | − | + T4.2.1 | + T4.3.2 |

Table 4.1: Closure properties

Closure properties are handy in proving languages to be or not to be in a given family. For example, if it is assumed to be known that the language $\{a^n b^n a^n \mid n \in \mathbb{N}\}$ is not context-free, as it is not, then it is easy to show that the language $\{u^n v^n u^n \mid n \in \mathbb{N}\}$ is not context-free either, as long as $u$ and $v$ are not powers of the same word. The inverse image of the morphism $a \to u$, $b \to v$ for this language is the language $\{a^n b^n a^n \mid n \in \mathbb{N}\}$ assumed to be known non-context-free. The family of context-free languages is closed under inverse morphisms, so $\{u^n v^n u^n \mid n \in \mathbb{N}\}$ cannot be context-free either. A direct proof for the lack of existence of a context-free grammar for that language would require quite a complicated argument. Similarly, it can be argued that the language $\{u^n v^n | n \in \mathbb{N}\}$ is context-free under the assumptions

that $\{a^n b^n \mid n \in \mathbb{N}\}$ is context-free and the family of context-free languages is closed under morphisms, although in this case, a context-free grammar for the language would also be easy to construct.

This chapter investigates the closure of Boolean languages and their subfamilies under inverse gsm-mappings. The proof also uses closure under other operations. It is proved that the families are closed under injective gsm-mappings and under inverses of a special case of gsm-mappings called weak codings. As the inverses of gsm-mappings can be represented as a combination of injective gsm-mappings and weak codings, the closure under inverse gsm-mappings follows. A different construction not using intermediate results can be found in the technical report *Boolean Grammars Are Closed Under Inverse Gsm Mappings* [22].

## 4.1 Gsm-mappings

In the remaining sections of this chapter, one of the contributions of this thesis is presented. It is proved that the family of Boolean languages and some of its subfamilies are closed under injective gsm-mappings and under inverse gsm-mappings. The proof is presented as in [23].

**Definition 4.1.1.** *A (deterministic) generalized sequential machine (gsm) is a septuple* $M = (\Sigma, \Gamma, Q, q_0, \delta, \lambda, F)$*, where*

- $\Sigma$ *is the input alphabet*
- $\Gamma$ *is the output alphabet*
- $Q$ *is the set of states*
- $q_0 \in Q$ *is the initial state*
- $\delta : Q \times \Sigma \to Q$ *is the transition function*
- $\lambda : Q \times \Sigma \to \Gamma^*$ *is the output function*
- $F \subseteq Q$ *is the set of final states*

The functions $\delta$ and $\lambda$ are extended to $Q \times \Sigma^*$ in the usual way, as $\delta(q, \varepsilon) = q$, $\delta(q, aw) = \delta(\delta(q, a), w)$ and as $\lambda(q, \varepsilon) = \varepsilon$, $\lambda(q, aw) = \lambda(q, a)\lambda(\delta(q, a), w)$. A gsm $M$ computes a partial function $M : \Sigma^* \to \Gamma^*$, where $M(w) = \lambda(q_0, w)$ and $\delta(q_0, w) \in F$. In some literature, gsm's are defined not to reject any input and hence compute complete functions from $\Sigma^*$ to $\Gamma^*$. This thesis consistently uses partial mappings computed by gsm's with accepting states. The partial mapping can also be considered as a mapping $M : L(M) \to \Gamma^*$, where $L(M)$ is the language recognized by the underlying finite automaton of the gsm.

**Example 4.1.1.** *Let $M = (\{a,b\}, \{a,b\}, \{q\}, q, \delta, \lambda, \{q\})$ be a gsm, where the transition function is defined by $\delta(q,a) = \delta(q,b) = q$, and the output function by $\lambda(q,a) = u$ and $\lambda(q,b) = v$ for some $u, v \subseteq \{a,b\}^*$.*

*It maps instances of a's and b's into instances of u's and v's, so that, for example, $M(a^n) = u^n$, $M(b^n) = v^n$, $M((ab)^n) = (uv)^n$ and $M(a^n b^n) = u^n v^n$.* □

The mapping in the above example is a morphism. Every morphism can be computed by a one-state gsm in a similar manner, so that morphisms are a special case of gsm-mappings.

**Example 4.1.2.** *Define a gsm $M = (\{a,b\}, \{a,b\}, \{q_0,q_1\}, q_0, \delta, \lambda, \{q_0,q_1\})$, where the transition function is defined by $\delta(q_0,a) = \delta(q_0,b) = q_1$ and $\delta(q_1,a) = \delta(q_1,b) = q_0$, and the output function by $\lambda(q_0,a) = \lambda(q_1,b) = a$ and $\lambda(q_0,b) = \lambda(q_1,a) = b$.*

*It changes every second letter in a word, so that, for example, $M(a^{2n}) = (ab)^n$, $M(b^{2n}) = (ba)^n$ and $M((ab)^n) = a^{2n}$.* □

A state $q$ of a gsm $M$ is said to be *useful* if it is reachable from the initial state and there is a path from it to some accepting state, that is, $\delta(q_0,u) = q$ and $\delta(q,v) \in F$ for some $u, v \in \Sigma^*$. Denote by $\widehat{Q} \subseteq Q$ the set of useful states.

A gsm $M$ is called *injective* if the function it computes is injective, that is, if $M(w) = M(w')$ implies $w = w'$. The graph of any injective gsm's transitions that output $\varepsilon$ (erasing transitions) is acyclic:

**Lemma 4.1.1.** *Let $M$ be an injective gsm. Consider the graph $(V,E)$ of its erasing transitions, with $V = \widehat{Q}$ and $E = \{(q,q') \mid \exists a \in \Sigma : \delta(q,a) = q', \lambda(q,a) = \varepsilon\}$. Then this graph is acyclic.*

*Proof.* Suppose the contrary, that is, there is a cycle passing through some state $q \in \widehat{Q}$. Then there is a word $w \neq \varepsilon$ with $\delta(q,w) = q$ and $\lambda(q,w) = \varepsilon$. Since $q$ is useful, there exist words $u, v \in \Sigma^*$ with $\delta(q_0,u) = q$ and $\delta(q,v) \in F$. Therefore, $\delta(q_0,uv) = \delta(q_0,uwv) \in F$ and $\lambda(q_0,uv) = \lambda(q_0,uwv)$, that is, $M(uv) = M(uwv)$, where $uv \neq uwv$. This contradicts the assumption that $M$ is injective. □

Since the graph is acyclic, there is a bound on the length of its paths. In terms of the given injective gsm, every path in this graph corresponds to a sequence of erasing transitions. So it is known that the number of such consecutive transitions is bounded, and for every state $q \in \widehat{Q}$ the following bounds can be defined:

$$d(q) = \max\{|w| \mid \delta(q,w) \in \widehat{Q}, \lambda(q,w) = \varepsilon\},$$
$$d'(q) = \max\{|w| \mid \exists q' : \delta(q',w) = q, \lambda(q',w) = \varepsilon\}.$$

In other words, $d(q)$ (and $d'(q)$, respectively) is the greatest number of erasing transitions starting from state $q$ (before entering state $q$, respectively) which is well-defined because of Lemma 4.1.1.

**Definition 4.1.2.** *Let $\Sigma$ be an alphabet. A mapping $h : \Sigma \to \Gamma^*$ is called a weak coding if $|h(a)| \leqslant 1$ for all $a \in \Sigma$.*

A weak coding extends to a mapping $h : \Sigma^* \to \Gamma^*$ by $h(a_1 a_2 \cdots a_n) = h(a_1)h(a_2)\cdots h(a_n)$. This is a gsm-mapping, computed by a gsm $M = (\Sigma, \Gamma, \{q\}, q, \delta, \lambda, \{q\})$, where $\delta(q, a) = q$ and $\lambda(q, a) = h(a)$ for all $a \in \Sigma$.

The goal of this chapter is to prove the closure of Boolean languages under inverse gsm-mappings. This will be done using the general result below. The construction is from Ginsburg and Ullian [8], who used it in the case of unambiguous context-free languages to establish their closure under inverse gsm-mappings.

**Proposition 4.1.1** ( [8])**.** *Let $\mathcal{L}$ be a family of languages closed under injective gsm-mappings and inverse weak codings. The family $\mathcal{L}$ is closed under inverse gsm-mappings.*

*Proof.* Let $L \subseteq \Gamma^*$ be any language. and let $M = (\Sigma, \Gamma, Q, q_0, \delta, \lambda, F)$ be a gsm. It is assumed that $\Sigma$ and $\Gamma$ are disjoint.

The first step is to define a simulation of $M$, which maps words in $\Sigma^*$ to words over $\Delta = \Sigma \cup \Gamma$. Define a gsm $M_1 = (\Sigma, \Delta, Q, q_0, \delta, \lambda_1, F)$, where

$$\lambda_1(q, a) = a\lambda(q, a).$$

Now the image $M_1(w)$ contains the computation history of $M$ on $w$, including the symbols of $w$ and the outputs. Then $M_1(\Sigma^*)$ becomes the language of valid computation histories of $M$; note that it is regular as an image of a gsm.

Next, define a weak coding $h : \Delta \to \Gamma$ that converts such a computation history into an output word of $M$. Let

$$h(c) = \begin{cases} \varepsilon, & \text{if } c \in \Sigma, \\ c, & \text{if } c \in \Gamma. \end{cases}$$

Now $h(M_1(w)) = M(w)$ for all $w \in \Sigma^*$.

It is also possible to reconstruct the input word of $M$ and $M_1$ from a computation history of this form. Let $A = (\Delta, Q_2, q_0^2, \delta_2, F_2)$ be a DFA recognizing the language $M_1(\Sigma^*)$. Define a gsm $M_2 = (\Delta, \Gamma, Q_2, q_0^2, \delta_2, \lambda_2, F_2)$, which in every state $q \in Q_2$ outputs $\lambda_2(q, a) = a$ for all $a \in \Sigma$ and $\lambda_2(q, b) = \varepsilon$ for all $b \in \Gamma$. Then $M_2(M_1(w)) = w$ for every $w \in \Sigma^*$ accepted by $M_1$. On the other hand, $M_2$ contains a recognizer for $M_1(\Sigma^*)$, so every word $x \notin M_1(\Sigma^*)$ is rejected. Therefore, every word $w$ in the domain of $M_1$ has a unique pre-image $M_2^{-1}(w) = M_1(w)$, that is, $M_2$ is injective.

Now the pre-image of every language $L \subseteq \Gamma^*$ under $M$ can be represented as follows:

$$M^{-1}(L) = M_2\big(h^{-1}(L)\big).$$

Here $h^{-1}$ attempts to re-construct computation histories of $M$ on words $w$ with $M(w) \in L$, then recognition of $M_1(\Sigma^*)$ by $M_2$ filters out ill-formed computation histories, and finally, $M_2$ extracts the actual words $w$ from these computation histories.

If $L \in \mathcal{L}$, then $h^{-1}(L)$ is in $\mathcal{L}$ as an inverse image of a weak coding and $M_2\big(h^{-1}(L)\big)$ belongs to $\mathcal{L}$ by the closure under injective gsm-mappings. So $M^{-1}(L)$ is in $\mathcal{L}$, which proves the closure of $\mathcal{L}$ under inverse gsm-mappings. □

## 4.2 Boolean grammars and injective gsm-mappings

In this section, it is proved that the family of languages generated by Boolean grammars is closed under injective gsm-mappings.

Let $M = (\Sigma, \Gamma, Q, q_0, \delta, \lambda, F)$ be an injective gsm, and let $\widehat{Q} \subseteq Q$ be the set of its useful states. Assume $G = (\Sigma, N, R, S)$ is in the binary normal form without the rule $S \to \varepsilon$. If $\varepsilon$ is generated by $G$, this would only mean that $M(\varepsilon) = \varepsilon$ should be in $M(L(G))$ (provided that $q_0 \in F$), and it will suffice to add a single rule to the final grammar to complete the construction.

Construct $G' = (\Gamma, N_1' \cup N_2' \cup \{S'\}, R', S')$ with $N_1' = \widehat{Q} \times N \times \widehat{Q}$ and $N_2' = \widehat{Q} \times N \times N \times \widehat{Q}$. For all $\varphi = B_1C_1\& \ldots \& B_mC_m\&\neg D_1E_1\& \ldots \&\neg D_nE_n\&\neg\varepsilon$ appearing as the right-hand side of a rule in $R$, denote

$$(q, \varphi, q') = \mathop{\&}_{1 \leqslant i \leqslant m} (q, B_i, C_i, q')\& \mathop{\&}_{1 \leqslant j \leqslant n} \neg(q, D_j, E_j, q')\&\neg\varepsilon \tag{4.1}$$

The set of rules $R'$ of the new grammar is comprised of the following rules:

$$(q, A, q') \to (q, \varphi, q') \quad (A \to \varphi \in R), \tag{4.2a}$$

$$(q, B, C, q') \to (q, B, q'')(q'', C, q') \quad (q'' \in \widehat{Q}), \tag{4.2b}$$

$$(q, A, \delta(q, a)) \to \lambda(q, a) \quad (A \to a \in R;\ q, \delta(q, a) \in \widehat{Q};\ \lambda(q, a) \neq \varepsilon) \tag{4.2c}$$

$$(q, A, q') \to \varepsilon \quad (\exists w \in L_G(A) : \lambda(q, w) = \varepsilon,\ \delta(q, w) = q') \tag{4.2d}$$

$$S' \to (q_0, S, q_f) \quad (q_f \in F) \tag{4.2e}$$

Now the task is to prove that $G'$ is well-defined, that it actually generates $M(L(G))$, and that it preserves unambiguity. The case of a conjunctive $G$ also requires some remarks. These properties of $G'$ are established in the below claims.

**Lemma 4.2.1.** *The system (3.3.4) corresponding to $G'$ has a strongly unique solution.*

*Proof.* It has to be proved that the solution modulo every finite subword-closed $K \subseteq \Gamma^*$ is unique. The proof is done by induction on $|K|$.

**Basis:** $K = \{\varepsilon\}$. The unique solution $L$ modulo $K$ is defined as follows. For nonterminals in $N_1'$, $L_{(q,A,q')} = \{\varepsilon\}$ if there is a rule $(q, A, q') \to \varepsilon$ in $P'$, and $L_{(q,A,q')} = \varnothing$ otherwise. For nonterminals in $N_2'$, $L_{(q,B,C,q')} = \{\varepsilon\}$ if there exists $q''$ with rules $(q, B, q'') \to \varepsilon$ and $(q'', C, q') \to \varepsilon$, and $L_{(q,B,C,q')} = \varnothing$ otherwise.

**Induction step:** Suppose the solution modulo $K$ is unique, let $x \notin K$, while all proper subwords of $x$ are in $K$. It has to be proved that the solution modulo $K \cup \{x\}$ is also unique. Suppose it is not, and let $L$ and $L'$ be two different solutions modulo $K \cup \{x\}$, with $x \in L_X$ and $x \notin L'_X$ for some nonterminal $X$. We can choose $X$ with minimal $d(q) + d'(q')$.

If $X = (q, A, q') \in N_1$, then $x \in L_{(q,A,q')}$ is generated either by a rule $(q, A, \delta(q, a)) \to \lambda(q, a)$, which means that $x \in L'_{(q,A,q')}$ by the same rule, or by a long rule $(q, A, q') \to (q, \varphi, q')$, which means that the solutions differ on some nonterminal $(q, B, C, q') \in N_2$ also.

If $X = (q, B, C, q') \in N_2$, then there is a rule $(q, B, C, q') \to (q, B, q'')(q'', C, q')\&\neg\varepsilon$ with $x \in L_{(q,B,q'')}L_{(q'',C,q')}$ and $x \notin L'_{(q,B,q'')}L'_{(q'',C,q')}$. Let $x = x_1x_2$, where $x_1 \in L_{(q,B,q'')}$ and $x_2 \in L_{(q'',C,q')}$. If $x_1, x_2 \in K$, then, by the induction hypothesis, $x_1 \in L'_{(q,B,q'')}$ and $x_2 \in L'_{(q'',C,q')}$, and thus $x \in L'_{(q,B,C,q')}$.

Otherwise, let $x_1 = x$ and $x_2 = \varepsilon$. Then $\varepsilon \in L_{(q'',C,q')}$ by a rule $(q'', C, q') \to \varepsilon$, and such a rule exists only if there is a nonempty word $w \in L_G(C)$ with $\lambda(q'', w) = \varepsilon$ and $\delta(q'', w) = q'$. Then $d'(q'') < d'(q'') + |w| \leqslant d'(q')$. Consider the variable $(q, B, q'')$: since $d(q) + d'(q'') < d(q) + d'(q')$, by the assumption, the solutions $L$ and $L'$ do not differ on this variable and thus $x \in L_{(q,B,q'')} = L'_{(q,B,q'')}$. On the other hand, by the basis of induction, $\varepsilon \in L_{(q'',C,q')}$ is equivalent to $\varepsilon \in L'_{(q'',C,q')}$. Therefore, $x \in L'_{(q,B,q'')}L'_{(q'',C,q')} \subseteq L'_{(q,B,C,q')}$ which is a contradiction.

In the other case of $x_1 = \varepsilon$ and $x_2 = x$, it similarly holds that $d(q'') < d(q)$. Then for the variable $(q'', C, q')$ it holds that $d(q'') + d'(q') < d(q) + d'(q')$, so $L'_{(q'',C,q')} = L_{(q'',C,q')}$ by assumption and $x \in L'_{(q'',C,q')}$. At the same time, $\varepsilon \in L'_{(q,B,q'')}$ according to the basis, and $x \in L'_{(q,B,q'')}L'_{(q'',C,q')}$ yields the same contradiction as above. □

**Lemma 4.2.2.** *$x \in L_{G'}\big((q, A, q')\big)$ if and only if $x = \lambda(q, w)$ for some $w \in L_G(A)$ with $\delta(q, w) = q'$.*

*Proof.* The proof is an induction on the lexicographically ordered pairs $(|x|, d(q) + d'(q'))$. Note that the second component is well-defined due to Lemma 4.1.1.

**Basis:** If $x = \varepsilon$, then it can be generated only by a rule $(q, A, q') \to \varepsilon$, and the definition of these rules implies that there is $w \in L_G(A)$ with $\delta(q, w) = q'$

and $\lambda(q,w)=\varepsilon$.

**Induction step:** Let $|x|\geqslant 1$. Assuming the induction hypothesis, the following statement is proved first:

**Claim 4.2.2.1.** *$x\in L_{G'}\big((q,B,C,q')\big)$ if and only if $x=\lambda(q,w)$ for some $w\in L_G(BC)$ with $\delta(q,w)=q'$.*

To prove this, assume first $x\in L_{G'}\big((q,B,C,q')\big)$. Then $x=yz$ with $y\in L_{G'}\big((q,B,q'')\big)$ and $z\in L_{G'}\big((q'',C,q')\big)$ for some $q''$. If $y,z\neq\varepsilon$, then $y=\lambda(q,u)$ and $z=\lambda(q'',v)$ for some $u\in L_G(B)$, $v\in L_G(C)$, $\delta(q,u)=q''$ and $\delta(q'',v)=q'$ by the induction hypothesis. Now $x=\lambda(q,uv)$ with $uv\in L_G(BC)$ and $\delta(q,uv)=q'$.

If $y=\varepsilon$ and $z=x$, then by definition there is such a word $u\in L_G(B)$ that $\lambda(q,u)=\varepsilon$ and $\delta(q,u)=q''$. Since $u$ is nonempty, $d(q'')<d(q'')+|u|\leqslant d(q)$, and thus the induction hypothesis is applicable to $x\in L_{G'}\big((q,C,q'')\big)$ which asserts that there is $v\in L_G(C)$ with $\lambda(q'',v)=x$ and $\delta(q'',u)=q'$. Again, $x=\lambda(q,uv)$ with $uv\in L_G(BC)$ and $\delta(q,uv)=q'$.

The case of $y=x$ and $z=\varepsilon$ is handled similarly.

Assume conversely that $x=\lambda(q,w)$ for some $w\in L_G(BC)$ with $\delta(q,w)=q'$. Then $w=uv$ with $u\in L_G(B)$ and $v\in L_G(C)$, where $u,v\neq\varepsilon$. Let $q''=\delta(q,u)$. If $\lambda(q,u)=\varepsilon$, then $d(q)<d(q'')$ and if $\lambda(q,v)=\varepsilon$, then $d'(q'')<d'(q')$. In both cases, the induction hypothesis is applicable for $\lambda(q,u)\in L_{G'}\big((q,B,q'')\big)$ and $\lambda(q,v)\in L_{G'}\big((q'',C,q')\big)$, as is in the case $\lambda(q,u),\lambda(q,v)\neq\varepsilon$. Thus $\lambda(q,u)\in L_{G'}\big((q,B,q'')\big)$ and $\lambda(q'',v)\in L_{G'}\big((q'',C,q')\big)$, and $x=\lambda(q,u)\lambda(q'',v)\in L_{G'}\big((q,B,C,q')\big)$, which proves Claim 4.2.2.1.

**Claim 4.2.2.2.** *Let $\varphi=B_1C_1\&\ldots\&B_mC_m\&\neg D_1E_1\&\ldots\&\neg D_nE_n\&\neg\varepsilon$ and $A\to\varphi$ be a rule in $P$, assume $x\neq\varepsilon$. Then $x\in L_{G'}\big((q,\varphi,q')\big)$ if and only if $x=\lambda(q,w)$ for some $w\in L_G(\varphi)$ with $\delta(q,w)=q'$.*

Recall that $(q,\varphi,q')$ is a notation for a Boolean expression (4.1). Assume that $x\in L_{G'}\big((q,\varphi,q')\big)$, then by (4.1) $x\in L_{G'}\big((q,B_i,C_i,q')\big)$ for $i=1,\ldots,m$. By Claim 4.2.2.1, there are words $w_i\in L_G(B_iC_i)$ with $\delta(q,w_i)=q'$ and $\lambda(q,w_i)=x$. Since $M$ is injective, all the words $w_1=\ldots=w_m$ are the same, denote this word by $w$. Furthermore, $w\notin L_G(D_jE_j)$ for $j=1,\ldots,n$ again by Claim 4.2.2.1, so $w\in L_G(\varphi)$.

Conversely, let $x=\lambda(q,w)$ for some $w\in L_G(\varphi)$ with $\delta(q,w)=q'$. Now for each $i=1,\ldots,m$, $w\in L_G(B_iC_i)$ and hence, $x\in L_{G'}\big((q,B_i,C_i,q')\big)$ by Claim 4.2.2.1. Similarly, $w\notin L_G(D_jE_j)$ for $j=1,\ldots,n$, and it has to be proved that $x\notin L_{G'}\big((q,D_j,E_j,q')\big)$. Suppose the contrary that $x\in L_{G'}\big((q,D_j,E_j,q')\big)$. Then, by Claim 4.2.2.1, there is $\widetilde{w}\in L_G(D_jE_j)$ with $\delta(q,\widetilde{w})=q'$ and $\lambda(q,\widetilde{w})=x$. Since $M$ is injective, $w=\widetilde{w}$, and hence $w\in L_G(D_jE_j)$ which contradicts the above. Therefore, $x\in L_{G'}\big((q,\varphi,q')\big)$ and Claim 4.2.2.2 is proved.

To begin with the proof of the induction step, assume first $x \in L_{G'}\big((q, A, q')\big)$. If $x$ is generated by a rule $(q, A, \delta(q, a)) \to x$, then, by the definition of these rules, $x = \lambda(q, a)$ and $a \in L_G(A)$. If $x$ is generated by a rule $(q, A, q') \to (q, \varphi, q')$, then by Claim 4.2.2.2 there is $w \in L_G(\varphi)$ with $\delta(q, w) = q'$ and $\lambda(q, w) = x$. It follows that $w \in L_G(A)$ by the rule $A \to \varphi$, which exists by the construction of (4.2a).

Conversely, assume that $x = \lambda(q, w)$ with $w \in L_G(A)$ and $\delta(q, w) = q'$. If $w = a \in \Sigma$, then there is a rule $A \to a \in R$, and $x \in L_{G'}\big((q, A, q')\big)$ by the rule $(q, A, \delta(q, a)) \to \lambda(q, a)$. If $|w| > 1$, then it is generated by a long rule $A \to \varphi$. Now $x \in (q, \varphi, q')$ by Claim 4.2.2.2 and thus $x \in L_{G'}\big((q, A, q')\big)$ by the rule $(q, A, q') \to (q, \varphi, q')$. This ends the proof of the induction step and of the entire Lemma 4.2.2. □

Using this statement, it is easy to show that the constructed grammar generates the image of $L(G)$ under $M$. Consider that $S'$ has the rules $S' \to (q_0, S, q_f)$ for all $q_f \in F$. So, it follows from Lemma 4.2.2 that $x \in L(G')$ if and only if $x = \lambda(q_0, w)$ for some $w \in L_G(S)$ with $\delta(q_0, w) = q_f \in F$, which means $x = M(w)$ for some $w \in L(G)$. This shows that $L(G') = M\big(L(G)\big)$, which proves Theorem 4.2.1 in the case of Boolean grammars.

To see that the same construction works for conjunctive grammars, note that no new negations except $\neg\varepsilon$ are added. If $G$ is conjunctive, then $G'$ can be made conjunctive as well by replacing each conjunct $\neg\varepsilon$ with a reference to a nonterminal generating $\Sigma^+$. So Theorem 4.2.1 has been proved for conjunctive grammars as well.

To complete the proof of Theorem 4.2.1 it remains to consider unambiguous conjunctive and Boolean grammars. It is sufficient to establish the following statement:

**Lemma 4.2.3.** *If $G$ is unambiguous, then $G'$ is unambiguous.*

*Proof.* First the uniqueness of the factorizations is proved. The only conjuncts in the rules of $R'$ with multiple nonterminals are those of the form $(q, B, q'')(q'', C, q')$. So let $x = yz = \tilde{y}\tilde{z}$ be two factorizations of $x$ with $y, \tilde{y} \in L_{G'}\big((q, B, q'')\big)$ and $z, \tilde{z} \in L_{G'}\big((q'', C, q')\big)$. Then, by Lemma 4.2.2 (applied four times), $y = \lambda(q, u)$, $\tilde{y} = \lambda(q, \tilde{u})$, $z = \lambda(q'', v)$ and $\tilde{z} = \lambda(q'', \tilde{v})$ for some $u, \tilde{u} \in L_G(B)$ and $v, \tilde{v} \in L_G(C)$ with $\delta(q, u) = \delta(q, \tilde{u}) = q''$ and $\delta(q'', v) = \delta(q'', \tilde{v}) = q'$. Combining the images in each pair, one obtains $\lambda(q, uv) = yz$ and $\lambda(q, \tilde{u}\tilde{v}) = \tilde{y}\tilde{z}$, that is, $\lambda(q, uv) = \lambda(q, \tilde{u}\tilde{v}) = x$. Since $M$ is injective, this implies $uv = \tilde{u}\tilde{v}$.

Consider that $u, \tilde{u} \in L_G(B)$ and $v, \tilde{v} \in L_G(C)$. Since the original grammar contains a conjunct $BC$, the factorization of any word into $L_G(B)L_G(C)$ must be unique, so $u = \tilde{u}$ and $v = \tilde{v}$. This implies $y = \tilde{y}$ and $z = \tilde{z}$, meaning that the factorization of $x$ is unique.

Next, it is proved that different rules generate disjoint languages. Start with the nonterminals in $N'_1$.

If the empty word is in $L_{G'}\big((q,A,q')\big)$, then it can only be generated by a rule $(q,A,q') \to \varepsilon$, since other rules explicitly deny it by a conjunct $\neg\varepsilon$.

In the case $x \neq \varepsilon$ and $x \in L_{G'}\big((q,A,q')\big)$, there is a $w \in L_G(A)$ with $\delta(q,w) = q'$ and $\lambda(q,w) = x$. There is only one such $w$, since $M$ is injective. There are two cases, namely the case of $x$ generated by $(q,A,\delta(q,a)) \to x$ and the case of $x$ generated by $(q,A,q') \to (q,\varphi,q')$. In the first of these, $w = a \in \Sigma$ and in the second, $w$ is generated by a rule $A \to \varphi$. The cases are clearly disjoint. In the first case, there is only one such rule. In the second case, let $x$ be generated by two different rules $(q,A,q') \to (q,\varphi,q')$ and $(q,A,q') \to (q,\psi,q')$. Then $w$ is generated by two different rules $A \to \varphi$ and $A \to \psi$ by Claim 4.2.2.2 contradicting the unambiguity of $G$.

Then consider the rules for $x \in L_{G'}\big((q,B,C,q')\big)$. If there are two rules for it, say $(q,B,C,q') \to (q,B,q'')(q'',C,q')$ and $(q,B,C,q') \to (q,B,q''')(q''',C,q')$, then there are two corresponding factorizations $x = yz = \tilde{y}\tilde{z}$. Now, by Lemma 4.2.2 four times, there are such words $u, \tilde{u} \in L_G(B)$ and $v, \tilde{v} \in L_G(C)$, that $\delta(q,u) = q''$ and $\delta(q,\tilde{u}) = q'''$, and that $\lambda(q,u) = y$, $\lambda(q'',v) = z$, $\lambda(q,\tilde{u}) = \tilde{y}$ and $\lambda(q''',\tilde{v}) = \tilde{z}$. Now $\lambda(q,uv) = \lambda(q,\tilde{u}\tilde{v})$ and it follows from the injectivity of $M$ that $uv = \tilde{u}\tilde{v}$. Since $G$ in unambiguous, $u = \tilde{u}$ and $v = \tilde{v}$, so that $q'' = q'''$ and the rules are the same. □

**Theorem 4.2.1.** *For every injective gsm $M\colon \Sigma^* \to \Gamma^*$ and for every Boolean (conjunctive, unambiguous Boolean, unambiguous conjunctive) grammar $G$ over $\Gamma$, there exists and can be effectively constructed a Boolean (conjunctive, unambiguous Boolean, unambiguous conjunctive) grammar over $\Gamma$ generating the language $M(L(G))$.*

## 4.3 Boolean grammars and inverse gsm-mappings

In this section, it is proved that the family of languages generated by Boolean grammars is closed under inverse gsm-mappings. First this is proved for weak codings.

So let $h\colon \Sigma^* \to \Gamma^*$ be a weak coding. Denote $\Sigma_0 = \{a \in \Sigma \mid h(a) = \varepsilon\}$ and $\Sigma_1 = \{a \in \Sigma \mid h(a) \in \Gamma\}$.

Let $G = (\Gamma, N, R, S)$ be a Boolean grammar in binary normal form; for technical reasons, assume that $S \to \varepsilon \notin R$. The goal is to construct a grammar $G' = (\Sigma, N', R', S')$ for the language $h^{-1}(L(G))$. Let $N' = N \cup$

$\{S', T\}$ be the set of nonterminals. Then $R'$ contains the following rules:

$$S' \to TST \tag{4.3a}$$

$$T \to a_0 T \mid \varepsilon \quad (\text{for all } a_0 \in \Sigma_0) \tag{4.3b}$$

$$A \to B_1 T C_1 \& \ldots \& B_m T C_m \& \neg D_1 T E_1 \& \ldots \& \neg D_n T E_n \& \neg \varepsilon \quad (\text{for all } A \to B_1 C_1 \& \ldots \& B_m C_m \& \neg D_1 E_1 \& \ldots \& \neg D_n E_n \& \neg \varepsilon \in R) \tag{4.3c}$$

$$A \to a \quad (\text{for all } a \in \Sigma_1 \text{ and } A \to h(a) \in R) \tag{4.3d}$$

If $\varepsilon$ should be in $L(G)$, then a rule $S' \to T$ can be added to $G'$; this case is not considered in the below proof.

It is easy to see that the constructed grammar is well-defined:

**Lemma 4.3.1.** *The system (3.3.4) corresponding to $G'$ has a strongly unique solution.*

*Proof.* The equation for $T$ has a unique solution $T = \Sigma_0^*$ modulo every language. Now, for the entire system, it has to be proved that for every finite subword-closed language $M$, the solution modulo $M$ is unique. This is proved by an induction on $|M|$.

**Basis:** The unique solution modulo $\{\varepsilon\}$ has $L_A = \varnothing$ for all $A \in N$ and $L_T = \{\varepsilon\}$, and accordingly $L_{S'} = \varnothing$.

**Induction step:** Let $M = M_0 \cup \{w\}$, with $w \notin M_0$ and with all subwords of $w$ in $M_0$. By the induction hypothesis, the solution modulo $M_0$ is unique.

Let $(L_{S'}, L_S, \ldots, L_A, \ldots)$ and $(L'_{S'}, L'_S, \ldots, L'_A, \ldots)$ be any two different solutions modulo $M$. If $L_{S'}$ and $L'_{S'}$ are different, then so are $L_S$ and $L'_S$, so it is sufficient to consider the case of the solutions being different on a variable $A \in N$. Assume, without loss of generality, that $w \in L_A$ and $w \notin L'_A$. If $w$ is in $L_A$ by a rule (4.3d), then it is in $L'_A$ by the same rule. Let $w$ be in $L_A$ by a rule (4.3c), while the same rule (4.3c) does not produce $w$ in $L'_A$. Then there is a conjunct $B_i T C_i$ with $w \in L_{B_i} L_T L_{C_i}$ and $w \notin L'_{B_i} L'_T L'_{C_i}$ (or, symmetrically, a conjunct $D_j T E_j$ with $w \notin L_{D_j} L_T L_{E_j}$ and $w \in L'_{D_j} L'_T L'_{E_j}$). Accordingly, $w = uxv$ with $u \in L_{B_i}$, $x \in L_T$ and $v \in L_{C_i}$. Since $\varepsilon \notin L_B, L_C$, $u$ and $v$ are shorter than $w$ and hence are in $M_0$. Then, since the solution modulo $M_0$ is unique, $u \in L'_{B_i}$ and $v \in L'_{C_i}$, and therefore $w \in L'_{B_i} L'_T L'_{C_i}$, which contradicts the assumption. □

Next, note that each nonterminal $A \in N$ in the grammar $G'$ may generate only words that begin and end with symbols from $\Sigma_1$.

**Lemma 4.3.2.** *For every $A \in N$, $L_{G'}(A) \subseteq \Sigma_1 \cup \Sigma_1(\Sigma_0 \cup \Sigma_1)^* \Sigma_1$.*

*Proof.* Suppose the contrary that there is $w \in L_{G'}(A)$ that starts or ends with a symbol from $\Sigma_0$ (the case of $w = \varepsilon$ is clearly impossible by (4.3)).

Let $w$ be the shortest such word. It cannot be generated by a rule (4.3d) because that would imply $w \in \Sigma_1$. So, $w$ must be generated by a rule (4.3c), and by the first conjunct of this rule, $w = uxv$ with $u \in L_{G'}(B)$, $t \in \Sigma_0^*$ and $v \in L_{G'}(C)$. Since $u, v \neq \varepsilon$, both $u$ and $v$ are shorter than $w$. Then, if the first symbol of $w$ is from $\Sigma_0$, then $u$ is a shorter word with the first symbol from $\Sigma_0$, and if $w$ ends with a symbol from $\Sigma_0$, then so does $v$ which is shorter. In both cases, this contradicts the choice of $w$. □

The correctness of the construction is established by the following correspondence of nonterminals in $G$ and $G'$:

**Lemma 4.3.3.** *Let $w \in \Sigma_1 \cup \Sigma_1(\Sigma_0 \cup \Sigma_1)^*\Sigma_1$ and $A \in N$. Then $w \in L_{G'}(A)$ if and only if $h(w) \in L_G(A)$.*

*Proof.* Induction on $|w|$.

**Basis:** $w = a \in \Sigma_1$. Then $a \in L_{G'}(A)$ if and only if there is a rule $A \to a$ in $R'$, which, by (4.3d), exists if and only if $h(a) \in L_G(A)$.

**Induction step:** Let $w \in \Sigma_1(\Sigma_0 \cup \Sigma_1)^*\Sigma_1$. First it is proved that under the induction hypothesis the following statement holds:

**Claim 4.3.3.1.** *$w \in L_{G'}(BTC)$ if and only if $h(w) \in L_G(BC)$.*

If $w \in L_{G'}(BTC)$, there is a factorization $w = uxv$, where $u \in L_{G'}(B)$, $x \in L_{G'}(T)$, and $v \in L_{G'}(C)$. Then $u, v \neq \varepsilon$, and hence $|u|, |v| < |w|$. By the induction hypothesis for $u$ and $v$, $h(u) \in L_G(B)$ and $h(v) \in L_G(C)$. Also, $h(x) = \varepsilon$ by (4.3b), so $h(uxv) = h(u)h(x)h(v) = h(u)h(v) \in L_G(BC)$.

Conversely, if $h(w) \in L_G(BC)$, there is a factorization $w = u'v'$, such that $h(u') \in L_G(B)$ and $h(v') \in L_G(C)$. This implies $u', v' \neq \varepsilon$ and thus $|u'|, |v'| < |w|$. Let $x \in \Sigma_0^*$ be the longest suffix of $u'$ comprised of symbols from $\Sigma_0$, that is, $u' = ux$ with $u \in \Sigma_1 \cup \Sigma_1(\Sigma_0 \cup \Sigma_1)^*\Sigma_1$. Then $h(u) = h(u')$ and, by the induction hypothesis, $u \in L_{G'}(B)$. Similarly, let $y \in \Sigma_0^*$ be the longest prefix of $v'$ containing only symbols from $\Sigma_0$: then $v' = yv$ with $v \in \Sigma_1 \cup \Sigma_1(\Sigma_0 \cup \Sigma_1)^*\Sigma_1$, and the induction hypothesis gives $v \in L_{G'}(C)$. Combining these facts, $w = uxyv \in L_{G'}(BTC)$, which completes the proof of Claim 4.3.3.1.

Next, a similar statement for the right hand sides of the long rules will be established:

**Claim 4.3.3.2.** *$w \in L_{G'}(B_1TC_1\& \ldots \&B_mTC_m\&\neg D_1TE_1\& \ldots \&\neg D_nTE_n\&\neg\varepsilon)$ if and only if $h(w) \in L_G(B_1C_1\& \ldots \&B_mC_m\&\neg D_1E_1\& \ldots \&\neg D_nE_n\&\neg\varepsilon)$.*

Suppose $w \in L_{G'}(B_1TC_1\& \ldots \&B_mTC_m\&\neg D_1TE_1\& \ldots \&\neg D_nTE_n\&\neg\varepsilon)$. This is the case if and only if $w \in L_{G'}(B_iTC_i)$ for all applicable $i$, and $w \notin L_{G'}(D_jTE_j)$ for all applicable $j$. By Claim 4.3.3.1, this is equivalent to $h(w) \in L_G(B_iC_i)$ for all $i$, and $h(w) \notin L_G(D_jE_j)$ for all $j$, which is

equivalent to $h(w) \in L_G(B_1C_1\& \ldots \& B_mC_m\& \neg D_1E_1\& \ldots \& \neg D_nE_n\& \neg\varepsilon)$. Thus, Claim 4.3.3.2 has been proved.

To prove the induction step, consider that $w \in L_{G'}(A)$ is equivalent to the existence of a rule

$$A \to B_1TC_1\& \ldots \& B_mTC_m\& \neg D_1TE_1\& \ldots \& \neg D_nTE_n\& \neg\varepsilon$$

in $R'$. Such a rule exists if and only if $P$ contains a rule

$$A \to B_1C_1\& \ldots \& B_mC_m\& \neg D_1E_1\& \ldots \& \neg D_nE_n\& \neg\varepsilon.$$

On the other hand, by Claim 4.3.3.2, this is equivalent to $h(w) \in L_G(A)$. This completes the proof of Lemma 4.3.3. □

It remains to show that the construction preserves unambiguity, which will complete the proof of Theorem 4.3.1.

**Lemma 4.3.4.** *If $G$ is unambiguous, then $G'$ is unambiguous.*

*Proof.* Consider factorizations of words in $G'$ according to its conjuncts. For the rule (4.3a), if $w \in L_{G'}(TST)$, then $w = xw'y$, where $x, y \in \Sigma_0^*$ and $w' \in \Sigma_1 \cup \Sigma_1(\Sigma_0 \cup \Sigma_1)^*\Sigma_1$, is the unique factorization of $w$ with respect to $L_{G'}(T)$ and $L_{G'}(S)$.

All the other conjuncts that have multiple nonterminals are of the form $BTC$. So, let $w \in L_{G'}(BTC)$. Suppose $w = u_1x_1v_1 = u_2x_2v_2$ with $u_1, u_2 \in L_{G'}(B)$, $x_1, x_2 \in L_{G'}(T)$, and $v_1, v_2 \in L_{G'}(C)$. Then by Lemma 4.3.3, $h(w) = h(u_1v_1) = h(u_2v_2) \in L_G(BC)$. In addition, $u_1, u_2 \in \Sigma_1 \cup \Sigma_1(\Sigma_0 \cup \Sigma_1)^*\Sigma_1$ by Lemma 4.3.2. It follows that if $|u_1| < |u_2|$, then also $|h(u_1)| < |h(u_2)|$. This means there would be two different factorizations of $h(w)$ with respect to $L_G(B)$ and $L_G(C)$, which contradicts the unambiguity of $G$. This proves that conjuncts of $G'$ yield unique factorizations.

If a word $w$ is generated by some rule (4.3c) of $G'$, then, by Claim 4.3.3.2, there is a corresponding rule of $G$ that generates $h(w)$. Because $G$ is unambiguous, $h(w)$ is generated by a unique rule for $A$. From this it follows that the languages generated by distinct rules are disjoint, and furthermore, that $G'$ is unambiguous. □

Now Theorem 4.3.1 can be proved for Boolean grammars.

**Theorem 4.3.1.** *For every weak coding $h : \Sigma^* \to \Gamma^*$ and for every Boolean (conjunctive, unambiguous Boolean, unambiguous conjunctive) grammar $G$ over $\Gamma$, there exists and can be effectively constructed a Boolean (conjunctive, unambiguous Boolean, unambiguous conjunctive) grammar over $\Sigma$ generating the language $h^{-1}(L(G))$.*

*Proof.* First note that no $w \in \Sigma_0^*$ can be in $L_{G'}(S')$ and by assumption $h(w) = \varepsilon \notin L_G(S)$. So, let $w \in \Sigma^*$ be such that $h(w) \neq \varepsilon$. Suppose $h(w) \in L_G(S)$. Then $w$ can be factorized as $w = xw'y$ where $x, y \in \Sigma_0^*$, and $w' \in \Sigma_1 \cup \Sigma_1(\Sigma_0 \cup \Sigma_1)^*\Sigma_1$ with $h(w') = h(w) \in L_G(S)$. By Lemma 4.3.3 and the definition of $T$, this is equivalent to $w' \in L_{G'}(S)$ and $x, y \in L_{G'}(T)$. This holds by the rule (4.3a) if and only if $w \in L_{G'}(S')$, and thus indeed $L(G') = h^{-1}\big(L(G)\big)$.

If $G$ is conjunctive, then no new negations are added and $G'$ is conjuctive, so Theorem 4.3.1 holds for conjunctive grammars as well. □

The above results for the closure of Boolean (conjunctive, unambiguous Boolean, unambiguous conjunctive) languages under injective gsm-mappings and inverse weak codings can now be combined to establish their closure under inverse gsm-mappings according to Proposition 4.1.1:

**Theorem 4.3.2.** *For every gsm $M\colon \Sigma^* \to \Gamma^*$ and for every Boolean (conjunctive, unambiguous Boolean, unambiguous conjunctive) grammar $G$ over $\Gamma$, there exists and can be effectively constructed a Boolean (conjunctive, unambiguous Boolean, unambiguous conjunctive) grammar over $\Sigma$ generating the language $M^{-1}(L(G))$.* □

# Chapter 5

# Morphisms preserving language families

In the previous chapter, the question whether a family of languages is closed under some operation was under discussion. In this chapter, the same problem is considered, but the question asked is under which morphisms a family of languages is closed. A similar characterization was established by Okhotin for the linear variant of conjunctive languages [30]. They are closed under injective morphisms and the trivial morphism mapping everything to the empty word.

Algebraic morphisms constitute the most important and natural class of mappings, at least when dealing with algebraic structures. The explanation of their importance lies in their definition through the algebraic operations of the underlying structure; a mapping is a morphism if and only if it respects the operations. In the monoid of words $(\Sigma^*, \cdot, \varepsilon)$, this means that the image of the concatenation of two words is mapped to the concatenation of the images of those words, so $h\colon \Sigma^* \to \Gamma^*$ is a morphism if and only if $h(uv) = h(u)h(v)$ for all $u, v \in \Sigma^*$.

Injective morphisms, or codes, have the property that different words are mapped to different words. This is a nice property that enables the coding of words into words of a (not necessarily) different alphabet. An important subclass of codes is that of codes of bounded deciphering delay, which consists of codes that can be decoded by finite machines, gsm's. The families of deterministic and LL context-free languages turn out to be closed under a code exactly when it is of bounded deciphering delay.

The proofs use some known closure properties of the families under consideration. The family of deterministic context-free languages is closed under right-quotient with a regular language and intersection with a regular language [5]. Furthermore, it is closed under the inverse gsm-mappings [5].

The LL context-free languages are not closed even under inverse mor-

phisms, a counter-example is given in Example 5.2.1. Therefore, the proofs concerning LL languages require proofs that do not use closure properties.

The last family of languages discussed in this chapter is that of unambiguous context-free languages, which is closed under inverse gsm-mappings [8].

This chapter is based on the article presented at DLT 2012 [26].

## 5.1 Codes

Codes are sets of words that can be used to represent messages. For the codes to be decipherable, the representations of the messages have to be unique. Formally, this means that if $C = \{c_0, c_1, \ldots, c_{k-1}\}$ with $c_i \in \Sigma^*$ is a code and $x = x_1 x_2 \cdots x_m$ and $y = y_1 y_2 \cdots y_n$ are codewords with $x_i, y_j \in C$ and $x = y$, then $m = n$ and $x_i = y_i$ for all $i = 1, \ldots, n$. A code can be represented as a morphism $c\colon \Sigma_k^* \to \Sigma^*$ defined by $c(i) = c_i$, and in this case, a morphism is a code if and only if it is injective. For other alphabets the definition is of course the same, and for example, the morphism $h\colon \{a, b\} \to \{a, b\}$ is a code if and only if $h(a)$ and $h(b)$ are not powers of the same word. This characterizes all two-element codes; $\{x, y\}$ is a code if and only if $x$ and $y$ are not powers of the same word.

In the next section, a connection between deterministic subclasses of context-free languages and a subclass of codes is established. The mentioned subclass of codes is that of codes of bounded deciphering delay:

**Definition 5.1.1.** *A code $h : \Sigma^* \to \Gamma^*$ is of bounded deciphering delay $d \in \mathbb{N}$ if whenever $h(u)$ and $h(v)$ have a common prefix of $d$ symbols, then the first symbols of $u$ and $v$ are equal.*

Intuitively, bounded deciphering delay means that the symbols in the beginning of a coded word can be concluded after some fixed number of symbols from the beginning of the codeword are known. This enables the decoding of codes of bounded deciphering delay by a finite machine: a deterministic gsm. In other words, the inverse mapping of a code of bounded deciphering delay can be computed by a gsm:

**Lemma 5.1.1** ([1, Prop. 5.1.6])**.** *For every code of bounded deciphering delay $h : \Sigma^* \to \Gamma^*$, there exists a deterministic gsm $M : h(\Sigma^*) \to \Sigma^*$, such that $M \circ h$ is the identity mapping on $\Sigma^*$ (that is, $M$ is the inverse mapping of $h$).* □

It is easy to see by induction that the first $k$ symbols of a coded word can be concluded after reading a bounded number of symbols on the codeword as well, for any fixed $k$.

**Lemma 5.1.2.** *If $h\colon \Sigma^* \to \Gamma^*$ is a code with deciphering delay bounded by $d \geqslant 1$, then, for every $k \geqslant 1$, $\mathrm{pref}_{k'}(h(u)) = \mathrm{pref}_{k'}(h(v))$ (where $k' = d + (k-1)\max_{a\in\Sigma} |h(a)|$) implies $\mathrm{pref}_k(u) = \mathrm{pref}_k(v)$.*

*Proof.* Induction on $k$. If $k = 1$, then the first $d$ letters of $h(u)$ and $h(v)$ are the same, and thus the first letters of $u$ and $v$ must also be the same.

For $k > 1$, assume the condition holds for smaller values of $k$. By the same argument as above, the first letters of $u$ and $v$ are the same, and so the words can be written as $u = bu'$ and $v = bv'$ for some $b \in \Sigma$ and $u', v' \in \Sigma^+$. Now $h(u')$ and $h(v')$ are equal on the first $d + (k-1)\max_a |h(a)| - |h(b)| \geqslant d + (k-2)\max_a |h(a)|$ letters. Thus, by the induction assumption, $u'$ and $v'$ have the same $k-1$ first letters. It follows that $u = bu'$ and $v = bv'$ are equal on the $k$ first letters, as claimed. □

If a code is not of bounded deciphering delay, it is said to have unbounded deciphering delay. A code is of unbounded deciphering delay if and only if there is a pair of different right-infinite words with the same image:

$$h(a_0a_1a_2\cdots) = h(b_0b_1b_2\cdots), \quad a_0 \neq b_0.$$

The following Lemma gives a characterization for codes of unbounded deciphering delay used in the next section.

**Lemma 5.1.3.** *Let $h : \Sigma^* \to \Gamma^*$ be a code. Then it is of unbounded deciphering delay if and only if there exist $x, y, z \in \Gamma^+$ with $x, xy, yz, zy \in h(\Sigma^*)$ and $y, z \notin h(\Sigma^*)$.*

*Proof.* Assume $h$ is a code, but not of bounded deciphering delay. Then there exist two infinite words $a_0a_1a_2\ldots$ and $b_0b_1b_2\ldots$, such that $a_0 \neq b_0$, but that the images $h(a_0)h(a_1)h(a_2)\ldots = h(b_0)h(b_1)h(b_2)\ldots$ are the same. Since $h$ is a code, for every prefix $a_0a_1\cdots a_k$ of $a_0a_1a_2\ldots$ there exists a unique prefix $b_0b_1\cdots b_l$ of $b_0b_1b_2\ldots$, such that $h(b_0b_1\cdots b_l)$ is shorter and $h(b_0b_1\cdots b_{l+1})$ is longer than $h(a_0a_1\cdots a_k)$. Therefore one can define a mapping $f : \mathbb{N} \to \mathbb{N}$, so that

$$|h(b_0b_1\cdots b_{f(k)-1})| < |h(a_0a_1\cdots a_k)| < |h(b_0b_1\cdots b_{f(k)})|$$

holds for all $k$.

Since $h(a_0a_1\cdots a_k)$ is a strict prefix of $h(b_0b_1\cdots b_{f(k)})$, there exists a non-empty word $y_k \in \Gamma^+$ satisfying $h(a_0a_1\cdots a_k)y_k = h(b_0b_1\cdots b_{f(k)})$. As $h(b_0b_1\cdots b_{f(k)-1})$ is a strict prefix of $h(a_0a_1\cdots a_k)$, it follows that $|y_k| < |h(b_{f(k)})|$. Consequently, $|y_k| < \max_{a\in\Sigma}(|h(a)|)$ for any $k$, so there are two indices $l < l'$ such that $y_l = y_{l'}$ by the pigeon hole principle.

Now $|h(a_0a_1\cdots a_l)y_l| < |h(a_0a_1\cdots a_{l'})|$, since otherwise

$$|h(b_0b_1\cdots b_{f(l)-1})| < |h(a_0a_1\cdots a_l)| < |h(a_0a_1\cdots a_{l'})| \leqslant |h(b_0b_1\cdots b_{f(l)})|,$$

and thus $f(l') = f(l)$. This would mean that $h(a_0a_1\cdots a_l)y_l = h(a_0a_1\cdots a_{l'})y_{l'}$ and furthermore that $h(a_0a_1\cdots a_l) = h(a_0a_1\cdots a_{l'})$, which is a contradiction.

Denote $y = y_l = y_{l'}$ and define $x, z \in \Gamma^+$ by

$$\begin{aligned} x &= h(a_0 a_1 \cdots a_l) \\ yz &= h(a_{l+1} a_{l+2} \cdots a_{l'}). \end{aligned}$$

Now $x = h(a_0 a_1 \cdots a_l) \in h(\Sigma^*)$, $xy = h(b_0 b_1 \cdots b_{f(l)}) \in h(\Sigma^*)$, $yz = h(a_{l+1} a_{l+2} \cdots a_{l'}) \in h(\Sigma^*)$ and $zy = h(b_{f(l)+1} b_{f(l)+2} \cdots b_{f(l')}) \in h(\Sigma^*)$. At the same time $y \notin h(\Sigma^*)$, since otherwise there would be two different factorizations for $h(a_0 a_1 \cdots a_l) y_l = h(b_0 b_1 \cdots b_{f(l)})$ by the words in $h(\Sigma)$ contradicting the assumption that $h$ is a code. Furthermore, if $z \in h(\Sigma^*)$, then also $zyz \in h(\Sigma^*)$. In this case, it could be factorized in two ways, $z \cdot yz$ and $zy \cdot z$, into words in $h(\Sigma^+)$. This contradiction proves that also $z \notin h(\Sigma^*)$.

Conversely, assume that there exist such words. Now $x \cdot yz \cdot yz \cdot yz \cdots$ and $xy \cdot zy \cdot zy \cdot zy \cdots$ are two different factorizations of the same infinite word, so $h$ is not of bounded deciphering delay. □

The next Lemma sharpens the characterization in Lemma 5.1.3 by presenting some conditions on the words $x$, $y$ and $z$ obtained in the latter Lemma. It basically states that as long as $h$ is a code, these three words must be different from each other to a certain extent.

**Lemma 5.1.4.** *If $h\colon \Sigma^* \to \Gamma^*$ is a code and $x, xy, yz, zy \in h(\Sigma^*)$ for some $x, y, z \in \Gamma^+$ with $y, z \notin h(\Sigma^*)$, then the following conditions hold:*

*i. $x$ and $y$ are not powers of the same word.*

*ii. $x$ and $xy$ (or $x$ and $yx$) are not powers of the same word.*

*iii. $x$ and $yz$ are not powers of the same word.*

*iv. $xy$ and $zy$ are not powers of the same word.*

*v. $xy$ and $yz$ are not powers of the same word, or $yz \neq zy$.*

*Proof.* If $x$ and $y$ are powers of the same word, then $xy = yx$. Since $x, xy \in h(\Sigma^*)$, the word $xyx \in h(\Sigma^*)$ could be factorized into code words as $x \cdot yx$ or as $xy \cdot x$ implying $y \in h(\Sigma^*)$. This is a contradiction.

The words $x$ and $xy$ ($x$ and $yx$) are powers of the same word if and only if $x^k = (xy)^l$ ($x^k = (yx)^l$) for some $k, l > 0$. But then the words $x$ and $y$ would be powers of the same word, contradicting the above.

If $x$ and $yz$ are powers of the same word, then $x^k = (yz)^l$ for some $k$ and $l$. By the assumptions $xy \in h(\Sigma^*)$, and thus also $x^{k-1}xy \in h(\Sigma^*)$. Then $x^{k-1}xy = x^k y = (yz)^l y$, so $(yz)^l y(zy)^l \in h(\Sigma^*)$ as $(yz)^l y \in h(\Sigma^*)$ and $(zy)^l \in h(\Sigma^*)$. The word $(yz)^l y(zy)^l \in h(\Sigma^*)$ can be factorized as $(yz)^l y \cdot (zy)^l$ or as $(yz)^l \cdot y(zy)^l$, which implies $y \in h(\Sigma^*)$ contradicting the assumption.

If $xy$ and $zy$ are powers of the same word, then $(xy)^k = (zy)^l$ for some $k$ and $l$. By the assumptions $x, xy \in h(\Sigma^*)$, and thus also $(xy)^{k-1}x \in h(\Sigma^*)$. Now $(xy)^{k-1}x = (zy)^{l-1}z$, by the assumption $(xy)^k = (zy)^l$. Furthermore, $y(xy)^{k-1}x = (yz)^l \in h(\Sigma^*)$. Now the word $(xy)^{k-1}xy(xy)^{k-1}x$ can be factorized as $(xy)^{k-1}x \cdot y(xy)^{k-1}x$ and as $(xy)^{k-1}xy \cdot (xy)^{k-1}x$. Again, this implies $y \in h(\Sigma)^*$, a contradiction.

If $xy$ and $yz$ are powers of the same word and $yz = zy$, then $xy$ and $zy$ would be powers of the same word as well, contradicting the previous statement.

□

Although the words $x$ and $yz$ (or $xy$ and $zy$) cannot be powers of the same word, the words $xy$ and $yz$ can, as shown by the following example:

**Example 5.1.1.** *The code $h\colon \{a,b,c\}^* \to \{a,b\}^*$ defined by $h(a) = ababa$, $h(b) = baaba$ and $h(c) = ababaab$ is of unbounded deciphering delay. One can choose $x = ababaab$, $y = aba$ and $z = ba$. They satisfy the conditions $x, xy, yz, zy \in h(\{a,b,c\}^*)$ and $y, z \notin h(\{a,b,c\}^*)$, while $xy = yzyz$. In this case, $yz = ababa$ is different from $zy = baaba$.* □

The next result gives further conditions on the form of the words $x$, $y$ and $z$ in Lemma 5.1.3. It shows that their pre-images with respect to $h$ must also be distinguishable from each other in certain occasions.

**Lemma 5.1.5.** *Let $h\colon \Sigma^* \to \Gamma^*$ be a code, and let the words $w, w', u, v \in \Sigma^+$ be encoded as $h(w) = x$, $h(w') = xy$, $h(u) = yz$ and $h(v) = zy$ for some $x, y, z \in \Gamma^+$ with $y, z \notin h(\Sigma^*)$. Then:*

*i. Neither of the words $w$ and $w'$ is a prefix of the other, and, in particular, their longest common prefix is of length less than $\min(|w|, |w'|)$;*

*ii. The longest common prefix of the infinite words $w^\omega$ and $u^\omega$ is of length less than $|w| + |u|$;*

*iii. The longest common prefix of $(w')^\omega$ and $v^\omega$ is of length less than $|w'| + |v|$;*

*iv. The longest common prefix of $w'v^\omega$ and $u^\omega$ is of length at most $|w'|+|v|$.*

*Proof.* Firstly, $w'$ cannot be a prefix of $w$, as the image of $w$ is a prefix of the image of $w'$. Secondly, if $w$ would be a prefix of $w'$, that is, $w' = w\hat{w}$ for some $\hat{w} \in \Sigma^*$, then $xy = h(w)h(\hat{w})$ and $x = h(w)$ imply $y = h(\hat{w}) \in h(\Sigma^*)$, contradicting the assumption that $y \notin h(\Sigma^*)$. Thus, the length of the common prefix of $w^\omega$ and $w'v^\omega$ must be less than $\min(|w|, |w'|)$.

If $w^\omega$ and $u^\omega$ have a common prefix of length $|w| + |u|$, then $uw^\omega$ and $u^\omega$ have a common prefix of length $|w| + 2|u|$, and $w^\omega$ and $wu^\omega$ have a common

prefix of length $2|w|+|u|$. Consequently, $uw^\omega$ and $wu^\omega$ have a common prefix of length $|w|+|u|$, so $uw = wu$. As the words $w$ and $u$ commute, they are powers of the same word, and therefore, $h(w) = x$ and $h(u) = yz$ would also be powers of the same word, contradicting Lemma 5.1.4.

The case of $(w')^\omega$ and $v^\omega$ can be handled in the same way as the case of $w^\omega$ and $u^\omega$. The contradiction with Lemma 5.1.4 just comes from $xy$ and $zy$ instead of $x$ and $yz$.

In the last case, if $u^\omega$ and $w'v^\omega$ have a common prefix of length $|w'|+|v|$, then $h(u^\omega) = (yz)^\omega$ and $h(w'v^\omega) = x(yz)^\omega$ have a common prefix of length $|xy| + |yz|$. Similarly to the above, $yzx(yz)^\omega$ would have the same prefix of length $|x|+|yz|$ with $x(yz)^\omega$. This implies $xyz = yzx$, which is a contradiction as $x$ and $yz$ are not powers of the same word by Lemma 5.1.4. □

## 5.2 The families DetCF and LL and bounded deciphering delay

In this section, it is proved that the family of deterministic context-free languages is closed under those codes that have a finite deciphering delay.

The language

$$\{a^n b^n \mid n \geqslant 0\} \cup \{a^n b^{2n} \mid n \geqslant 0\}$$

is not deterministic, as the PDA would have to guess if the number of $a$'s corresponds to the number of $b$'s or the number of $b$'s multiplied by two before reaching the border between $a$'s and $b$'s.

The code $h(a) = a, h(b) = ab, h(c) = bb$ is of unbounded delay. Deciphering a codeword $ab^n$ from left to right needs a similar nondeterministic guess as in the case of recognizing the language above by a pushdown automaton. The pre-image is

$$ac^{(\frac{n}{2})}$$

if $n$ is even and

$$bc^{(\frac{n-1}{2})}$$

if $n$ is odd, forcing a nondeterministic guess after reading the first $a$. Now, consider the language $L = \{a^n c^n \mid n \geqslant 0\} \cup \{a^{n-1} b c^{2n} \mid n \geqslant 1\}$ which is clearly deterministic context-free. However, its image under $h$ is $h(L) = \{a^n b^{2n} \mid n \geqslant 0\} \cup \{a^n b^{4n+1} \mid n \geqslant 1\}$, and this language is not deterministic context-free.

This is not a coincidence, as there is a connection between codes of bounded deciphering delay and deterministic context-free languages. First of all, deterministic context-free languages are closed under these codes which follows from their decipherability by deterministic gsm-mappings:

**Lemma 5.2.1.** *Let $h : \Sigma^* \to \Gamma^*$ be a code of bounded deciphering delay and $L \subseteq \Sigma^*$ a deterministic context-free language. Then the language $h(L)$ is deterministic context-free.*

*Proof.* Let $L$ be a deterministic context-free language. By Lemma 5.1.1, the inverse mapping of $h$ is computed by a gsm $M\colon h(\Sigma^*) \to \Sigma^*$. The family of deterministic context-free languages is closed under inverse gsm-mappings [5, Thm. 3.2], and thus, $h(L) = M^{-1}(L)$ is deterministic context-free as was claimed. □

Conversely, if a code is of unbounded deciphering delay, then it does not preserve deterministic context-free languages:

**Lemma 5.2.2.** *Let $h : \Sigma^* \to \Gamma^*$ be a code of unbounded deciphering delay. Then there exists a deterministic context-free language $L \subseteq \Sigma^*$, such that $h(L)$ is not deterministic context-free.*

*Proof.* By Lemma 5.1.3, there exist $x, y, z \in \Gamma^*$ with $x, xy, yz, zy \in h(\Sigma^*)$ and $y, z \notin h(\Sigma^*)$. Let $w, w', u, v \in \Sigma^*$ be the words with $h(w) = x$, $h(w') = xy$, $h(u) = yz$ and $h(v) = zy$ and consider the language

$$L = \{w^n u^n \mid n \geqslant 1\} \cup \{w^{n-1} w' v^{2n} \mid n \geqslant 1\}$$

which is generated by the grammar

$$\begin{aligned} S &\to A \mid B \\ A &\to wAu \mid \varepsilon \\ B &\to wBvv \mid w'vv. \end{aligned}$$

A deterministic pushdown automaton simulating this grammar would push $w$'s into the stack until it sees either $u$ or $w'$. For that, it should be able to notice the border between $w$ and $u$ in $w^n u^n$, and the border between $w$ and $w'$ in $w^n w' v^{2n}$, and be able to distinguish these cases from each other. This can be done, as the lengths of the common prefices of $w^\omega$, $u^\omega$ and $w'v^\omega$ are bounded by $\max(|w| + |u|, |w'| + |u|)$, which follows from Lemma 5.1.5.

So the border and the type of the words can be distinguished deterministically after reading the $\max(|w| + |u|, |w'| + |u|)$ letters when the border has been passed, and afterwards a deterministic pushdown automaton may start popping the stack, reading $u$ or $vv$ depending on the case.

The image of $L$ under $h$ is

$$h(L) = \{x^n (yz)^n \mid n \geqslant 1\} \cup \{x^n (yz)^{2n} y \mid n \geqslant 1\}.$$

Let $M : a^+b^+(\varepsilon \cup c) \to \Gamma^*$ be a gsm-mapping with $M(a^k b^l c^m) = x^k (yz)^l y^m$ for $k, l \geqslant 1$ and $m \in \{0, 1\}$.

The pre-image $M^{-1}\big(h(L)\big)$ is easily figured out as $M$ is injective. To prove the injectivity, assume that two distinct words in $a^+b^+(\varepsilon \cup c)$ are mapped to the same word. It turns out that in this case, $x$ and $yz$ would be powers of the same word, contradicting Lemma 5.1.4. There are three possible cases:

If $x^{k_1}(yz)^{l_1} = x^{k_2}(yz)^{l_2}$ for different exponents, then $x$ and $yz$ are powers of the same word.

The case $x^{k_1}(yz)^{l_1}y = x^{k_2}(yz)^{l_2}y$ can be handled by the same argument by removing $y$ from the end.

In the third possible case $x^{k_1}(yz)^{l_1} = x^{k_2}(yz)^{l_2}y$, consider the suffices of length $|yz|$. Since $l_1, l_2 > 0$, the suffix on the left-hand side is $yz$ and $zy$ on the right-hand side. They are equal so $yz = zy$, and thus, $y$ and $z$ are powers of the same word $z'$. Substituting $y$ and $z$ by powers of $z'$ in the equation results in an equation $x^{k_1}z'^{l'_1} = x^{k_2}z'^{l'_2}$ which implies that $x$ and $z'$, and thus also $x$ and $yz$, are powers of the same word.

So $M$ is injective, and thus $M^{-1}\big(h(L)\big) = \{a^n b^n | n \geqslant 1\} \cup \{a^n b^{2n} c | n \geqslant 1\}$. It remains to be proved that this is not a deterministic context-free language. This can be seen by taking a right quotient with $\{\varepsilon, c\}$ and intersecting with $a^+b^+$. The result is a known non-deterministic language $\{a^n b^n \mid n \geqslant 0\} \cup \{a^n b^{2n} \mid n \geqslant 0\}$, while deterministic languages are closed under right quotients and intersections with regular languages [5]. □

The language $L = \{ac^n a^n | n \geqslant 0\} \cup \{bc^n a^{2n} | n \geqslant 0\}$ is LL(1) context-free, but its image $h(L) = \{ab^{2n}a^n \mid n \geqslant 0\} \cup \{ab^{2n+1}a^{2n} \mid n \geqslant 0\}$ is not LL($k$) context-free for any $k$, where $h$ is the code $h(a) = a, h(b) = ab, h(c) = bb$ of unbounded deciphering delay. It actually is the case that the LL languages are closed under a code if and only if it is of bounded delay. However, as the next example shows, the family is not closed under inverse gsm-mappings or even morphisms, so a separate argument for the closure is needed.

**Example 5.2.1.** *Let $\Sigma = \{1, 2, 3, 4, 5, 6\}$ and $\Gamma = \{a, b, c, d\}$, and define a morphism $h\colon \Sigma^* \to \Gamma^*$ by $h(1) = a$, $h(2) = bc$, $h(3) = cb$, $h(4) = ab$, $h(5) = d$, $h(6) = cd$. Then the language $L = \{a^n(bc)^n d \mid n \geqslant 0\}$ is LL(1) context-free, while its inverse image*

$$h^{-1}(L) = \{1^n 2^n 5 \mid n \geqslant 0\} \cup \{1^{n-1} 43^{n-1} 6 \mid n \geqslant 1\}$$

*is not LL($k$) for any $k$.* □

However, the bounded deciphering delay guarantees that $k$ symbols of the original word can be known after reading a bounded number of symbols (depending on the delay bound and $k$) of the image word.

**Lemma 5.2.3.** *Let $h : \Sigma^* \to \Gamma^*$ be a code of deciphering delay bounded by $d$. Then, for every LL($k$) context-free language $L \subseteq \Sigma^*$, the language $h(L)$ is an LL($k'$) context-free language, where $k' = d + (k-1)\max_{a\in\Sigma} |h(a)|$.*

*Proof.* Let $G = (\Sigma, N, P, S)$ be an LL($k$) context-free grammar generating $L$, and extend $h$ to a code $h\colon (\Sigma \cup N) \to (\Gamma \cup N)$ by setting $h(A) = A$ for all $A \in N$. Define the grammar $G' = (\Gamma, N, P', S)$ with the set of rules

$P' = \{A \to h(\alpha) \mid A \to \alpha \in P\}$. Then $L(G') = h(L(G))$ and, more generally, $L_{G'}(h(\alpha)) = h(L_G(\alpha))$ for each $\alpha \in (\Sigma \cup N)^*$. It remains to prove that $G'$ is LL($k'$).

Let $S \to^* h(x)Ah(\beta)$ in $G'$, where $A \to \alpha_1$, $A \to \alpha_2 \in P$, and assume that $h(w_1) \in L_{G'}(h(\alpha_1\beta))$, $h(w_2) \in L_{G'}(h(\alpha_2\beta))$ and $\mathrm{pref}_{k'}(h(w_1)) = \mathrm{pref}_{k'}(h(w_2))$. Then, by Lemma 5.1.2, $\mathrm{pref}_k(w_1) = \mathrm{pref}_k(w_2)$. Consider the corresponding derivation $S \to^* xA\beta$ in $G$, with $w_1 \in L_G(\alpha_1\beta)$, $w_2 \in L_G(\alpha_2\beta)$. Since $G$ is LL($k$), this implies that $\alpha_1 = \alpha_2$ and further that $h(\alpha_1) = h(\alpha_2)$. □

A similar non-closure result as for deterministic context-free languages holds for LL context-free languages as well. However, due to the limited closure properties of this language family, the proof involves a lengthy low-level analysis of a parser's computation, and only a short sketch of a proof is presented here.

For every LL($k$) grammar, there exists a partial function $T\colon N \times \Sigma^{\leqslant k} \to P$, such that whenever $S \to^* xA\beta$, $A \to \alpha \in P$ and $w \in L_G(\alpha\beta)$, the rule $A \to \alpha$ is given by $T(A, \mathrm{pref}_k(w))$.

For every LL($k$) grammar $G$, the membership of a given input word $w$ in $L(G)$ is recognized by an LL($k$) parser as follows. The parser's configurations are pairs of the form $(u, \eta)$, where $u \in \Sigma^*$ is an unread suffix of the input word and $\eta \in (\Sigma \cup N)^*$ is the contents of the parser's stack. In a configuration $(u, A\eta)$, the parser looks up $T(A, \mathrm{pref}_k(u))$, and if it contains a rule $A \to \alpha$, the parser enters the configuration $(u, \alpha\eta)$. In a configuration $(au, a\eta)$, the next configuration is $(u, \eta)$. In all other cases, an error is reported. The configuration $(\varepsilon, \varepsilon)$ is accepting.

**Lemma 5.2.4.** *For every code $h : \Sigma^* \to \Gamma^*$ of unbounded deciphering delay, there exists a language $L \subseteq \Sigma^*$ generated by a simple linear context-free grammar, such that $h(L)$ is not an LL($k$) context-free language for any $k$.*

*Proof.* The words $x, y, z \in \Gamma^*$ with $x, xy, yz, zy \in h(\Sigma^*)$ and $y, z \notin h(\Sigma^*)$ exist by Lemma 5.1.3. Consider $w, w', u, v \in \Sigma^*$ with $h(w) = x$, $h(w') = xy$, $h(u) = yz$ and $h(v) = zy$. The language

$$L = \{wu^n w^n \mid n \geqslant 0\} \cup \{w'v^n(w')^n \mid n \geqslant 0\}$$

is generated by the following context-free grammar

$$\begin{aligned} S &\to wA \mid w'B \\ A &\to uAw \mid \varepsilon \\ B &\to vBw' \mid \varepsilon. \end{aligned}$$

It is claimed that this grammar is LL($k$) for $k = |u|+|w|$. By Lemma 5.1.5(i), the choice between the rules $S \to wA$ and $S \to w'B$ can be made based on

the first $\min(|w|, |w'|)$ symbols of the input. Choosing between the rules $A \to uAw$ and $A \to \varepsilon$ requires the parser to distinguish between $u^\omega$ and $w^\omega$ (which can be done using the first $|u| + |w|$ symbols according to Lemma 5.1.5(ii)), as well as between $u^i w^\omega$ and $w^\omega$ for any $i \geqslant 1$; the latter can be done using $|u| + |w|$ symbols, because if the first $|u| + |w|$ symbols of $u^i w^\omega$ and $w^\omega$ are the same, then either $|u| + |w| < |u^i|$ and they can be distinguished the same way as $u^\omega$ and $w^\omega$, or $|u^i| \leqslant |u| + |w|$. In the latter case, $u^i$ is a prefix of $w^\omega$, and thus $w^\omega$ and $u^{2i} w^\omega$ have a common prefix of length $|u|+|v|$ as well, and repeating this would eventually lead to $w^\omega$ and $u^j w^\omega$ with $|u| + |w| < |u^j|$ having a common prefix of length $|u| + |w|$, a contradiction. Similarly, Lemma 5.1.5(iii) implies that a parser can choose between the rules $B \to vBw'$ and $B \to \varepsilon$ using $|v| + |w'|$ symbols of the input.

The image of $L$ is the following language:

$$h(L) = \{x(yz)^n x^n \mid n \geqslant 0\} \cup \{x(yz)^n y(xy)^n \mid n \geqslant 0\}.$$

It is not $\mathrm{LL}(k)$ for any $k$. A precise proof for the non-existence of an LL grammar for $h(L)$ would be rather long and technical, and is omitted. Intuitively, the parser just sees

$$xyzyzyzyzyzyzyzyzyz \ldots$$

in the beginning and cannot decide which rule is producing them without knowing how the word is going to end. □

## 5.3 Non-codes preserving LL, DetCF and UnambCF

In this section, the non-codes preserving LL, deterministic and unambiguous context-free languages are characterized, finalising the characterization for morphisms in general. The non-codes preserving these families map everything into the submonoid $w^*$ generated by some word $w$.

It is easy to see that the families are closed under these morphisms, as the image of any context-free language under them is regular:

**Lemma 5.3.1.** *Let $h : \Sigma^* \to \Gamma^*$ be a morphism satisfying $h(\Sigma) \subseteq w^*$ for some $w \in \Gamma^*$. Then, for every context-free language $L$, the language $h(L)$ is regular.*

*Proof.* By Parikh's theorem, there exists a regular language $L'$ letter equivalent to $L$. Since regular languages are closed under morphisms and $h(L) = h(L')$, it follows that $h(L)$ is regular. □

If the image of a non-code is not included in some $w^*$, then there exists an LL(1) language with an inherently ambiguous image. Thus, none of the families are preserved.

**Lemma 5.3.2.** *Let $h : \Sigma^* \to \Gamma^*$ be a morphism that is not a code and for which $h(a)$ and $h(b)$ are not powers of the same word for some $a, b \in \Sigma$. Then there exists a language $L \subseteq \Sigma^*$ generated by an LL(1) grammar, such that $h(L)$ is an inherently ambiguous context-free language.*

*Proof.* Since $h$ is not a code, there exist two distinct words $u, v \in \Sigma^*$ with $h(u) = h(v)$. It can be assumed that $u$ and $v$ differ on the first letter, or in the case one of them, say $u$, is empty, that $v$ is of length one and $v \neq a$. Define

$$L = \{ua^n b^i a^n \mid i, n \geqslant 1\} \cup \{va^i b^n a^n \mid i, n \geqslant 1\}.$$

It is generated by the grammar

$$\begin{aligned} S &\to uaS_1a \mid vaS_2 \\ S_1 &\to aS_1a \mid bA \\ S_2 &\to aS_2 \mid bBa \\ A &\to bA \mid \varepsilon \\ B &\to bBa \mid \varepsilon \end{aligned}$$

For each non-terminal symbol, the words generated by different rules differ on the first symbol. Therefore the grammar, and thus also $L$, is LL(1).

The image of $L$ is

$$h(L) = \{xh(a)^n h(b)^i h(a)^n \mid i, n \geqslant 1\} \cup \{xh(a)^i h(b)^n h(a)^n \mid i, n \geqslant 1\},$$

where $x = h(u) = h(v)$.

It remains to be proved that $h(L)$ is inherently ambiguous.

For this, define a gsm-mapping $M : a^+b^+a^+ \to \Gamma^*$ such that

$$M(a^k b^l a^m) = xh(a)^k h(b)^l h(a)^m).$$

If

$$M(a^{k_1} b^{l_1} a^{m_1}) = M(a^{k_2} b^{l_2} a^{m_2})$$

for some exponents, then $h(a)^{k_1} h(b)^{l_1} h(a)^{m_1} = h(a)^{k_2} h(b)^{l_2} h(a)^{m_2}$. By assumption, $h(a)$ and $h(b)$ are not powers of the same word, so $k_1 = k_2$, $l_1 = l_2$ and $m_1 = m_2$. It follows that $M$ is injective and $M^{-1}\big(h(L)\big) = \{a^n b^i a^n | i, n \geqslant 1\} \cup \{a^i b^n a^n \mid i, n \geqslant 1\}$ which is an inherently ambiguous language. As unambiguous context-free languages are closed under inverse gsm-mappings, it follows that $h(L)$ is inherently ambiguous as well. □

Combining the results in this and the previous section results in the following characterization for morphisms preserving deterministic or LL context-free languages:

**Theorem 5.3.1** ([26]). *Let $h\colon \Sigma^* \to \Gamma^*$ be a morphism. Then the family of deterministic (LL) context-free languages is closed under $h$ if and only if $h$ is a code of bounded deciphering delay or $h(\Sigma) \subseteq w^*$ for some word $w \in \Gamma^*$.*

Furthermore, there is the characterization for morphisms preserving the unambiguous context-free languages:

**Theorem 5.3.2** ([26]). *Let $h\colon \Sigma^* \to \Gamma^*$ be a morphism. Then the family of unambiguous context-free languages is closed under $h$ if and only if $h$ is a code or $h(\Sigma) \subseteq w^*$ for some word $w \in \Gamma^*$.*

# Bibliography


[1] J. Berstel, D. Perrin, C. Reutenauer, *Codes and Automata*, Cambridge University Press, 2010.

[2] V. Bruyère, "Maximal codes with bounded deciphering delay", *Theoretical Computer Science*, 84:1 (1991), 53–76.

[3] J. H. Conway, *Regular Algebra and Finite Machines*, Chapman and Hall (1971).

[4] K. Culik II, J. Gruska, A. Salomaa, "Systolic trellis automata: stability, decidability and complexity", *Information and Control*, 71 (1986) 218–230.

[5] S. Ginsburg, S. A. Greibach, "Deterministic context-free languages", *Information and Control*, 9:6 (1966), 620–648.

[6] S. Ginsburg, H. G. Rice, "Two families of languages related to ALGOL", *Journal of the ACM*, 9 (1962), 350–371.

[7] S. Ginsburg, G. Rose, "Operations which preserve definability in languages", *Journal of the ACM*, 10:2 (1963), 175–195.

[8] S. Ginsburg, J. Ullian, "Preservation of unambiguity and inherent ambiguity in context-free languages", *Journal of the ACM*, 13:3 (1966), 364–368.

[9] M. A. Harrison, *Introduction to Formal Language Theory*, Addison-Wesley, 1978.

[10] T. N. Hibbard, J. Ullian, "The independence of inherent ambiguity from complementedness among context-free languages", *Journal of the ACM*, 13:4 (1966), 588–593.

[11] O. H. Ibarra, S. M. Kim, "Characterizations and computational complexity of systolic trellis automata", *Theoretical Computer Science*, 29 (1984), 123–153.

[12] A. Jeż, "Conjunctive grammars can generate non-regular unary languages", *International Journal of Foundations of Computer Science*, 19:3 (2008), 597–615.

[13] A. Jeż, A. Okhotin "On the computational completeness of equations over sets of natural numbers", *Automata, Languages and Programming* (ICALP 2008, Reykjavik, Iceland, July 6-13 2008), part II, LNCS 5126, 63-74.

[14] A. Jeż, A. Okhotin, "Equations over sets of natural numbers with addition only", *26th Annual Symposium on Theoretical Aspects of Computer Science* (STACS 2009, Freiburg, Germany, 26–28 February 2009), Dagstuhl Seminar Proceedings 09001, 577–588.

[15] A. Jeż, A. Okhotin, "Least and greatest solutions of equations over sets of integers", *Mathematical Foundations of Computer Science* (MFCS 2010, Brno, Czech Republic, 23-27 August 2010), LNCS 6281, 441-452.

[16] A. Jeż, A. Okhotin, "Representing hyper-arithmetical sets by equations over sets of integers", *Theory of Computing Systems*, 51:2 (2012), 196–228.

[17] L. Kari, "On language equations with invertible operations", *Theoretical Computer Science*, 132 (1994), 129–150.

[18] M. Kunc, "The power of commuting with finite sets of words", *STACS 2005*, 569–580.

[19] M. Kunc, "The power of commuting with finite sets of words", *Theory of Computing Systems*, 40:4 (2007), 521–551.

[20] M. Kunc, A. Okhotin "Language equations", in: J.-E. Pin (Ed.) *Automata: from Mathematics to Applications*, to appear.

[21] T. Lehtinen, "On equations $X+A=B$ and $(X+X)+C=(X-X)+D$ over sets of numbers" *Mathematical Foundations of Computer Science*, (MFCS 2012, Bratislava, Slovakia, August 26–31 2012), LNCS 7464, 615–629.

[22] T. Lehtinen, A. Okhotin, "Boolean Grammars Are Closed Under Inverse Gsm Mappings", *TUCS Technical Reports* 911, Turku Centre for Computer Science (2008).

[23] T. Lehtinen, A. Okhotin, "Boolean grammars and gsm mappings", *International Journal of Foundations of Computer Science*, 21:5 (2010), 799–815.

[24] T. Lehtinen, A. Okhotin, "On language equations $XXK = XXL$ and $XM = N$ over a unary alphabet" *Developments in Language Theory* (DLT 2010, London, Ontario, Canada, 17–20 August 2010), LNCS 6224, 291–302.

[25] T. Lehtinen, A. Okhotin, "On equations over sets of numbers and their limitations", *International Journal of Foundations of Computer Science*, 22:2 (2011), 377–393.

[26] T. Lehtinen, A. Okhotin, "Homomorphisms preserving deterministic context-free languages", *Developments in Language Theory*, (DLT 2012, Taipei, Taiwan, August 14–17 2012), LNCS 7410, 154–165.

[27] A. Okhotin, "Decision problems for language equations with Boolean operations", *Automata, Languages and Programming*, (ICALP 2003, Eindhoven, The Netherlands, 30 June–4 July 2003), LNCS 2719, 239-251.

[28] A. Okhotin, "On the equivalence of linear conjunctive grammars to trellis automata", *RAIRO Informatique Théorique et Applications*, 38:1 (2004), 69–88.

[29] A. Okhotin, "Unresolved systems of language equations: expressive power and decision problems", *Theoretical Computer Science*, 349:3 (2005), 283–308.

[30] A. Okhotin, "Homomorphisms preserving linear conjunctive languages", *Journal of Automata, Languages and Combinatorics*, 13:3–4 (2008), 299–305.

[31] A. Okhotin, "Decision problems for language equations", *Journal of Computer and System Sciences*, 76:3–4 (2010), 251–266.

[32] A. Okhotin, "Conjunctive and Boolean grammars: the true general case of the context-free grammars", submitted, (2012).

[33] R. Parikh, A. K. Chandra, J. Y. Halpern, A. R. Meyer, "Equations Between Regular Terms and an Application to Process Logic", *SIAM J. Comput.*, 14(4): 935–942 (1985)

[34] D. J. Rosenkrantz, R. E. Stearns, "Properties of deterministic top-down grammars", *Information and Control*, 17 (1970), 226–256.

[35] A. Tarski, "A lattice-theoretical fixpoint theorem and its applications", *Pacific J. Math.*, Volume 5, Number 2 (1955), 285–309.

[36] V. Terrier, "On real-time one-way cellular array", *Theoretical computer science*, 141 (1995), 331–335.

# Turku Centre for Computer Science
# TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspnäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, $Z_4$-Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations Between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen,** On the NTRU Cryptosystem
64. **Saeed Salehi**, Varieties of Tree Languages
65. **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
66. **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
67. **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
68. **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
69. **Luigia Petre**, Modeling with Action Systems
70. **Lu Yan**, Systematic Design of Ubiquitous Systems
71. **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
72. **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
73. **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
74. **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
75. **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
76. **Jussi Salmi**, Improving Data Analysis in Proteomics
77. **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
78. **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
79. **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
80. **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
81. **Luka Milovanov**, Agile Software Development in an Academic Environment
82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Mehtods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming

128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performace Estimation
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
149. **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
150. **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
151. **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
152. **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
153. **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
154. **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip
155. **Mikko Pelto**, On Identifying and Locating-Dominating Codes in the Infinite King Grid
156. **Anton Tarasyuk**, Formal Development and Quantitative Verification of Dependable Systems
157. **Muhammad Mohsin Saleemi**, Towards Combining Interactive Mobile TV and Smart Spaces: Architectures, Tools and Application Development
158. **Tommi J. M. Lehtinen**, Numbers and Languages

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

**University of Turku**
*Faculty of Mathematics and Natural Sciences*
- Department of Information Technology
- Department of Mathematics and Statistics

*Turku School of Economics*
- Institute of Information Systems Science

**Åbo Akademi University**
*Division for Natural Sciences and Technology*
- Department of Information Technologies

ISBN 978-952-12-2849-0
ISSN 1239-1883

Tommi J. M. Lehtinen

# Numbers and Langauges

Tommi J. M. Lehtinen

## Numbers and Languages

Tommi J. M. Lehtinen

Numbers and Languages