Algebraic Closures in Complexity Theory

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Bayerischen Julius-Maximilians-Universität Würzburg

vorgelegt von

Elmar Böhler

aus Kempten

Würzburg, 2005

Eingereicht am: 26.9.2005
bei der Fakultät für Mathematik und Informatik
1. Gutachter: Prof. Dr. Klaus W. Wagner
2. Gutachter: Prof. Dr. Heribert Vollmer
Tag der mündlichen Prüfung: 16.12.2005

Auf keinen Fall wäre es mir möglich gewesen diese Arbeit ohne die viele Hilfe anzufertigen, die freundliche Menschen mir zuteil werden ließen. Insbesondere gilt mein Dank meinem Doktorvater Klaus Wagner, der seinen Mitarbeitern nicht nur das ideale Forschungsumfeld zu schaffen versteht, sondern mit seinen vielen guten Ideen dafür sorgt, dass die Gruppe stets an spannenden neuen Problemen arbeiten kann. Ebenso gilt mein Dank Heribert Vollmer der mich immer auch in die Arbeit Hannoveraner Gruppe mit eingebunden hat, wodurch mir eine größere Breite in diesem (hoffentlich noch nicht letzten) Lernabschnitt meines Lebens gegeben wurde. Insofern hatte ich das Glück eigentlich von zwei Doktorvätern betreut worden zu sein.

Mein Dank gilt weiterhin Nadia Creignou mit der ich zusammenarbeiten durfte und die mich mehrfach in Marseille aufgenommen hat. Auch danke ich Edith Hemaspaandra die so freundlich war, mich an schönen Arbeiten zu beteiligen.

Besonders wichtig war mir immer die Diskussion mit meinen Kollegen Christian Glaßer, Daniel Meister, Steffen Reith und Stephen Travers. Besonderer Dank im Zusammenhang mit dieser Arbeit gebührt Steffen, der sie teilweise, und Daniel, der sie ernsthafterweise ganz Korrektur gelesen hat!

Schließlich möchte ich mich bei all jenen bedanken, die mir mein Studium und meine Promotion möglich gemacht haben. Insbesondere meine ich hier meine Tante Helga Wolf, die mich finanziell unterstützt hat; ohne sie wäre mein Studium nicht möglich gewesen.

It would not have been possible to finish this thesis without the help of many friendly people. I have to thank my supervisor Klaus Wagner especially, since he knows how to create a research-friendly environment and because his fruitful ideas lead the department to ever new and interesting problems. Also, I thank Heribert Vollmer who took care that I was integrated in the work of the group in Hannover, too, thus giving breadth to this learning epsisode of my live (which is, hopefully, not the last one). In this respect, I was lucky to have actually two supervisors for my doctorate.

Furthermore, I want to thank Nadia Creignou, who I was allowed to work with and who hosted me several times in Marseille. I thank Edith Hemaspaandra because she let me work with her on beautiful papers.

The discussion with my collegues Christian Glaßer, Daniel Meister, Steffen Reith and Stephen Travers was always of special importance to me. In connection with this thesis, I want to thank Steffen and Daniel particularly, since the former proofread half of the script and the latter seriously proofread the whole thing!

Finally, I want to thank all those who enabled me to study and to graduate. I am especially grateful to my aunt Helga Wolf, who supported me financially; I would not have been able to study without her.

Table of Contents

1. Introduction

There are two sorts of sciences: Deductive ones and inductive ones. The former, like biology and chemistry, study nature and try to derive rules that describe it as good as possible. The latter start from just a few basic laws, or objects and combine them in order to gain deeper cognitions and more elaborated tools.

Computer science is an inductive discipline. Although modern computer systems, networks, and software can be very complicated, all of them are built using a few basic objects which are then put together following some simple rules or operations.

Take for example a functional programming language: In the basic package of such a language there are just a few important functions, and the only way one is allowed to combine them is basically the substitution of an argument of one function with another function. Nevertheless, most functional programming languages are provably as strong as the other popular computational models.

As another example, take a computer-chip like a central processing unit (CPU). They have to be able to do numerous computations with large arguments and they control and synchronize most of the other components of a computer. These are large and difficult tasks, and to meet the demands, the CPU's have to be very complicated. How can you build such a complicated device? Of course, the answer is: You do not start with the whole device but with only a very small part of it. You could say, that the first step is to build a few gates (which can be seen as very simple processors themselves). Then you wire the gates together thus building a simple circuit which is still transparent. Take several such circuits to build an even larger one, and so on, until you finally arrive at the finished CPU.

Naturally, this method works the other way round, too: We are not only able to build very complicated structures, we are also able to better control them. Take the chip example: If one looks at the design plans of a modern processor, one sees an ocean of gates that are all somehow linked together. But the designer of the processor will be able to point out small subsections, with only a few hundred gates, and he will be able to describe the function of that subsection to us. He can then go further and brake the subsection down to a sub-sub-section with only a handful of gates the function of which is comparatively easy to understand. By this divide and conquer method, we will be able to understand the whole processor in time.

In algebra, *closures* are deeply related with the inductive principle. A closure is a set that is *closed* under a number of operations. We can visualize a set as a bag of objects and the operations as a toolkit containing tools to change or combine the objects in the bag. A

set S is closed under the operations in Ω if we cannot build new objects from the objects in S by application of the tools of Ω. With other words, if we apply a tool of Ω on, say, two objects of S the result is an object which is equal to one that is already in the bag. So, if S were not closed, you could build a closed set from S by applying the operations from Ω on S thus building a larger set, on which you apply the operations again, and so on, until you finally arrive at a set where new objects cannot be built any more, since all possible objects are there already (this could take infinitely long, though).

In this thesis, the objects in our bags are functions and we are interested in building new functions from basic ones. The most straightforward way to do this, is to write down formulas with symbols that stand for the known functions. Take for example two functions $f(x, y)$ and $g(z)$: Then, if we substitute the argument x of f with $g(z)$, we get the formula $f(g(z), y)$ which may describe a new function that is neither equal to f nor to g. The operations, or the toolkit, we use to assemble new functions from old ones is called *superposition*. Sets of functions that are closed under superposition are called *clones*. So, a function h is in the superposition-closure of $\{f, g\}$ if and only if there is a formula, for example $f(f(g(x_1), x_2), g(f(x_3, x_4)))$, that describes h. This thesis deals with questions that arise in connection with clones and computational complexity.

In computational complexity, you ask how difficult certain questions are to answer and how many resources a computer needs to answer them. Take, for example, a map with a number of cities on it which may or may not be connected with roads. One question is, given two cities A and B on the map, can you reach A from B? This is easy if there are just a few cities and roads but it becomes more and more difficult the more cities and roads are involved. This means that a computer needs the more resources, e.g. time, the larger the map is. Therefore the complexity of a question is always dependent on the question's size.

The above reachability-question is a rather easy problem; a human being will be able to efficiently solve it quickly, granted that the map is presented in reasonable form, and there are very efficient algorithms solving this problem. However, there are much more complex problems like the so called *Traveling Salesman Problem (TSP)*: Here, we are also confronted with a map with a number of cities that are conneted by roads. Each road between two cities has a certain length. A salesman has to visit all the, say n, cities. In order to be efficient, he should not travel more than, say k, kilometers. The question is, whether there is a route through the cities that makes this possible. There is no known algorithm that solves the TSP in polynomial time. This means that every known algorithm needs more than n^c steps to solve the problem for every natural number c.

In order to understand what it means that a problem cannot be solved in polynomial time, we have a look at a chart where the running times of different algorithms are compared. We assume in this chart that a computing-step can be carried out in 10^{-10} seconds which corresponds roughly to a 10 GHz computer. From left to right, we note the running times, from top to bottom, the size of the input (the number of cities in the example of TSP).

n (# cities)	$n \log n$	n^2	n^{20}	$2^{(\log n)^2}$	2^n
5	$< 10^{-9}$s	$2.5 \cdot 10^{-9}$s	~ 2.7 hours	$4 \cdot 10^{-9}$s	$3.2 \cdot 10^{-9}$s
10	$1.7 \cdot 10^{-9}$s	10^{-8}s	~ 317 years	$2 \cdot 10^{-7}$s	10^{-7}s
50	$2.8 \cdot 10^{-8}$s	$2.5 \cdot 10^{-7}$s	$3 \cdot 10^{16}$ years	0.3s	31.3 hours
100	$6.6 \cdot 10^{-8}$s	10^{-6}s	$3.2 \cdot 10^{22}$ years	21.6min	$9.6 \cdot 10^{13}$ years
1000	$9.9 \cdot 10^{-7}$s	10^{-4}s	$7.6 \cdot 10^{43}$ years	10^{12} years	$3.4 \cdot 10^{283}$ years

Note, that a running time of $n \log n$ is acceptable, even for large inputs, as is a running time for polynomials n^k where k is small. For large k, the running time grows very fast, but in practice such running times seldomly occur; in most cases, if an algorithm was found with a polynomial time bound shortly thereafter another one was found which was bounded by a polynomial of low degree. On the other hand, algorithms with super-polynomial running time are nearly always unacceptable. In the case of TSP this means that it cannot be exactly solved (although in special cases approximated), neither by human beings nor by computers, as far as we know.

The goal of complexity theory is to determine the complexity, or difficulty, of such problems as the TSP. That means that on the one hand, you want to find a fast algorithm for a problem. Such a fast algorithm establishes an upper complexity bound. On the other hand, you want to find a lower bound for the problem, i.e., you want to prove that you really need, say polynomial time, and no less, to solve the problem. We can group different problems that share the same upper (and sometimes lower) bounds together in *complexity classes*.

This makes the rating of the difficulty of problems easier. So, the TSP is in the complexity class NP. Furthermore, we can show that it is at least as difficult as every other problem in NP. This has two important consequences. Firstly, this means that the TSP is really very difficult; problems like it are known for roughly 35 years but no efficient, i.e., polynomial-time, algorithms have been found for solving one of them, so far. The question of whether such an algorithm exists consitutes the famous P-NP problem. Secondly, if we find an efficient algorithm to solve the TSP we can use it to efficiently solve every problem in NP. This means that we can use single problems as representatives for a whole class of problems.

This thesis shows that algebraic closures are a useful tool in complexity theory. We can use an existing classification of Boolean clones in order to classify different problems in connection with Boolean functions and Boolean relations according to their complexity. We show that a lot of complexity classes like FP are closed under superposition, and study some properties of the clone FP. We examine variations of the generation-problem, which is a problem defined with algebraic closures. The complexity of this problem catches exactly that of several important complexity classes, like P, NP, and PSPACE.

1.1 Overview

The second chapter is a preliminary one where we introduce the basics of universal algebra and complexity theory that are needed in the following chapters. We introduce our basic computation model, the Turing machine, in several variations. This we use to define the complexity classes we need.

In the third chapter we introduce clones and co-clones. We observe that the clones of functions over a domain form a lattice. Co-clones are closed sets of relations over a domain. We show that the worlds of clones and co-clones have a nice Galois connection. We then concentrate on Boolean clones and co-clones. We give a finite base for every Boolean clone and depict their inclusion structure, that was discovered by E. Post in the first half of the last century [Pos41]. The inclusion structure for the Boolean co-clones follows by said Galois connection. We also give "minimal" bases for co-clones and justify why these bases are indeed minimal.

We then, in the fourth chapter, have a closer look at the complexity of various problems in connection with Boolean circuits. The circuit satisfiability problem for B-circuits with selected exceptions is the question of whether or not a B-circuit is satisfiable with inputs other then a finite number of specified ones. Here, a B-circuit is a circuit where each gate is labeled with a function from B. A frozen variable in a B-circuit is a variable that takes the same value in all satisfying assignments for the circuit. We examine the complexity of finding out whether or not a circuit has one or more frozen variables and variations of this question. We classify the complexity of these problems according to the clone, B falls in optimally.

In the fifth chapter, we examine the complexity-class FP of functions computable in polynomial time. We observe that FP is a clone with a finite base. This implies that there is a lattice of clones below FP. There are several important complexity-classes which are clones in this lattice. Since the lattice is dual-atomic, these classes are either equal to FP or they are dual-atoms of FP or they are true sub-clones of such a dualatom. Therefore, the dualatoms are of special interest to us. We show that there are uncountably many dual-atoms in the lattice below FP. All functions in FP can be described by families of B-circuits of polynomial size in the number of input gates if the closure of B is the set of all Boolean functions. For all sets of Boolean functions A, we determine those for which families of A-circuits of polynomial size describe dualatoms in the lattice below FP. Finally, we show that there is no time complexity class that is a dual-atom in this lattice.

In the final chapter, we examine the generation problem GEN which is defined as follows. For a given binary function f on $\mathbb{N}$, we ask whether a function g is contained in the clone generated by f where we define g via a finite number of inputs and outputs. We get a more special version $\mathrm{GEN_s}$ of this problem if we give only one input $(a_1, \ldots, a_n)$ and one output d of g. This is equivalent of asking whether d can be generated by repeated application of f on $\{a_1, \ldots, a_n\}$. We first examine GEN and $\mathrm{GEN_s}$ for polynomial-time computable functions f and show that both are recursive enumerable if f is arbitrary

in FP, in EXPTIME if it is length-monotonic, in PSPACE if it is length-monotonic and associative, and in NP if it is length-monotonic, associative, and commutative. For all cases we find an f such that the easier problem $\mathrm{GEN_s}$ is hard for the respective class. We then examine the problems for f such that f is a polynomial. We show that for most polynomials both problems are in NP. We then prove NP-lower bounds for $\mathrm{GEN_s}$ with some polynomials. As by-product of this proofs, we show that a generalized version of the sum-of-subset problem SOS is NP-complete.

1.2 Publications

Parts of this thesis are published in the following papers:

[BCRV03] E. Böhler, N. Creignou, S. Reith, and H. Vollmer, *Playing with Boolean blocks, Part I: Post's lattice with applications to complexity theory.* ACM-SIGACT Newsletter 34, 2003.

[BCRV04] E. Böhler, N. Creignou, S. Reith, and H. Vollmer, *Playing with Boolean blocks, Part II: Constraint satisfaction problems.* ACM-SIGACT Newsletter 35, 2004.

[BGSW] E. Böhler, C. Glaßer, B. Schwarz, and K. W. Wagner, *Generation problems.* to appear in Theoretical Computer Science.

[BSRV05] E. Böhler, H. Schnoor, S. Reith, and H. Vollmer *Bases for Boolean Co-Clones.* To appear in Information Processing Letters 96, October 2005.

Moreover, the technical reports [BBC+05] and [Böh05] contain parts of this thesis, and Sections 4.1 and 4.2 are based on unpublished joint work with N. Creignou, M. Galota, and S. Reith.

2. Basics and Notations

2.1 Sets, Relations, and Functions

For a set A, let $2^A \stackrel{\text{def}}{=} \{B : B \subseteq A\}$ be the *power set* of A. Let $|A| = \#A$ be the number of elements in A. Let $\mathbb{N} = \{0, 1, 2, \dots\}$ be the set of natural numbers and let $\mathbb{Z} = \{0, 1, -1, 2, -2, \dots\}$ be the set of integers. For sets A and B, let $A \times B \stackrel{\text{def}}{=} \{(a, b) : a \in A \text{ and } b \in B\}$ be the *cross product* of A and B. Let $A^0 \stackrel{\text{def}}{=} \{\varepsilon\}$, $A^1 \stackrel{\text{def}}{=} A$ and for $n > 1$, let $A^n \stackrel{\text{def}}{=} A \times A^{n-1}$. We write $(a_1, a_2, \dots, a_{n-1}, a_n)$ for $(a_1, (a_2, \dots (a_{n-1}, a_n) \dots)) \in A^n$. Also, we sometimes regard the elements of A^n as words of length n with letters from A. In this case we write $a_1 \cdots a_n$ instead of $(a_1, \dots, a_n)$ and call A an *alphabet*. Mostly, we denote a finite but non-empty alphabet with Σ. Since there are easily computable bijections between Σ^* and $\mathbb{N}$, all our definitions and claims on the one set also hold for the other one and we will use both notations interchangeably. For a word $w \in A^n$, let $|w| \stackrel{\text{def}}{=} n$. Let $A^* \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}} A^n$ and $A^+ \stackrel{\text{def}}{=} A^* - \{\varepsilon\}$. If $w = w_1 w_2 \cdots w_n \in A^n$ is a word, let $w[i] \stackrel{\text{def}}{=} w_i$ be the i-th letter of w and, similarly, for tuples $\alpha = (a_1, \dots, a_n) \in A^n$, let $\alpha[i] \stackrel{\text{def}}{=} a_i$ $(1 \leq i \leq n)$. If $B \subseteq A^*$, let $\overline{B} \stackrel{\text{def}}{=} \{x : x \in A^* - B\}$; hence the semantic of the notation $\overline{B}$ is always relative to a basic universum A, which will always be clear from the context. For example, for subsets B of $\mathbb{N}$, let $\overline{B} \stackrel{\text{def}}{=} \mathbb{N} - B$.

We say B *is a relation over* A, if there is an $n \geq 1$ such that $B \subseteq A^n$. Then n is the *arity* of B, and B is n-ary. Let $\text{eq}_A \stackrel{\text{def}}{=} \{(a, a) : a \in A\}$ be the *equality relation*. We write just eq, if A is clear from the context. Let $\mathcal{R}^n(A)$ be the set of n-ary relations over A, and let $\mathcal{R}(A)$ be the set of all relations over A with finite arity.

An $n+1$-ary relation f over a set A is called *function*, if for all tuples $(a_1, \dots, a_n, a_{n+1})$, $(b_1, \dots, b_n, b_{n+1}) \in f$ holds that $(a_1, \dots, a_n) = (b_1, \dots, b_n)$ implies $a_{n+1} = b_{n+1}$. We say f is a *function from* A^n *to* A, and write $f : A^n \to A$. If $(a_1, \dots, a_n, a_{n+1}) \in f$, we write $f(a_1, \dots, a_n)$ for a_{n+1}. We let $f^{-1}(a) \stackrel{\text{def}}{=} \{\alpha \in A^n : f(\alpha) = a\}$. If $|f^{-1}(a)| = 1$ for all a, we often treat the set $f^{-1}(a)$ as if it were its single element. A function f is a *function on* A if there is an $n \in \mathbb{N}$ such that f is a function from A^n to A. We say that n is the *arity* of f or f *is* n*-ary*. A function on A with arity 0 can be identified with an element from A and is called a *constant* from A. A function of arity 1 is called *unary*, a function of arity 2 is called *binary*. Let $\mathcal{F}^n(A)$ be the set of all n-ary functions on A and let $\mathcal{F}(A)$ be the set of all functions on A. Let f be an n-ary function on A. In order to avoid cumbersome notation, let $f((a_1, \dots, a_n)) \stackrel{\text{def}}{=} f(a_1 \cdots a_n) \stackrel{\text{def}}{=} f(a_1, \dots, a_n)$ for $a_1, \dots, a_n \in A$. For example, if

$A = \{0,1\}$, and $n = 3$, then $f((1,1,0)) = f(110) = f(1,1,0)$. f is called *essentially unary*, if there is a unary function $g : A \to A$ and an $i \in \{1,\dots,n\}$ such that $f(a_1,\dots,a_n) = g(a_i)$ for all $a_1,\dots,a_n \in A$. An n-ary function f on A is called *total* if it is defined on every possible input from A^n, i.e., if for every $(a_1,\dots,a_n) \in A^n$ there is a $a_{n+1} \in A$ such that $(a_1,\dots,a_n,a_{n+1}) \in f$. For a function φ, let $D_\varphi \stackrel{\text{def}}{=} \{x : \varphi(x) \text{ is defined}\}$ be the *domain* of φ. If f is a unary function and $x \in A$, let $f^0(x) \stackrel{\text{def}}{=} x$ and $f^k(x) \stackrel{\text{def}}{=} f(f^{k-1}(x))$ for $k \geq 1$.

For every set A, all $n \geq 1$, and all $i \in \{1,\dots,n\}$ we define n-ary functions $\mathrm{I}_i^{A,n}$ such that for all $a_1,\dots,a_n \in A$ holds $\mathrm{I}_i^{A,n}(a_1,\dots,a_n) = a_i$. If A is clear from the context, we write I_i^n for $\mathrm{I}_i^{A,n}$. We call $\mathrm{I}_i^{A,n}$ an *identity function*, *identity*, or *projection*. For the unary identity I_1^1, we also write id. For all $k \in \mathbb{N}$, let $c_k \in \mathcal{F}^1(\mathbb{N})$ be the unary function with $c_k(x) = k$. Let $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$ be the successor function with $\mathrm{succ}(x) \stackrel{\text{def}}{=} x+1$ for all $x \in \mathbb{N}$. For all natural numbers $k \geq 2, n \geq 1$ we define a bijection $\mathrm{enc}_{n,k}$ between $\{0,\dots,k-1\}^n$ and $\{0,\dots,k^n-1\}$ with

$$\mathrm{enc}_{n,k}(a_{n-1},\dots,a_0) \stackrel{\text{def}}{=} \sum_{i=0}^{n-1} a_i k^i$$

For the following definition, let $m_x \stackrel{\text{def}}{=} \lfloor \log x + 1 \rfloor$ if $x > 0$ and let $m_x \stackrel{\text{def}}{=} 1$ if $x = 0$. Define for $n \geq m_x$ $\mathrm{bin}_n(x) \stackrel{\text{def}}{=} w_{n-1}\dots w_0$ such that $\mathrm{enc}_{n,2}(w_{k-1},\dots,w_0) = x$ and $\mathrm{bin}(x) \stackrel{\text{def}}{=} \mathrm{bin}_{m_x}(x)$. If $x \in \mathbb{N}$, let $|x| \stackrel{\text{def}}{=} |\mathrm{bin}(x)|$. With log we denote the logarithm to the base of 2.

Let $f, g \in \mathcal{F}^1(\mathbb{N})$ be total. We say f *is less or equal than* g ($f \leq g$) if and only if $f(x) \leq g(x)$ for all $x \in \mathbb{N}$. We say that f is less than or equal to g *almost everywhere*, if $f(x) \leq g(x)$ for all, but finite $x \in \mathbb{N}$. In this case, we write $g \leq_{\text{ae}} f$. We say, $f = O(g)$, if there is a $k \in \mathbb{N}$ such that $f \leq k + k \cdot g$. Also, let $O(g) \stackrel{\text{def}}{=} \{g' : g' \leq k + k \cdot g \text{ for a } k \in \mathbb{N}\}$ and for a class of functions $\mathcal{K}$, let $O(\mathcal{K}) \stackrel{\text{def}}{=} \{g' : \text{there is a } g \in \mathcal{K} \text{ and a } k \in \mathbb{N} \text{ such that } g' \leq k + k \cdot g\}$.

2.2 Algebraic Structures

2.2.1 Closures

Let A be a set, and let f be an n-ary function on A. For every $m \geq 1$ we define a function f^{m*} on A^m as follows: For $\alpha_1,\dots,\alpha_n \in A^m$, where $\alpha_i = (a_{i1},\dots,a_{im})$, let

$$f^{m*}(\alpha_1,\dots,\alpha_n) \stackrel{\text{def}}{=} (f(a_{11},\dots,a_{n1}),\dots,f(a_{1m},\dots,a_{nm})) \stackrel{\text{def}}{=} (b_1,\dots,b_n)$$

be the coordinate-wise application of f; Figure 2.1 illustrates the concept.

We write f instead of f^{m*} in most cases, since it will be clear from the context which function is meant. Let $R \subseteq A^m$ be an m-ary relation over the set A, and let f be an n-ary function on A. We say R *is closed under* f or f *preserves* R or f *is a polymorphism of* R

$$\begin{array}{ccccccc}
 & & \overbrace{\phantom{a_{11}}}^{f} & \overbrace{\phantom{a_{12}}}^{f} & \overbrace{\phantom{a_{13}}}^{f} & & \overbrace{\phantom{a_{1m}}}^{f} \\
\alpha_1 & = & a_{11} & a_{12} & a_{13} & \cdots & a_{1m} \\
\alpha_2 & = & a_{21} & a_{22} & a_{23} & \cdots & a_{2m} \\
\vdots & & \vdots & \vdots & \vdots & & \vdots \\
\alpha_n & = & \underbrace{a_{n1}} & \underbrace{a_{n2}} & \underbrace{a_{n3}} & \cdots & \underbrace{a_{nm}} \\
 & & \| & \| & \| & & \| \\
\beta & = & b_1 & b_2 & b_3 & \cdots & b_m
\end{array}$$

Fig. 2.1: $f^{m*}(\alpha_1, \ldots, \alpha_n) \stackrel{\text{def}}{=} \beta$.

if and only if for all $\alpha_1, \ldots, \alpha_n \in R$ we have $f(\alpha_1, \ldots, \alpha_n) \in R$. If f preserves R, we write $f \sim R$ and say R is *invariant* with respect to f.

Let Ω be a set of (not necessarily total) functions on A, and let $B \subseteq A$. Then $[B]_\Omega$ is the smallest set that contains B and that is closed under every function in Ω. That means that for every, say, n-ary function $f \in \Omega$ and all $a_1, \ldots, a_n \in [B]_\Omega$ holds $f(a_1, \ldots, a_n) \in [B]_\Omega$. If $A = [A]_\Omega$ then A *is closed under* Ω. We call $[A]_\Omega$ the Ω-*closure of* A. If $\Omega = \{f\}$ just contains a single function, we also write $[A]_f$ for $[A]_\Omega$. Furthermore, we also write $[a_1, \ldots, a_n]_\Omega$ for $[\{a_1, \ldots, a_n\}]_\Omega$. If for a set C holds $[C]_\Omega = A$, then C is a Ω-*base* of A. If Ω is clear from the context, we also write just *base* for Ω-base. Also, if B is a Ω-base for A, we call B Ω-*complete* (or just *complete*) for A. If there is a finite C such that $[C]_\Omega = A$ then A is *finitely* Ω-*generated*.

2.2.2 Algebras, Sub-algebras, and Lattices

An Ω-*algebra* $\mathcal{A}$ is a tuple (A, Ω) where A is a set of objects, the *universe* of $\mathcal{A}$, and Ω is a set of functions on A. We call these functions *operators*. If Ω is clear from the context, we will often identify $\mathcal{A}$ with its universe and simply write *algebra* instead of Ω-algebra. We call $\mathcal{A}$ *finitely generated*, if there is a finite $B \subseteq A$ such that $A = [B]_\Omega$. If $B \subseteq A$ is closed under Ω, then (B, Ω) is a *sub-algebra* of (A, Ω). A binary relation $\leq$ over a set X is called *partial order*, if for all $x, y, z \in X$ holds

1. $x \leq x$,
2. $x \leq y$ and $y \leq x$ implies $x = y$, and
3. $x \leq y$ and $y \leq z$ implies $x \leq z$.

With other words, $\leq$ is reflexive, antisymmetrical, and transitive. X together with such a relation is called a *partially ordered set* or *poset*. An element $a \in Y \subseteq X$ is the *maximum* of Y with respect to $\leq$, if for all $y \in Y$ with $a \leq y$ follows $y = a$. An element $a \in X$ is an *upper bound* of $Y \subseteq X$ if for all $y \in Y$ holds $y \leq a$. We define the *minimum* and the *lower bound* of Y analogously but with reversed order. For $Y \subseteq X$, we define $\bigsqcup Y$ to be the minimum of the set of upper bounds of Y and $\bigsqcap Y$ to be the maximum of the set of lower bounds of Y. If $Y = \{a, b\}$, we also write $a \sqcup b$ and $a \sqcap b$ for $\bigsqcup\{a, b\}$ and $\bigsqcap\{a, b\}$, respectively. We call $a \sqcup b$ the *meet* of a and b and $a \sqcap b$ the *join* of a and b. If for all

$a, b \in A$ both, meet and join, exist in A, we call the algebra $(A, \{\sqcup, \sqcap\})$ a *lattice*. A lattice is *complete* if for all $B \subseteq A$ the elements $\bigsqcap B$ and $\bigsqcup B$ exist in A. Therefore a complete lattice always has a minimal element $\bigsqcap A$ and a maximal element $\bigsqcup A$. For example, the natural numbers form a lattice with respect to $\leq_{\mathbb{N}}$, but the lattice is not complete, since $\bigsqcup \mathbb{N}$ does not exist. In a poset A with respect to $\leq$, for $a, b \in A$ we write $a < b$ if $a \leq b$ and $a \neq b$. We say *a is predecessor of b* or *b is successor of a* ($a \prec b$) if and only if $a < b$ and there is no $c \in A$ such that $a < c < b$. A lattice A is *atomar* if it has a minimal element $0 \in A$, and for every $a \in A - \{0\}$ there is a $b \succ 0$ such that $a \geq b$. For example, $\mathbb{N}$ is an atomar lattice with respect to $\leq_{\mathbb{Q}}$, the normal order relation on $\mathbb{Q}$, but $\mathbb{Q}^+$, the set of positive rational numbers together with 0, is not. Analogously, A is called *dual-atomic* if it has a maximum element $1 \in A$, and for every $a \in A - \{1\}$ there is a $b \prec 1$ such that $a \leq b$. In such a lattice, an element b with $b \prec 1$ is called *dual-atom* or *precomplete*. See [DP90] for a thorough introduction to lattices.
For every Ω-algebra $\mathcal{A} = (A, \Omega)$, the set of subalgebras

$$\mathcal{L}(\mathcal{A}) \stackrel{\text{def}}{=} \{(B, \Omega) : B = [B]_\Omega \subseteq A\}$$

forms a complete lattice [Coh65, Grä79]. Hence, the elements of the lattice are exactly the subsets of A that are closed under Ω. They are ordered via set inclusion. If B and C are elements in $\mathcal{L}((A, \Omega))$, the join of B and C is $[B \cup C]_\Omega$ and the meet is $B \cap C$. The minimal element of the lattice is always $[\emptyset]_\Omega$, and the maximal one is always A itself.

Proposition 2.1. *If the algebra $\mathcal{A} = (A, \Omega)$ is finitely generated then $\mathcal{L}(\mathcal{A})$ is dual-atomic.*

Proof. Suppose $\mathcal{L}(\mathcal{A})$ were not dual-atomic. Then there is an infinite chain $C_1 \subset C_2 \subset \cdots \subset A$ of sub-algebras directly below A, i.e., there is no sub-algebra $C \subset A$ such that $C_i \subset C$ for all $i \geq 1$. Let $D \stackrel{\text{def}}{=} \bigcup_{i \geq 1} C_i$. Then $C_i \subset D$ for all $i \geq 1$ and, since $\mathcal{L}(A)$ is not dual-atomic, $A = D$. Let B be a finite set with $[B]_\Omega = A = D$. Due to the nature of D, there has to be an $i \geq 1$ such that $B \subseteq C_i$ and therefore $[C_i]_\Omega = A$. ❑

We can use the structure of the lattice of sub-algebras to find bases for the elements of the lattice in the following way.

Proposition 2.2. *Let (A, Ω) be an algebra, $B \in \mathcal{L}((A, \Omega))$ and $C \subseteq A$. Then $[C]_\Omega = B$ if and only if $C \subseteq B$ and for all $D \in \mathcal{L}((A, \Omega))$ with $D \prec B$ holds $C \not\subseteq D$.*

Proof. If $[C]_\Omega = B$, then obviously $C \subseteq B$. Assume there were a $D \prec B$ such that $C \subseteq D$. Then $[C]_\Omega \subseteq D \neq B$.

Now, assume that right hand side of the claim holds. Since $C \subseteq B$, the closure $[C]_\Omega$ is a subalgebra of B or B itself. Since C is not contained in a predecessor of B, we have $[C]_\Omega = B$. ❑

2.3 Graphs, Trees, Boolean Functions, and Circuits

A (undirected) *graph* is a pair (V, E) such that $V \subseteq \mathbb{N}$ and $E \subseteq \{\{v_1, v_2\} : v_1, v_2 \in V \text{ and } v_1 \neq v_2\}$. We call $v \in V$ *nodes* of the graph and $e \in E$ its *edges*. In a *directed graph* (V, E), we have $E \in \mathcal{R}^2(V)$. Such a graph is called *cyclic* if there is a $n \in \mathbb{N}$ and $v_1, \ldots, v_n \in V$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i < n$ and $v_1 = v_n$. A graph that is not cyclic is called *acyclic*. The *size* of a (directed) graph is $|V|$. The *depth* of a directed graph (V, E) is defined as the largest n such that there are pairwise different $v_1, \ldots, v_n \in V$ and $(v_i, v_{i+1}) \in E$ for $1 \leq i < n$. The (directed) graph (V, E) is called *connected* if for all $v_1, v_2 \in V$ where $v_1 \neq v_2$ there is a $n \in \mathbb{N}$ and $w_1, \ldots, w_n \in V$ such that $v_1 = w_1$ and $v_2 = w_n$ and $\{w_i, w_{i+1}\} \in E$ (either $(w_i, w_{i+1}) \in E$ or $(w_{i+1}, w_i) \in E$) for $1 \leq i < n$. Let (V, E) be a directed graph. For a $v \in V$, let $\#\{w : (v, w) \in E\}$ be the *fan-out* of v and let $\#\{w : (w, v) \in E\}$ be the *fan-in* of v. If $(v, w) \in E$, we call v *parent* of w and we call w *child* or *successor* of v. A *tree* is a directed, connected, acyclic graph where every node has fan-in of at most 1. In a tree, a node with no successors is called *leaf*. A *binary tree* is a tree where every node has either two children or none. If $G = (V, E)$ is a tree, there is exactly one node in V with fan-in 0. We call this node the *root node* or simply *root* of G. In a binary tree (V, E), every node can be characterized by a *path*, i.e., a word $w \in \{l, r\}^*$, as follows: The path of the root node is ε. If $v \in V$ is characterized by the path w and has children v_1, v_2 where $v_1 < v_2$, then v_1 is characterized by the path wl and v_2 is characterized by the path wr. A node v_1 is called *ancestor* of another node v_2 if there are $w_1, w_2 \in \{l, r\}^*$ such that v_2 is characterized by $w_1 w_2$, and w_1 characterizes v_1. The *depth* of a node v is $|w|$ if w characterizes v.

A *Boolean function* f is a function from $\{0,1\}^n$ to $\{0,1\}$ where $n \in \mathbb{N}$. We use several ways to define Boolean functions. For one we have some special Boolean functions, we will often need: And, Or, Not, Imp, Eq, Xor, Nand, Nor, c_0, and c_1 are the Boolean and, or, not, implication, equivalence, exclusive or, nand, nor, and the 0-ary constants 0 and 1, respectively (see Figure 2.2). A way to define a Boolean function is to explicitly give its truth table, i.e., listing the input-output behavior of the function. The *characteristic string* of a Boolean function is a compressed version of such a truth-table. We say $w \in \{0,1\}^{2^n}$ is the characteristic string of the n-ary Boolean function f if and only if for all $i \in \{0, \ldots, 2^n - 1\}$ holds $w[i+1] = f(\mathrm{bin}_n(i))$. Although characteristic strings are a short description of the truth-table of a Boolean function, a much more efficient description of these functions exists in most cases. We mostly use propositional formulas (see below) to describe Boolean functions. The connectives we use in these formulas are subsumed in Figure 2.2.

Definition 2.3. *Let B be a set of Boolean functions. A tuple $C = (V, E, o, \phi, \psi)$ is called* B-circuit *if it satisfies the following conditions.*

- *$V \subseteq \mathbb{N}$ is finite and (V, E) is a tree with root node o. We call o the* output node *of C. Since $o \in V$, there is no B-circuit with $|V| = 0$. The elements of V are also called* gates.
- *$\phi : V \to B \cup \{x_1, x_2, \ldots\}$ is a function that assigns to every node from V either a variable or a function from B in the following way: If $v \in V$ has fan-in 0, then $\phi(v) = x_i$, for*

Name	arity	Description	Connective(s)	Characteristic String
AND	2	Boolean and	$x \wedge y$, $x \cdot y$, xy	0001
OR	2	Boolean or	$x \vee y$	0111
IMP	2	implication	$x \rightarrow y$	1101
EQ	2	equivalence	$x \leftrightarrow y$	1001
XOR	2	exclusive or	$x \oplus y$	0110
NAND	2	negation of and	$\text{ND}(x, y)$	1110
NOR	2	negation of or	$\text{NR}(x, y)$	1000
NOT	1	negation	$\overline{x}$, $\neg x$	10
id	1	identity	x	01
c_0	0	constant zero	0	0
c_1	0	constant one	1	1

Fig. 2.2: Description of Important Boolean Functions

some $i \geq 1$, or $\phi(v)$ is a 0-ary function from B. If v has fan-in $n \geq 1$, then $\phi(v) = f$ for some n-ary function from B.
- *$\psi : V \cup E \rightarrow \mathbb{N}$ is an injective function that assigns a natural number to every node from V and every edge from E.*

If $x_1, \ldots, x_n$ are the variables assigned to some gate of C then in every gate v of C, an n-ary Boolean function f_v is calculated as follows.

- *If $\phi(v) = x_i$ then $f_v \stackrel{\text{def}}{=} \text{I}_i^n$.*
- *If $\phi(v) = c_a$ for an $a \in \{0, 1\}$ then f_v is the n-ary constant a.*
- *If $\phi(v) = g$ for an m-ary $g \in B$ and $(v_1, v), \ldots, (v_m, v) \in E$ such that $\psi((v_i, v)) < \psi((v_j, v))$ if $i < j$ then $f_v(x_1, \ldots, x_n) \stackrel{\text{def}}{=} g(f_{v_1}(x_1, \ldots, x_n), \ldots, f_{v_m}(x_1, \ldots, x_n))$.*

Finally we say C describes *the Boolean function $f_C \stackrel{\text{def}}{=} f_o$, which is the Boolean function calculated in the output gate of C.*

Observe that the Boolean functions that can be described by B-circuits are exactly those in $[B]$. If we do not want to emphasise the base B, we say *Boolean circuit* for B-circuit. For a more detailed introduction to Boolean circuits, see [Vol99]. A *propositional formula* is a special Boolean circuit where the fan-out of every node is at most one. For $\alpha \in \{0, 1\}^n$, we often write $C(\alpha)$ instead of $f_C(\alpha)$, if C is an n-ary Boolean circuit that describes f. An n-tuple of Boolean values is often called *assignment* for a Boolean circuit of arity n. We say two Boolean circuits C_1, C_2 are *equivalent* ($C_1 \equiv C_2$) if and only if they describe the same Boolean function.

2.4 Complexity-Theoretical Basics

2.4.1 Turing Machines

We want to fix Turing machines with separate input-tape and output-tape and a constant number of working-tapes where each tape has infinite space on both sides as the standard

model of computation. Let us give a formal definition of this model. A *Turing machine* M is a tuple $(k, \Sigma, Z, s, h, \varphi)$, where

- $k \geq 1$ is a natural number, the number of working tapes,
- Σ is a finite alphabet with $\{0, 1, \square\} \subseteq \Sigma$, the *working alphabet*,
- Z is a finite set of *states*,
- $s \in Z$ is the *initial state*,
- $h \in Z$ is the *halting state*, and
- $\varphi : Z \times \Sigma \cup \{*\} \times \Sigma^k \to 2^{Z \times \Sigma^{k+1} \times \{L,R,O\}^{k+2}} - \{\emptyset\}$ where $* \notin \Sigma$, is a total function, the *transition function*

of M. A *configuration* of a Turing machine with k working tapes is a $(k+3)$-tuple from

$$Z \times ((\Sigma \cup \{*\})^+ \cdot p \cdot (\Sigma \cup \{*\})^+) \times (\Sigma^+ \cdot \{p\} \cdot \Sigma^+)^{k+1}$$

where $p \notin \Sigma \cup \{*\}$. Let $C \stackrel{\text{def}}{=} (z, v_0 p w_0, \ldots, v_k p w_k, v_{k+1} p w_{k+1})$ be a configuration of M where $z \in Z$ and $v_i, w_i \in \Sigma^+$ for $0 \leq i \leq k+1$. Let furthermore

$$\varphi(z, w_0[1], \ldots, w_k[1]) = \{(z'_j, a_{j,1}, \ldots, a_{j,k+1}, X_{j,0}, \ldots, X_{j,k+1}) : 1 \leq j \leq \ell\},$$

for some $\ell \geq 1$. Each configuration has successor configurations. For that we define the successor-function S_M with $S_M(C) = \{C\}$ if $z = h$ and $S_M(C) \stackrel{\text{def}}{=} \{(z'_j, u_{j,0}, \ldots, u_{j,k+1}) : 1 \leq j \leq \ell\}$, otherwise, where

$$u_{j,i} = \begin{cases} v_i p a_{j,i} w'_i & \text{, if } X_{j,i} = O \\ v_i a_{j,i} p w'_i & \text{, if } X_{j,i} = R \text{ and } |w_i| > 1 \\ v_i a_{j,i} p \square & \text{, if } X_{j,i} = R \text{ and } |w_i| = 1 \\ v_i[1] \cdots v_i[|v_i| - 1] p v_i[|v_i|] a_{j,i} w'_i & \text{, if } X_{j,i} = L \text{ and } |v_i| > 1 \\ \square p v_i a_{j,i} w'_i & \text{, if } X_{j,i} = L \text{ and } |v_i| = 1 \end{cases}$$

for $1 \leq j \leq \ell$, $1 \leq i \leq k+1$, and $w'_i \stackrel{\text{def}}{=} w_i[2] \cdots w_i[|w_i|]$ and

$$u_{j,0} = \begin{cases} v_0 p w_0 & \text{, if } X_{j,0} = O \\ v_0 w_0[1] p w_0[2] \cdots w_0[|w_0|] & \text{, if } X_{j,0} = R \text{ and } |w_0| > 1 \\ v_0 w_0 p \square & \text{, if } X_{j,0} = R \text{ and } |w_0| = 1 \\ v_0[1] \cdots v_0[|v_0| - 1] p v_0[|v_0|] w_0 & \text{, if } X_{j,0} = L \text{ and } |v_0| > 1 \\ \square p v_0 w_0 & \text{, if } X_{j,0} = L \text{ and } |v_0| = 1 \end{cases}$$

for $1 \leq j \leq \ell$.

For two configurations C_1, C_2 we say M *transforms* C_1 *into* C_2 *in* n *steps* ($C_1 \xrightarrow{M,n} C_2$) if and only if there are configurations $K_1, \ldots, K_{n+1}$ such that $K_1 = C_1$ and $K_{n+1} = C_2$ and $K_{i+1} \in S_M(K_i)$ for $1 \leq i \leq n$. Call all C' such that $C \xrightarrow{M,1} C'$ the M*-successors* of C or simply the *successors* of C. We say M *transforms* C_1 *into* C_2 ($C_1 \xrightarrow{M,*} C_2$) if there is an $n \in \mathbb{N}$ such that $C_1 \xrightarrow{M,n} C_2$.

Let ψ be an n-ary function on $\mathbb{N}$. We say the Turing machine M *computes* $\psi(a_1, \ldots, a_n)$ in $m \in \mathbb{N}$ steps if and only if

$$\begin{array}{l}(s, \Box p\mathrm{bin}(a_1) * \cdots * \mathrm{bin}(a_n), \Box p\Box, \ldots, \Box p\Box) \stackrel{M,m}{\longrightarrow} \\ (h, \Box p\mathrm{bin}(a_1) * \cdots * \mathrm{bin}(a_n)w_0, w_1 p r w_2, w_3 p w_4, \ldots, w_{2k+1} p w_{2k+2})\end{array} \tag{2.1}$$

where $w_i \in \{\Box\}^*$ for $0 \leq i \leq 2k+2$ and $r = \mathrm{bin}(\psi(a_1, \ldots, a_n))$. Let us call the configuration on the left hand side the *starting configuration of M for $a_1, \ldots, a_n$* and the configurations on the right hand side *halting configurations of M for $a_1, \ldots, a_n$*. Let us furthermore call r the *result* of the halting configuration. The machine M computes $\psi(a_1, \ldots, a_n)$ with *space* ℓ if and only if there is an m such that (2.1) holds and $|\mathrm{bin}(\psi(a_1, \ldots, a_n))| + \Sigma_{i=1}^{2k}|w_i| \leq \ell$. We say M computes ψ if and only if for all $(a_1, \ldots, a_n) \in D_\psi$ there is an $m \in \mathbb{N}$ such that (2.1) holds. Such a ψ is called *computable*; if ψ is total, it is called *recursive*.

A set $A \subseteq A$ is *recursively enumerable* if there is a recursive function $f : \mathbb{N} \to \mathbb{N}$ such that $f(\mathbb{N}) = A$. Turing machines can be encoded with a finite alphabet; there are many encoding schemes for Turing machines such that the set of all encodings of Turing machines can be enumerated recursively.

A Turing machine is called *deterministic*, if for all possible inputs α of its transition function φ, holds $|\varphi(\alpha)| = 1$. Otherwise, it is called *nondeterministic*.

An *alternating Turing machine* $M = (k, \Sigma, Z, s, h, \varphi, \pi)$ is a Turing machine together with a function $\pi : Z \to \{e, u\}$ that partitions the states of M in *existential* (if $\pi(z) = e$) and *universal* (if $\pi(z) = u$) states (for $z \in Z$). Let C be a configuration of an alternating Turing machine on input $(a_1, \ldots, a_n)$. If

$$C \quad = \quad (h, \Box p\mathrm{bin}(a_1) * \cdots * \mathrm{bin}(a_n)w_0, w_1 p 1 w_2, w_3 p w_4, \ldots, w_{2k-1} p w_{2k}) \tag{2.2}$$

where $w_0, \ldots, w_{2k} \in \{\Box\}^*$, then C is *accepting*. Such a configuration is called *accepting end-configuration for $a_1, \ldots, a_n$*. If $C = (z, v_0, \ldots, v_k)$, such that $z \in Z - \{h\}$ and $\{C_1, \ldots, C_\ell\}$ are all successors of C, then C is accepting if and only if one of the following holds:

1. $\pi(z) = e$ and there is an $i \in \{1, \ldots, \ell\}$ such that C_i is accepting.
2. $\pi(z) = u$ and C_i is accepting for all $i \in \{1, \ldots, \ell\}$.

All configurations that are not accepting are *rejecting*. An alternating Turing machine *accepts* an input $(a_1, \ldots, a_n)$, if the starting configuration for $a_1, \ldots, a_n$ is accepting.

2.4.2 Complexity Classes

For a function $f \in \mathcal{F}^1(\mathbb{N})$, we say the Turing machine M *computes ψ in time f* (M *computes ψ with space f*) if and only if ψ is total and M computes $\psi(a_1, \ldots, a_n)$ in $f(\Sigma_{i=1}^n|\mathrm{bin}(a_i)|)$ steps (with space $f(\Sigma_{i=1}^n|\mathrm{bin}(a_i)|)$) for all but finite $(a_1, \ldots, a_n) \in \mathbb{N}^n$.

Let $\mathcal{K} \subseteq \mathcal{F}^1(\mathbb{N})$ and let $\mathcal{P}$ be the set of unary polynomials. We define the following *function-complexity classes*:

$$
\begin{aligned}
\mathrm{FDTIME}(\mathcal{K}) &\stackrel{\text{def}}{=} \{\psi : \text{there is a deterministic Turing machine } M \text{ and an } f \in \mathcal{K} \\
&\qquad \text{such that } M \text{ computes } \psi \text{ in time } f\} \\
\mathrm{FDSPACE}(\mathcal{K}) &\stackrel{\text{def}}{=} \{\psi : \text{there is a deterministic Turing machine } M \text{ and an } f \in \mathcal{K} \\
&\qquad \text{such that } M \text{ computes } \psi \text{ with space } f\} \\
\mathrm{FPSPACE} &\stackrel{\text{def}}{=} \mathrm{FDSPACE}(\mathcal{P}) \cap \\
&\qquad \{f : \text{ there is a polynomial } p \text{ such that } |f(x)| \leq p(|x|) \text{ for all } x\} \\
\mathrm{FP} &\stackrel{\text{def}}{=} \mathrm{FDTIME}(\mathcal{P}) \\
\mathrm{FL} &\stackrel{\text{def}}{=} \mathrm{FDSPACE}(\log n)
\end{aligned}
$$

Note, that all functions in FPSPACE have an output-length of only polynomial size. Since it is possible to produce outputs of exponential length with functions computable in polynomial space, this definition would not suit our needs, since we want FDSPACE to be closed under substitution. The *characteristic function* of a set $A \subseteq \mathbb{N}$ is a total function $c_A : \mathbb{N} \to \{0, 1\}$ such that $c_A(x) = 1$ if and only if $x \in A$. We say c_A *decides* A and A is *decidable*, if c_A is recursive. Analogous to the function-complexity classes, we define the following *decision-complexity classes*:

$$
\begin{aligned}
\mathrm{DTIME}(\mathcal{K}) &\stackrel{\text{def}}{=} \{A : c_A \in \mathrm{FDTIME}(\mathcal{K})\} \\
\mathrm{DSPACE}(\mathcal{K}) &\stackrel{\text{def}}{=} \{A : c_A \in \mathrm{FDSPACE}(\mathcal{K})\} \\
\mathrm{EXPTIME} &\stackrel{\text{def}}{=} \mathrm{DTIME}(2^{\mathcal{P}}) \\
\mathrm{PSPACE} &\stackrel{\text{def}}{=} \mathrm{DSPACE}(\mathcal{P}) \\
\mathrm{P} &\stackrel{\text{def}}{=} \mathrm{DTIME}(\mathcal{P}) \\
\mathrm{L} &\stackrel{\text{def}}{=} \mathrm{DSPACE}(\log n)
\end{aligned}
$$

A nondeterministic Turing machine can transform the starting configuration into several halting configurations simultaneously. We say, M decides the set A, if for all α holds:

– If $\alpha \in A$ then there is a halting configuration of M on input α with result 1.
– If $\alpha \notin A$ then for all halting configurations of M on input α the result is 0.

For an $f \in \mathcal{F}^1(\mathbb{N})$, we say M *decides* A *in time* f (M *decides* A *with space* f), if M decides A and for all $\alpha \in A$ there is a halting configuration C as in Equation 2.2 of M on x such that M transforms the starting configuration of M on x into C in at most $f(|x|)$ steps (such that $\Sigma_{i=1}^{2k}|w_i| \leq f(|x|)$).

We define the following nondeterministic complexity classes:

$$\begin{aligned}
\mathrm{NTIME}(\mathcal{K}) &\stackrel{\text{def}}{=} \{A : \text{there is a nondeterministic Turing machine and an } f \in \mathcal{K} \\
&\qquad \text{such that } M \text{ decides } A \text{ in time } f\} \\
\mathrm{NSPACE}(\mathcal{K}) &\stackrel{\text{def}}{=} \{A : \text{there is a nondeterministic Turing machine and an } f \in \mathcal{K} \\
&\qquad \text{such that } M \text{ decides } A \text{ with space } f\} \\
\mathrm{NP} &\stackrel{\text{def}}{=} \mathrm{NTIME}(\mathcal{P}) \\
\mathrm{NL} &\stackrel{\text{def}}{=} \mathrm{NSPACE}(\log n)
\end{aligned}$$

Another way to define NP is with an existential quantifier as follows: A is in NP if and only if there is a set $B \in \mathrm{P}$ and a polynomial p such that for all x holds

$$x \in A \Leftrightarrow \exists y(|y| \leq p(|x|) \text{ and } (x, y) \in B)$$

For a complexity class $\mathcal{C}$, let $\mathrm{co}\mathcal{C} \stackrel{\text{def}}{=} \{\overline{A} : A \in \mathcal{C}\}$. Another way to define coNP is as in the above definition for NP, but with a $\forall$-quantifier. We can use this quantifier approach to define a hierarchy of complexity classes as follows. For $k \geq 0$, a set A is in Σ_k^{p} if and only if there is a $B \in \mathrm{P}$ and a polynomial p such that for all x holds:

$$x \in A \Leftrightarrow \exists y_1 \forall y_2 \ldots Q_k y_k(|y_i| \leq p(|x|) \text{ for all } 1 \leq i \leq k \text{ and } (x, y_1, \ldots, y_k) \in B),$$

where $Q_k = \exists$ if k is odd, and $Q_k = \forall$ otherwise. Let $\Pi_k^{\mathrm{p}} \stackrel{\text{def}}{=} \mathrm{co}\Sigma_k^{\mathrm{p}}$. These classes form the *polynomial-time hierarchy*. Let $\mathrm{PH} \stackrel{\text{def}}{=} \bigcup_{k \geq 0} \Sigma_k^{\mathrm{p}}$.

Another hierarchy is the *Boolean hierarchy*. For that, let $\mathcal{C}$ and $\mathcal{K}$ be complexity classes and define

$$\mathcal{C} \oplus \mathcal{K} \stackrel{\text{def}}{=} \{A \triangle B : A \in \mathcal{C} \text{ and } B \in \mathcal{K}\}.$$

Here, $A \triangle B \stackrel{\text{def}}{=} (A \cup B) - (A \cap B)$. For $k \geq 1$, we say $A \in \mathcal{K}(k)$ if and only if

$$A \in \underbrace{\mathcal{K} \oplus \mathcal{K} \oplus \cdots \oplus \mathcal{K}}_{k \text{ times}}.$$

We are especially interested in the Boolean hierarchy over NP. Let $\mathrm{DP} \stackrel{\text{def}}{=} \mathrm{NP}(2)$.

We want to fix a notion of time- and space-complexity for alternating Turing machines M. For this, we define functions t_M^α and s_M^α for $\alpha = (a_1, \ldots, a_n)$ that map accepting configurations of M on α to $\mathbb{N}$ as follows: If C is an accepting end-configuration for $a_1, \ldots, a_n$ as in Equation 2.2, then $t_M^\alpha(C) \stackrel{\text{def}}{=} 1$ and $s_M^\alpha(C) \stackrel{\text{def}}{=} \Sigma_{i=1}^{2k} |w_i|$. If $C = (z, w_0, \ldots, w_k)$ is another accepting configuration which has exactly the successors $C_1, \ldots, C_\ell$, then

$$t_M^\alpha(C) \stackrel{\text{def}}{=} \begin{cases} 1 + \min\{t_M^\alpha(C_i) : C_i \text{ is accepting } (1 \leq i \leq \ell)\} & \text{, if } \pi(z) = e \\ 1 + \max\{t_M^\alpha(C_i) : 1 \leq i \leq \ell\} & \text{otherwise} \end{cases}$$

and

$$s_M^\alpha(C) \stackrel{\text{def}}{=} \begin{cases} \min\{s_M^\alpha(C_i) : C_i \text{ is accepting } (1 \leq i \leq \ell)\} & \text{, if } \pi(z) = e \\ \max\{s_M^\alpha(C_i) : 1 \leq i \leq \ell\} & \text{otherwise.} \end{cases}$$

We say M decides the set $A \subseteq \mathbb{N}^n$ in time (with space) $f \in \mathcal{F}^1(\mathbb{N})$ if for all $\alpha \in \mathbb{N}$ the following holds: If $\alpha \in A$ then the starting configuration C of M on α is accepting and $t_M^\alpha(C) \leq f(|\alpha|)$ ($s_M^\alpha(C) \leq f(|\alpha|)$). If $\alpha \notin A$ then C is rejecting. We define the following alternating complexity classes, where $\mathcal{K} \subseteq \mathcal{F}^1(\mathbb{N})$:

$$\begin{aligned} \mathrm{ATIME}(\mathcal{K}) &\stackrel{\mathrm{def}}{=} \{A : \text{there is an alternating Turing machine } M \text{ and an } f \in \mathcal{K} \\ &\qquad \text{such that } M \text{ decides } A \text{ in time } f\} \\ \mathrm{ASPACE}(\mathcal{K}) &\stackrel{\mathrm{def}}{=} \{A : \text{there is an alternating Turing machine } M \text{ and an } f \in \mathcal{K} \\ &\qquad \text{such that } M \text{ decides } A \text{ with space } f\} \end{aligned}$$

The definition of Turing machines and standard notions of complexity theory can be found in a number of textbooks [BDG95, BDG90, Pap94, WW86].

2.4.3 Families of Boolean Circuits and Uniformity

We have already given a definition of Boolean circuits in Definition 2.3. However, we want to be able to somehow compute functions on $\mathbb{N}$ as well. For that, let a *B-circuit with multiple outputs* be a tuple (V, E, O, ϕ, ψ) where V, E, ϕ, ψ are defined as in Definition 2.3 but $O \subseteq V$. We say B-circuit as a shortcut for B-circuit with multiple outputs when it is clear from the context that we mean a circuit with multiple outputs. Let $C = (V, E, O, \phi, \psi)$ be a circuit with multiple outputs and n input-gates. Let $O = \{o_1, \ldots, o_m\}$ where $\psi(o_i) < \psi(o_j)$ implies $i < j$. Then C computes the function that maps $a_1 \cdots a_n \in \{0,1\}^n$ to $f_{o_1}(a_1, \ldots, a_n) \cdots f_{o_m}(a_1, \ldots, a_n)$. Remember that $f_v(a_1, \ldots, a_n)$ denotes the Boolean function that is computed in node v on input $a_1, \ldots, a_n$. Let $\mathrm{size}(C)$ denote the number of gates in the circuit C and let $\mathrm{depth}(C)$ denote the length of a longest path from an input gate to an output gate, formally,

$$\begin{aligned} \mathrm{depth}(C) \stackrel{\mathrm{def}}{=} \max\{n : &\text{there are gates } c_1, \ldots, c_{n+1} \in V \text{ such that } c_1 \text{ is an input-gate} \\ &\text{and } c_{n+1} \text{ is an output-gate and } (c_i, c_{i+1}) \in E \text{ for all } 1 \leq i \leq n\}. \end{aligned}$$

Since every single Boolean circuit has a fixed number of input gates, for such a circuit it is possible to compute a function only for inputs of a fixed length. Therefore, we need sequences or *families* $\mathcal{C} = (C^n)_{n \in \mathbb{N}}$ of Boolean circuits to compute a function on $\mathbb{N}$. Here C^n denotes a circuit with n input-gates. A family of Boolean circuits is an infinite object and therefore different from the other models of computation we have considered so far, which are always finite. Therefore, it is not surprising that there are undecidable sets that can be decided by families of Boolean circuits:

Example: Let $A \stackrel{\mathrm{def}}{=} \{1^n : M_n \text{ accepts } n\}$ be the tally version of the special halting problem, which is undecidable (here M_n means the n-th Turing machine in some enumeration of all Turing machines). It is obvious, that for every $n \in \mathbb{N}$ there is a very small circuit C^n that outputs 1 if and only if $1^n \in A$ [Vol99]. □

So, we need a finite way to encode families of Boolean circuits. If a family of circuits can be encoded by a finite object, it is called a *uniform* family of circuits. There are several different notions of uniformity but in this thesis it suffices to define *log-space uniformity*.

Definition 2.4. *Let $\mathcal{C} = (C^n)_{n\in\mathbb{N}}$ be a family of circuits. Then $\mathcal{C}$ is called* log-space uniform *or* $\mathrm{U_L}$*-uniform if there is a function $f \in \mathrm{FL}$ such that $f(1^n) = C^n$ for all $n \in \mathbb{N}$.*

A function that can be calculated by a log-space uniform family of Boolean circuits is obviously computable, since we can compute $f(1^{|x|}) = C^{|x|}$ and evaluate $C^{|x|}$ on input x. The choice of log-space uniformity is based on the following reasons. First of all, the uniformity should not be too "weak" for the circuit classes in consideration. Too weak means that the power of the function $f : \{1\}^* \to (C^n)_{n\in\mathbb{N}}$ should not be greater than the power of the circuits themselves. In the above example, the circuits C^n could be "generated" by a non-recursive function f; in that sense, the family of circuits $\mathcal{C}$ that decides A is (non-recursively-)uniform. But this uniformity is clearly too weak if you are interested in the question whether A is decidable or not. In our case, log-space uniformity is strong enough, since we are mostly interested in circuits that have a polynomial size and a depth that is at most slightly less than polynomial. A log-space computation can always be computed by such circuits, since all functions from FL can be computed by circuits of polynomial size and a depth of $(\log n)^2$. On the other hand, log-space uniformity is weak enough to handle various encodings of Boolean circuits and gives us the freedom to not actually have to give a concrete encoding of the Boolean circuits we use. For a detailed disquisition of uniformity we refer to [Vol99].

We define the following circuit-complexity classes. Let $\mathcal{K}_1, \mathcal{K}_2 \in \mathcal{F}^1(\mathbb{N})$ and $i \geq 0$:

$$
\begin{aligned}
\mathrm{FSIZE\text{-}DEPTH}_B(\mathcal{K}_1,\mathcal{K}_2) &\stackrel{\text{def}}{=} \{f : \text{there are functions } s \in \mathcal{K}_1 \text{ and } d \in \mathcal{K}_2 \text{ such that } f \\
&\qquad \text{can be computed by a log-space uniform family of} \\
&\qquad B\text{-circuits } (C^n)_{n\in\mathbb{N}} \text{ and for all } n \in \mathbb{N} \text{ holds} \\
&\qquad \mathrm{size}(C^n) \leq s(n) \text{ and } \mathrm{depth}(C^n) \leq d(n)\} \\
\mathrm{FNC}^i &\stackrel{\text{def}}{=} \mathrm{FSIZE\text{-}DEPTH}_{\{\textsc{And}, \textsc{Or}, \textsc{Not}\}}(\mathcal{P}, O((\log n)^i)) \\
\mathrm{FNC} &\stackrel{\text{def}}{=} \bigcup_{i\geq 0} \mathrm{FNC}^i \\
\mathrm{NC}^i &\stackrel{\text{def}}{=} \{A : c_A \in \mathrm{FNC}^i\}
\end{aligned}
$$

We remark, that sometimes FNC^i and NC^i are defined as nonuniform classes of functions. However, their most important interpretation is that they are the functions/sets that are computable/decidable efficiently by means of parallel computing. The open question of $\mathrm{FNC} \stackrel{?}{=} \mathrm{FP}$ gains its importance only in the light of this interpretation. Hence we choose the uniform definition. Also, logspace-uniformity is normally considered too weak for NC^i and FNC^i where $i \leq 1$, since $\mathrm{FNC}^1 \subseteq \mathrm{FL}$. But in the scope of this thesis this will not lead to any problems, so we skip the introduction of another uniformity.

2.4.4 Reductions and Complete Sets

We want to compare the difficulty of deciding two sets $A, B \subseteq \mathbb{N}$. For this we introduce the notion of "A being at most as difficult as B". This shall be the case if there is a function f such that for all $x \in \mathbb{N}$ holds: $x \in A$ if and only if $f(x) \in B$. In order to fit our notion, the power of f should not be too strong, especially, it should not be stronger than the power we need to decide A. We define $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ if and only if there is an $f \in \mathrm{FP}$ such that $x \in A \Leftrightarrow f(x) \in B$. We say, $A \leq_{\mathrm{m}}^{\mathrm{p}}$-*reduces to* B, or simply A *reduces to* B. We write $A \equiv_{\mathrm{m}}^{\mathrm{p}} B$ if and only if $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ and $B \leq_{\mathrm{m}}^{\mathrm{p}} A$. Let A be a set, $\mathcal{K}$ be a complexity class. We say A is $\leq_{\mathrm{m}}^{\mathrm{p}}$-hard for $\mathcal{K}$ if for all $B \in \mathcal{K}$ holds $B \leq_{\mathrm{m}}^{\mathrm{p}} A$, and A *is* $\leq_{\mathrm{m}}^{\mathrm{p}}$-*complete for* $\mathcal{K}$ if A is $\leq_{\mathrm{m}}^{\mathrm{p}}$-hard for $\mathcal{K}$ and $A \in \mathcal{K}$. If it is clear from the context, we say A *is complete for* $\mathcal{K}$ instead of A is $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete for $\mathcal{K}$.

Another reducibility is the *log-space reducibility*. A set A is log-space reducible to a set B ($A \leq_{\mathrm{m}}^{\log} B$) if and only if there is a function $f \in \mathrm{FL}$ such that for all x holds $x \in A$ if and only if $f(x) \in B$. We define $\leq_{\mathrm{m}}^{\log}$-hard and $\leq_{\mathrm{m}}^{\log}$-complete as in the case of the $\leq_{\mathrm{m}}^{\mathrm{p}}$-reducibility.

It is easy to observe, that both, $\leq_{\mathrm{m}}^{\log}$ and $\leq_{\mathrm{m}}^{\mathrm{p}}$, are reflexive and transitive.

3. Boolean Clones and Boolean Co-Clones

3.1 Clones and Co-Clones and their Galois Connection

In this section we define two special kinds of algebras, clones and co-clones, which are very natural kinds of algebras on functions and relations, respectively. We begin with the definition of clones. We will need some special operators for that, so let f, g, h be functions of arities ℓ, m, n, respectively, where $\ell \geq 1$, $m \geq 0$ and $n \geq 2$.

- $\mathrm{PI} \stackrel{\text{def}}{=} \mathrm{I}_1^2$ is the *presence of the identity function*. This is a 0-ary operator on the set of functions.
- $\mathrm{RF}(f)(x_1, x_2, \ldots, x_\ell) \stackrel{\text{def}}{=} f(x_2, \ldots, x_\ell, x_1)$ is the *rotation of variables in a function*.
- $\mathrm{TF}(h)(x_1, \ldots, x_{n-1}, x_n) \stackrel{\text{def}}{=} h(x_1, \ldots, x_n, x_{n-1})$ is the *transposition of the last variables in a function*.
- $\mathrm{LVF}(h)(x_1, \ldots, x_{n-1}) \stackrel{\text{def}}{=} h(x_1, \ldots, x_{n-1}, x_{n-1})$ is the *identification of the last variables*.
- $\mathrm{SUB}(f, g)(x_1, \ldots, x_{\ell-1}, y_1, \ldots, y_m) \stackrel{\text{def}}{=} f(x_1, \ldots, x_{\ell-1}, g(y_1, \ldots, y_m))$ is the *substitution of g in f*.

We subsume these five operation under the term *superposition*. Let $\mathrm{SUP} \stackrel{\text{def}}{=} \{\mathrm{PI}, \mathrm{RF}, \mathrm{TF}, \mathrm{LVF}, \mathrm{SUB}\}$. A *clone of functions*, or simply a *clone*, is a set of functions that is closed under superposition (clone is short for **cl**osed set of operations **on** a set, abbreviation due to P. Hall). Let $[A] \stackrel{\text{def}}{=} [A]_{\mathrm{SUP}}$ be the *clone generated by A*.

Clones are of special interest, because they consist exactly of those functions that can be described by formulas over a basic set. Take for example the set $B \stackrel{\text{def}}{=} \{+, \cdot, 0, 1\}$, i.e., the set of addition, multiplication, the constant 0, and the constant 1, all of which are functions over the natural numbers. Then $[B]$ is exactly the set of functions that can be described by polynomials with positive coefficients. The *order* of a set of functions B is $\max\{n : \text{ there is an } n\text{-ary function in } B\}$ if such a maximum exists and ∞ otherwise. The *order* of a clone A is $\min\{n : \text{ there is a base of order } n \text{ for } A\}$.

Co-clones are algebras over sets or relations and again we need to define some operators. Let $n \geq 0$, $m \geq 2$, R be an n-ary, and Q be an m-ary relation over some set A.

- $\mathrm{PE} \stackrel{\text{def}}{=} \mathrm{eq}$ is the *presence of the equality relation*. It is a 0-ary operator on the set of relations.
- $\mathrm{RR}(Q) \stackrel{\text{def}}{=} \{(a_2, \ldots, a_{n-1}, a_1) : (a_1, \ldots, a_{n-1}, a_n) \in Q\}$ is the *rotation of variables in a relation*.

– $\mathrm{TR}(Q) \stackrel{\text{def}}{=} \{(a_1, \ldots, a_n, a_{n-1}) : (a_1, \ldots, a_{n-1}, a_n) \in Q\}$ is the *transposition of the last variables in a relation.*
– $\mathrm{FVR}(R) \stackrel{\text{def}}{=} \{(b, a_1, \ldots, a_n) : b \in A \text{ and } (a_1, \ldots, a_n) \in R\}$ is the *introduction of a fictive variable in a relation.*
– $\mathrm{PRO}(Q) \stackrel{\text{def}}{=} \{(a_1, \ldots, a_{m-1}) :$ there is a $b \in A$ such that $(b, a_1, \ldots, a_{m-1}) \in Q\}$ is the *projection.*

We call these operations together with the intersection of sets *relational superposition*, or $\mathrm{SUP_R}$, for short. That means $\mathrm{SUP_R} \stackrel{\text{def}}{=} \{\mathrm{PE}, \mathrm{RR}, \mathrm{TR}, \mathrm{FVR}, \mathrm{PRO}, \cap\}$, where $\cap$ is the normal intersection of sets. A set of relations B over A is called *co-clone* if it is closed under relational superposition. Let $\langle B \rangle \stackrel{\text{def}}{=} [B]_{\mathrm{SUP_R}}$ be the *co-clone generated by* B.

Proposition 3.1. *Let B be a co-clone over a set A.*

1. *For every $n \geq 1$, the full relation A^n is in B.*
2. *If $R, S \in B$ then $R \times S \in B$.*
3. *Let R be an n-ary relation in B. Then*

$$\mathrm{DIAG}(R) \stackrel{\text{def}}{=} \{(a_1, \ldots, a_{n-1}) : (a_1, \ldots, a_{n-1}, a_{n-1}) \in R\}$$

is also in B. $\mathrm{DIAG}(R)$ *is called the* diagonalization *of R.*

Proof. 1. Since $\mathrm{eq} \in B$, $A = \mathrm{PRO}(\mathrm{eq}) \in B$. Furthermore, for $1 \leq i < n$, is $A^{i+1} = \mathrm{FVR}(A^i) \in B$.
2. Let R be m-ary and S be n-ary. Then $R \times A^n \in B$ and $A^m \times S \in B$, due to FVR and RR. Finally, $R \times S = (R \times A^n) \cap (A^m \times S)$.
3. Observe, that $T \stackrel{\text{def}}{=} \{(a_1, \ldots, a_{n-2}, b, b) : a_i, b \in A (1 \leq i < n-1)\} = A^{n-2} \times \mathrm{eq}$ is in B. Then $\mathrm{DIAG}(R) = T \cap R$.

❑

Obviously, the diagonalization can be applied to arbitrary positions, not only to the last ones. Analogous to the order of sets of functions, we define an order for sets of relations. If $\Gamma \subseteq \mathcal{R}(A)$, then the *order of* Γ is defined as $\max\{n :$ there is an $R \in \Gamma$ that has arity $n\}$, if such a maximum exists and ∞ otherwise. The *order of a co-clone* A is $\min\{n :$ there is a base B of A with order $n\}$; here we assume $n < \infty$ for all $n \in \mathbb{N}$.

We now define two mappings between sets of functions and sets of relations. For a set of relations B over a set A, let

$$\mathrm{Pol}(B) \stackrel{\text{def}}{=} \{f : f \in \mathcal{F}(A) \text{ and } f \sim R \text{ for all } R \in B\}$$

be the set of *polymorphisms of all relations in B*. For a set of functions B on a set A, let

$$\mathrm{Inv}(B) \stackrel{\text{def}}{=} \{R : R \in \mathcal{R}(A) \text{ and } f \sim R \text{ for all } f \in B\}$$

be the set of all relations that are *invariant under all functions of B*. Observe, that if Ω is a set of functions on a set A, then $(X, \Omega) \in \mathcal{L}((A, \Omega))$ if and only if $X \in \mathrm{Inv}(\Omega)$.

Proposition 3.2. *Let $C, D \subseteq \mathcal{F}(A)$ be sets of functions, and let $X, Y \subseteq \mathcal{R}(A)$ be sets of relations.*

1. *If $C \subseteq D$, then* $\mathrm{Inv}(D) \subseteq \mathrm{Inv}(C)$
2. *If $X \subseteq Y$, then* $\mathrm{Pol}(Y) \subseteq \mathrm{Pol}(X)$
3. $C \subseteq \mathrm{Pol}(\mathrm{Inv}(C))$
4. $X \subseteq \mathrm{Inv}(\mathrm{Pol}(X))$

Proof. Let $C \subseteq D$. Then every relation that is invariant under all functions of D is invariant under all functions of C. Therefore $\mathrm{Inv}(D) \subseteq \mathrm{Inv}(C)$. The proof of the second claim is analogous. Assume $C \not\subseteq \mathrm{Pol}(\mathrm{Inv}(C))$. Then there is an $f \in C$ and an $R \in \mathrm{Inv}(C)$ such that $f \not\sim R$. But since $R \in \mathrm{Inv}(C)$, it is preserved by all functions from C, including f. The proof of the fourth claim is analogous. ❑

A pair of mappings like $(\mathrm{Pol}, \mathrm{Inv})$ between two ordered sets like $2^{\mathcal{F}(A)}$ and $2^{\mathcal{R}(A)}$ that satisfies properties 1. through 4. is called a *Galois correspondence*.

Proposition 3.3. *Let A be a set, B be a set of relations over A, and C be a set of functions on A.*

1. $\mathrm{Pol}(B) = \mathrm{Pol}(\langle B \rangle) = [\mathrm{Pol}(B)]$
2. $\mathrm{Inv}(C) = \mathrm{Inv}([C]) = \langle \mathrm{Inv}(C) \rangle$

Proof. All equalities can be shown by an easy induction over superposition or relational superposition respectively. ❑

Hence, for every set of functions B on A, $\mathrm{Pol}(\mathrm{Inv}(B))$ is a clone containing $[B]$, and for every set of relations C over A, $\mathrm{Inv}(\mathrm{Pol}(C))$ is a co-clone containing $\langle C \rangle$.

Proposition 3.4 ([Pös79]). *Let A be a set with $|A| = k$, B be a set of functions on A and C be a set of relations over A.*

1. $[B] = \mathrm{Pol}(\mathrm{Inv}(B))$
2. $\langle C \rangle = \mathrm{Inv}(\mathrm{Pol}(C))$

Proof. Both inclusions from left to right are clear from Propositions 3.2 and 3.3. First, we show the inclusion from right to left for 1.: We define a special relation Δ_n for all $n \geq 1$. For $1 \leq i \leq n$, let

$$\delta_n^i \stackrel{\text{def}}{=} (\mathrm{enc}_{n,k}^{-1}(0)[i], \ldots, \mathrm{enc}_{n,k}^{-1}(k^n - 1)[i])$$

and

$$\Delta_n' \stackrel{\text{def}}{=} \{\delta_n^i : 1 \leq i \leq n\}$$

Example: For $A = \{0, 1\}$, list all the elements of A^n such that they appear as the rows of a matrix that has 2^n rows and n columns.

$$\begin{array}{rcccccc} & & \delta_n^1 & \delta_n^2 & & \delta_n^{n-1} & \delta_n^n \\ \mathrm{enc}_{n,k}^{-1}(0) & = & (0, & 0, & \ldots & 0, & 0) \\ \mathrm{enc}_{n,k}^{-1}(1) & = & (0, & 0, & \ldots & 0, & 1) \\ \mathrm{enc}_{n,k}^{-1}(2) & = & (0, & 0, & \ldots & 1, & 0) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathrm{enc}_{n,k}^{-1}(2^n-2) & = & (1, & 1, & \ldots & 1, & 0) \\ \mathrm{enc}_{n,k}^{-1}(2^n-1) & = & (1, & 1, & \ldots & 1, & 1) \end{array}$$

Then the columns of the matrix form the elements of Δ'_n. □

Now, let

$$\Delta_n \stackrel{\text{def}}{=} \Delta'_n \cup \{f(\delta_n^1, \delta_n^2, \ldots, \delta_n^n) : f \in [B] \text{ is } n\text{-ary}\}$$

Observe, that Δ_n is preserved by every function from $[B]$: Let $f \in [B]$ be m-ary and let $\alpha_1, \ldots, \alpha_m \in \Delta_n$. Then there are $f_1, \ldots, f_m \in [B]$ such that $\alpha_i = f_i(\delta_n^1, \ldots, \delta_n^n)$ by the definition of Δ_n (in the case where $\alpha_i \in \Delta'_n$, for some i, remember that $\mathrm{I}_i^n \in [B]$). Let $g(x_1, \ldots, x_n) \stackrel{\text{def}}{=} f(f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$. Obviously, $g \in [B]$ and $g(\delta_n^1, \ldots, \delta_n^n) = f(\alpha_1, \ldots, \alpha_m)$. Hence $\Delta_n \in \mathrm{Inv}([B])$. Now let $f \in \mathrm{Pol}(\mathrm{Inv}(B)) = \mathrm{Pol}(\mathrm{Inv}([B]))$ be n-ary. Then $f \sim \Delta_n$, i.e., $f(\delta_n^1, \ldots, \delta_n^n) \in \Delta_n$ and therefore $f \in [B]$.

Now we prove the direction from right to left of 2. Let Δ'_n be defined as above for all $n \geq 1$ and let

$$\Delta_n \stackrel{\text{def}}{=} \Delta'_n \cup \{f(\delta_n^1, \delta_n^2, \ldots, \delta_n^n) : f \in \mathrm{Pol}(C) \text{ is } n\text{-ary}\}$$

Claim 3.5. $\mathrm{Inv}(\mathrm{Pol}(C)) \subseteq \langle \bigcup_{n\geq 1}\{\Delta_n\}\rangle$.

Proof: Let R be an m-ary relation in $\mathrm{Inv}(\mathrm{Pol}(C))$ with $R = \{\alpha_1, \ldots, \alpha_t\}$. Then for all $1 \leq i \leq m$ there are $j_1, \ldots, j_m \in \{0, \ldots, k^t - 1\}$ such that $(\alpha_1[i], \ldots, \alpha_t[i]) = \mathrm{enc}_{t,k}^{-1}(j_i)$. With other words, R can be computed from $(\Delta'_t)^m$ by finitely many applications of RR, TR, and PRO. Furthermore, since $R \in \mathrm{Inv}(\mathrm{Pol}(C))$, it is closed under every function from $\mathrm{Pol}(C)$. Therefore, if we apply RR, TR, and PRO to $(\Delta_t)^m$ in the same way we applied them to $(\Delta'_t)^m$, the result is again R. Hence $R \in \langle\{\Delta_m\}\rangle$, and the proof of the claim is finished. □

So, the following inclusions hold:

$$\langle C\rangle \subseteq \mathrm{Inv}(\mathrm{Pol}(C)) \subseteq \langle \bigcup_{n\geq 1}\{\Delta_n\}\rangle$$

This means that we can finish our proof by showing $\langle \bigcup_{n\geq 1}\{\Delta_n\}\rangle \subseteq \langle C\rangle$. For this, it suffices to show $\Delta_n \in \langle C\rangle$ for all $n \geq 1$. Choose an R from $\{R : R \in \langle C\rangle \text{ and } \Delta'_n \subseteq R\}$ such that $|R|$ is minimal. Since $\Delta'_n \subseteq A^{k^n} \in \langle C\rangle$, there exists such an R. We show $\Delta_n = R$. Since $R \in \langle C\rangle \subseteq \mathrm{Inv}(\mathrm{Pol}(C))$ it follows that R must be closed under every $f \in \mathrm{Pol}(C)$ and therefore $\Delta_n \subseteq R$.

Suppose $\Delta_n \neq R$. Then there is an $\alpha \in R - \Delta_n$. Hence, the n-ary function f with $f(\delta_n^1, \ldots, \delta_n^n) = \alpha$ is not in $\mathrm{Pol}(C)$ and therefore there is a relation $S \in C$ such that

$f \not\sim S$. Let S be m-ary and let $\beta, \beta_1, \ldots, \beta_n \in S$ be such that $f(\beta_1, \ldots, \beta_n) = \beta \notin S$. For every $i \in \{1, \ldots, m\}$, let $h : \{1, \ldots, m\} \to \{1, \ldots, k^n\}$ such that $(\beta_1[i], \ldots, \beta_n[i]) = (\delta_n^1[h(i)], \ldots, \delta_n^n[h(i)])$. Due to the nature of the δ_n^i's, such a function h exists. Let

$$T \stackrel{\text{def}}{=} \{(a_1, \ldots, a_{k^n}, b_1, \ldots, b_m) : a_j, b_\ell \in A \text{ for } 1 \leq j \leq k^n, 1 \leq \ell \leq m \text{ and } b_i = a_{h(i)} \text{ for all } 1 \leq i \leq m\}$$

such that coordinates $k^n + i$ and $h(i)$ of every element from T are equal. Observe, that T is member of any co-clone over A. Then $S' \stackrel{\text{def}}{=} (R \times S) \cap T$ is in $\langle C \rangle$, since $R, S, T \in \langle C \rangle$. Suppose there were $a_1, \ldots, a_{k^n} \in A$ such that $(a_1, \ldots, a_{k^n}, b_1, \ldots, b_m) \in S'$, where $\alpha = (a_1, \ldots, a_{k^n})$. Then, since $a_{h(i)} = b_i$ for $1 \leq i \leq m$, it follows that $(b_1, \ldots, b_m) = \beta$, which is not possible, since $\beta \notin S$. Therefore, if S'' is the projection of S' on the first k^n coordinates, then $S'' \subseteq R$ but $\alpha \in R - S''$ and therefore $S'' \subsetneq R$. Finally, for all $1 \leq i \leq n$ and all $\gamma \in S$ holds $\delta_n^i \times \gamma$ is contained in both, $R \times S$ and T. Hence $\Delta_n' \subseteq S''$ which is a contradiction to the minimality of R. ❑

We get the following connection between the lattice of clones and the lattice of co-clones over finite sets.

Proposition 3.6. *Let A be a finite set and B, C be clones from $2^{\mathcal{F}(A)}$. Then B is predecessor of C in $\mathcal{L}(\mathcal{F}(A))$ if and only if* $\mathrm{Inv}(C)$ *is predecessor of* $\mathrm{Inv}(B)$ *in $\mathcal{L}(\mathcal{R}(A))$.*

Proof. Let $B \subset C$. It is clear from the definition, that $\mathrm{Inv}(C) \subseteq \mathrm{Inv}(B)$. Suppose $\mathrm{Inv}(C) = \mathrm{Inv}(B)$. Then $[C] = \mathrm{Pol}(\mathrm{Inv}(C)) = \mathrm{Pol}(\mathrm{Inv}(B)) = [B]$, which is a contradiction, hence $\mathrm{Inv}(C) \subset \mathrm{Inv}(B)$. In the same way, one can prove that if $\mathrm{Inv}(C) \subset \mathrm{Inv}(B)$ then $B \subset C$. Now suppose there were an $X \in \mathcal{L}(\mathcal{R}(A))$ such that $\mathrm{Inv}(C) \subset X \subset \mathrm{Inv}(B)$. Then $B \subset \mathrm{Pol}(X) \subset C$ which is a contradiction to $B \prec C$. The argument works analogous in the other direction. ❑

3.2 Clones of Boolean Functions

The Boolean domain $\{0, 1\}$ is the largest domain for which all clones of functions are known. Moreover, their inclusion structure, a finite base for each clone, and the order of each clone was discovered in the twenties of the last century by E. L. Post (although the result was published only in the forties [Pos41]). Therefore, we call the lattice of all clones of Boolean functions *Post's lattice*. Observe Figure 3.1 and Figure 3.2: They show the inclusion structure and an example for a base for each clone, respectively.

We are especially interested in the dual-atoms M, D, L, R_1, and R_0 of $\mathcal{L}(\mathrm{BF})$, where BF denotes the set of all Boolean functions. Figure 3.3 gives a definition of these clones.

Nevertheless, we need to characterize some additional types of Boolean functions.

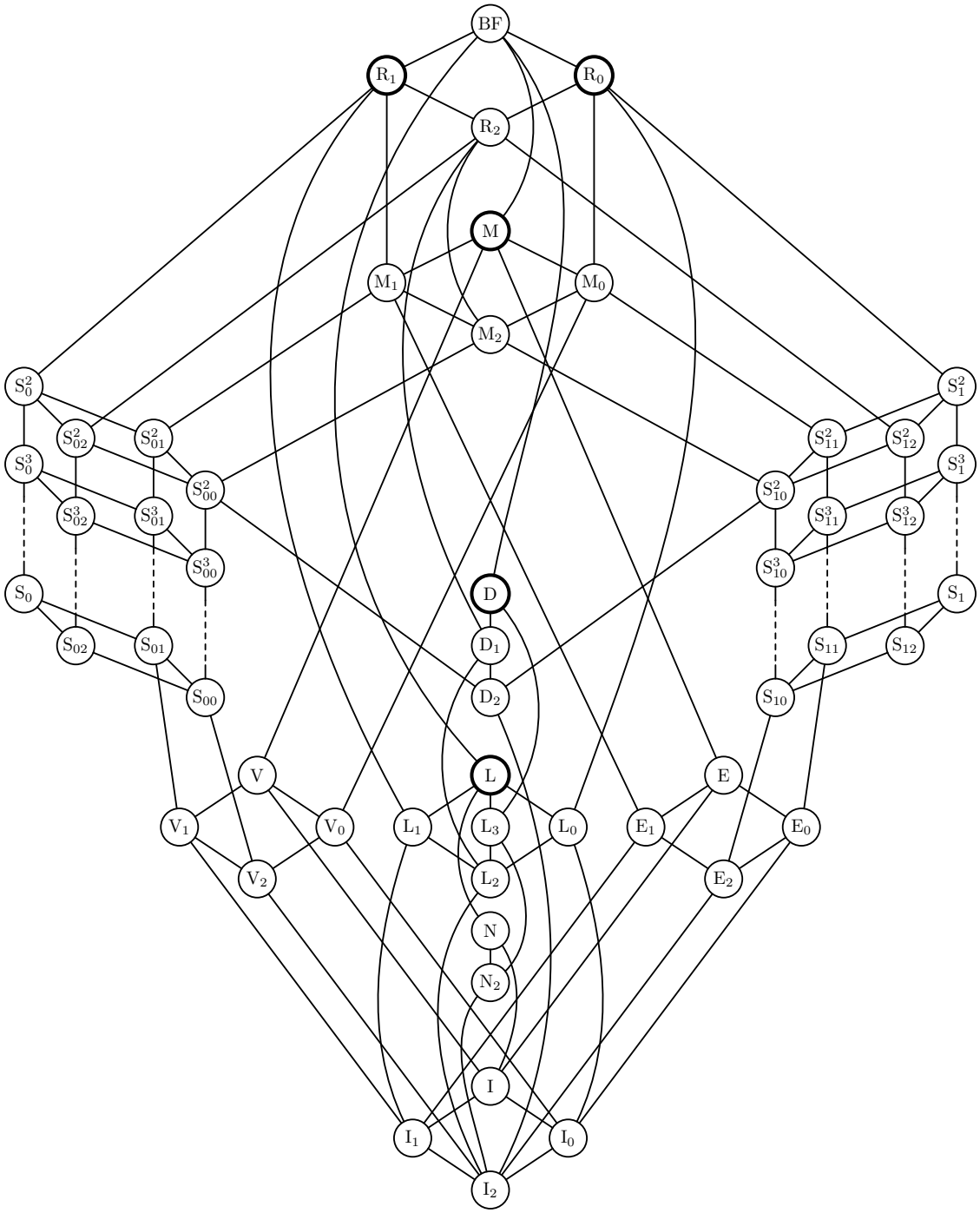


Fig. 3.1: Lattice $\mathcal{L}(\mathrm{BF})$ of Boolean Clones.

Class	**Definition**	**Order**	**Base(s)**
BF	all Boolean functions	2	$\{\text{AND}, \text{NOT}\}$
R_0	$\{ f \in \text{BF} \mid f \text{ is 0-reproducing} \}$	2	$\{\text{AND}, \text{XOR}\}$
R_1	$\{ f \in \text{BF} \mid f \text{ is 1-reproducing} \}$	2	$\{\text{OR}, x \oplus y \oplus 1\}$
R_2	$R_0 \cap R_1$	3	$\{\text{OR}, x \wedge (y \oplus z \oplus 1)\}$
M	$\{ f \in \text{BF} \mid f \text{ is monotonic} \}$	2	$\{\text{AND}, \text{OR}, c_0, c_1\}$
M_0	$M \cap R_1$	2	$\{\text{AND}, \text{OR}, c_1\}$
M_1	$M \cap R_0$	2	$\{\text{AND}, \text{OR}, c_0\}$
M_2	$M \cap R_2$	2	$\{\text{AND}, \text{OR}\}$
S_0^n	$\{ f \in \text{BF} \mid f \text{ is 0-separating of degree } n \}$	$n+1$	$\{\text{IMP}, \text{dual}(h_n)\}$
S_0	$\{ f \in \text{BF} \mid f \text{ is 0-separating} \}$	2	$\{\text{IMP}\}$
S_1^n	$\{ f \in \text{BF} \mid f \text{ is 1-separating of degree } n \}$	$n+1$	$\{x \wedge \overline{y},\ h_n\}$
S_1	$\{ f \in \text{BF} \mid f \text{ is 1-separating} \}$	2	$\{x \wedge \overline{y}\}$
S_{02}^n	$S_0^n \cap R_2$	$n+1$	$\{x \vee (y \wedge \overline{z}), \text{dual}(h_n)\}$
S_{02}	$S_0 \cap R_2$	$n+1$	$\{x \vee (y \wedge \overline{z})\}$
S_{01}^n	$S_0^n \cap M$	$n+1$	$\{\text{dual}(h_n), c_1\}$
S_{01}	$S_0 \cap M$	3	$\{x \vee (y \wedge z), c_1\}$
S_{00}^n	$S_0^n \cap R_2 \cap M$	$n+1$	$\{x \vee (y \wedge z), \text{dual}(h_n)\}$
S_{00}	$S_0 \cap R_2 \cap M$	3	$\{x \vee (y \wedge z)\}$
S_{12}^n	$S_1^n \cap R_2$	$n+1$	$\{x \wedge (y \vee \overline{z}), h_n\}$
S_{12}	$S_1 \cap R_2$	3	$\{x \wedge (y \vee \overline{z})\}$
S_{11}^n	$S_1^n \cap M$	$n+1$	$\{h_n, c_0\}$
S_{11}	$S_1 \cap M$	3	$\{x \wedge (y \vee z), c_0\}$
S_{10}^n	$S_1^n \cap R_2 \cap M$	$n+1$	$\{x \wedge (y \vee z), h_n\}$
S_{10}	$S_1 \cap R_2 \cap M$	3	$\{x \wedge (y \vee z)\}$
D	$\{ f \mid f \text{ is self-dual} \}$	3	$\{x\overline{y} \vee x\overline{z} \vee (\overline{y} \wedge \overline{z})\}$
D_1	$D \cap R_2$	3	$\{xy \vee x\overline{z} \vee y\overline{z}\}$
D_2	$D \cap M$	3	$\{xy \vee yz \vee xz\}$
L	$\{ f \mid f \text{ is linear} \}$	2	$\{\text{XOR}, c_1\}$
L_0	$L \cap R_0$	2	$\{\text{XOR}\}$
L_1	$L \cap R_1$	2	$\{\text{EQ}\}$
L_2	$L \cap R_2$	3	$\{x \oplus y \oplus z\}$
L_3	$L \cap D$	3	$\{x \oplus y \oplus z \oplus c_1\}$
V	$\{ f \mid f \text{ is an } n\text{-ary OR-function or a constant function} \}$	2	$\{\text{OR}, c_0, c_1\}$
V_0	$[\text{OR}] \cup [c_0]$	2	$\{\text{OR}, c_0\}$
V_1	$[\text{OR}] \cup [c_1]$	2	$\{\text{OR}, c_1\}$
V_2	$[\text{OR}]$	2	$\{\text{OR}\}$
E	$\{ f \mid f \text{ is an } n\text{-ary AND-function or a constant function} \}$	2	$\{\text{AND}, c_0, c_1\}$
E_0	$[\text{AND}] \cup [c_0]$	2	$\{\text{AND}, c_0\}$
E_1	$[\text{AND}] \cup [c_1]$	2	$\{\text{AND}, c_1\}$
E_2	$[\text{AND}]$	2	$\{\text{AND}\}$
N	$[\text{NOT}] \cup [c_0] \cup [c_1]$	1	$\{\text{NOT}, c_0\}$, $\{\text{NOT}, c_1\}$
N_2	$[\text{NOT}]$	1	$\{\text{NOT}\}$
I	$[c_1] \cup [c_0]$	0	$\{c_0, c_1\}$
I_0	$[c_0]$	0	$\{c_0\}$
I_1	$[c_1]$	0	$\{c_1\}$
I_2	$[\emptyset]$	0	$\{\emptyset\}$

Fig. 3.2: Bases for Boolean Clones. Here h_n is the $n+1$-ary Boolean function such that $h_n(x_1, \ldots, x_{n+1}) = 1$ if and only if $\Sigma_{i=1}^{n+1} x_i \geq n$. Furthermore, $\text{dual}(f)(a_1, \ldots, a_n) \stackrel{\text{def}}{=} \neg f(\overline{a_1}, \ldots, \overline{a_n})$.

Name	Symbol	Description
Monotonic functions	M	Let $n \geq 1$ and $\alpha = (a_1, \ldots, a_n) \in \{0,1\}^n$ and $\beta = (b_1, \ldots, b_n) \in \{0,1\}^n$. We write $\alpha \leq \beta$ if and only if $a_i \leq b_i$ for all $1 \leq i \leq n$. An n-ary Boolean function f is called *monotonic* if and only if for all $\alpha, \beta \in \{0,1\}^n$ with $\alpha \leq \beta$ follows $f(\alpha) \leq f(\beta)$.
Self-Dual functions	D	An n-ary Boolean function f is called *self-dual* if for all $(a_1, \ldots, a_n) \in \{0,1\}^n$ holds $f(a_1, \ldots, a_n) = \neg f(\overline{a_1}, \ldots, \overline{a_n})$, i.e., if and only if $f = \text{dual}(f)$.
Linear functions	L	An n-ary Boolean function is called *linear* if and only if there are constants $c_0, \ldots, c_n \in \{0,1\}$ such that $f(x_1, \ldots, x_n)$ can be described by the propositional formula $c_0 \oplus c_1 x_1 \oplus \cdots \oplus c_n x_n$. We call this formula the *linear normal form* of the function.
a-Reproducing functions	R_0, R_1	For $a \in \{0,1\}$, let the Boolean function f be called *a-reproducing* if and only if $f(a, \ldots, a) = a$.

Fig. 3.3: Functions forming dual-atoms in $\mathcal{L}(\text{BF})$.

Definition 3.7.
Let $M \subseteq \{0,1\}^n$ and $a \in \{0,1\}$. Then, M is a-separating *if and only if there is an $i \in \{1, \ldots, n\}$ such that for all $(a_1, \ldots, a_n) \in M$ we have $a_i = a$. Let M be called* a-separating of degree m *if and only if all $T \subseteq M$ with $|T| = m$ are a-separating. An n-ary Boolean function f is called* a-separating (of degree m)*, if $f^{-1}(a)$ is a-separating (of degree m).*
The dual *of a Boolean function is defined as follows: The n-ary function g is the dual function of f if and only if for all $a_1, \ldots, a_n \in \{0,1\}$ holds $g(a_1, \ldots, a_n) = \neg f(\overline{a_1}, \ldots, \overline{a_n})$. Let $\text{dual}(f) \stackrel{\text{def}}{=} g$ and for a set of Boolean functions A, let $\text{dual}(A) \stackrel{\text{def}}{=} \{\text{dual}(f) : f \in A\}$.*

Observe, that $\text{dual}(\text{dual}(f)) = f$ for all Boolean functions f, and hence for all sets of Boolean functions A we have $\text{dual}(\text{dual}(A)) = A$. Dual functions are very important when studying Post's lattice, because for every clone A of Boolean functions, $\text{dual}(A)$ is a clone, too [Pos41, JGK70]. Moreover, if a Boolean function is specified by a Boolean circuit, then another one, that specifies the dual function can easily be found, as the following proposition sugests:

Proposition 3.8 ([JGK70]). *Let $h, f, g_1, \ldots, g_n$ be Boolean functions such that*

$$h(x_1, \ldots, x_m) = f(g_1(x_1, \ldots, x_m), \ldots, g_n(x_1, \ldots, x_m)).$$

Then

$$\text{dual}(h)(x_1, \ldots, x_m) = \text{dual}(f)(\text{dual}(g_1)(x_1, \ldots, x_m), \ldots, \text{dual}(g_n)(x_1, \ldots, x_m))$$

Proof. Observe the following equation:

$$\begin{aligned}\mathrm{dual}(h)(x_1,\ldots,x_m) &= \neg h(\overline{x_1},\ldots,\overline{x_m})\\ &= \neg f(g_1(\overline{x_1},\ldots,\overline{x_m}),\ldots,g_n(\overline{x_1},\ldots,\overline{x_m}))\\ &= \neg f(\neg\neg g_1(\overline{x_1},\ldots,\overline{x_m}),\ldots,\neg\neg g_n(\overline{x_1},\ldots,\overline{x_m}))\\ &= \mathrm{dual}(f)(\mathrm{dual}(g_1)(x_1,\ldots,x_m),\ldots,\mathrm{dual}(g_n)(x_1,\ldots,x_m))\end{aligned}$$

❑

Hence, the dual circuit can be found by replacing the function of every gate by its dual one. If the circuit was built over a finite set of Boolean functions then this task can be done very efficiently.

In Figure 3.1, A and $\mathrm{dual}(A)$ are symmetrical with respect to the symmetry axis going through BF and I_2. For example, V_1 is the dual of E_0 and vice versa and M is its own dual as is D. The case of D is special insofar, as every function from D is its own dual whereas, for example, for every function f in M, the function $\mathrm{dual}(f)$ is also in M without necessarily $f = \mathrm{dual}(f)$.

3.3 Boolean Constraints, and Co-Clones of Boolean Relations

There are many applications in computer science where a program has to choose from a pool of objects those which fulfill certain properties. For example, think of a database containing the data of different cars. A user could want to see all cars which are yellow and have a maximum speed of 200 kmh^{-1} and have a steering wheel. So, the objects have to satisfy a list of so-called constraints simultaneously. We want to formalize this notion for Boolean objects and connect that notion to that of Boolean functions.

An n-ary *Boolean constraint* R is a subset of $\{0,1\}^n$. There are several ways to define a Boolean constraint. One way is to explicitly list all elements of the constraint, for example $R = \{(0,0,1),(0,1,0),(1,0,1),(1,1,1)\}$. A similar way is to define an order on $\{0,1\}^n$ and describe R by setting bits in a string of length 2^n. The *characteristic string* of an n-ary Boolean relation is a word $w \in \{0,1\}^{2^n}$ such that $w[i+1] = 1$ if and only if $(\mathrm{bin}_n(i)[1],\ldots,\mathrm{bin}_n(i)[n]) \in R$. For example, the characteristic string of the above relation is 01100101. However, since an n-ary Boolean constraint can contain up to 2^n elements, the above methods can be rather inefficient. Therefore, we often use Boolean functions to describe Boolean relations in the obvious way: A Boolean function f *corresponds* to the Boolean relation R if and only if $R = \{\alpha : f(\alpha) = 1\}$ is the set of satisfying assignments for f. So, we can use propositional formulas and Boolean circuits to describe Boolean relations.

We want to be able to express that several constraints hold simultaneously, as in the car example. For that we introduce *constraint formulas*.

Definition 3.9. *Let* Γ *be a set of Boolean constraints,* $n \geq 1$*, and* $R_1, \ldots, R_n \in \Gamma \cup \bigcup_{i \geq 1} \{0,1\}^i$ *be constraints of arity* $m_1, \ldots, m_n$*, respectively. Let* $k \stackrel{\text{def}}{=} \Sigma_{i=1}^{n} m_i$ *and let* $\{i_1, \ldots, i_k\}$ *be a set of indices that is equal to* $\{1, \ldots, \ell\}$ *for some* $\ell \leq k$*. Then*

$$F \stackrel{\text{def}}{=} R_1(x_{i_1}, \ldots, x_{i_{m_1}}) \wedge R_2(x_{i_{m_1+1}}, \ldots, x_{i_{m_1+m_2}}) \wedge \cdots \wedge R_n(x_{i_{k-m_n+1}}, \ldots, x_{i_k})$$

is a Γ-constraint formula*.* $F(x_{i_1}, \ldots, x_{i_k})$ *describes the constraint*

$$\begin{aligned} \{(a_1, \ldots, a_\ell) : \quad & (a_{i_1}, \ldots, a_{i_{m_1}}) \in R_1 \text{ and} \\ & (a_{i_{m_1+1}}, \ldots, a_{i_{m_1+m_2}}) \in R_2 \text{ and} \\ & \qquad \vdots \\ & (a_{i_{k-m_n+1}}, \ldots, a_{i_k}) \in R_n\}. \end{aligned}$$

Let $\mathrm{CF}(\Gamma) \stackrel{\text{def}}{=} \{R : \text{ there is a } \Gamma\text{-constraint formula } F \text{ describing } R\}$*.*

In this definition, all variables in constraint formulas need to be from a set $\{x_1, \ldots, x_\ell\}$ for some ℓ. We use this definition to fix the order of the variables. In practice, when this order is not necessary or is obvious, as in the case of x, y, z, we also use other variables. The reason we allow the R_i's to be not only from Γ but also from $\bigcup_{n \geq 1} \{0,1\}^n$ is that we always want to be able to express constraints with fictive variables.

If R is described by a Γ-constraint formula F, we write $R = F$, for simplicity. We say, a Γ-constraint formula F of arity n is *satisfied by* $\alpha \in \{0,1\}^n$ if and only if α is contained in the relation described by F. It is easy to see, that every constraint that can be described by a Γ-constraint formula is included in $\langle\Gamma\rangle$. We can even show more.

Proposition 3.10. *Let* $\Omega = \{\mathrm{PE}, \mathrm{RR}, \mathrm{TR}, \mathrm{FVR}, \cap\}$*. For every set of Boolean relations* Γ*,* $\mathrm{CF}(\Gamma \cup \{\mathrm{eq}\}) = [\Gamma]_\Omega$ *holds.*

Proof. Let $R \in \mathrm{CF}(\Gamma)$ and let $n \geq 1$, and R_i, m_i, k, ℓ, $\{i_1, \ldots, i_k\}$, and F be as in Definition 3.9 and let R be described by F. All R_i are in $[\Gamma]_\Omega$, hence $T \stackrel{\text{def}}{=} R_1 \times \cdots \times R_n \in [\Gamma]_\Omega$ (see Proposition 3.1). Let T' be derived from T by diagonalizing all variables at positions s, t such that $i_s = i_t$. Then there is a permutation π on $\{1, \ldots, \ell\}$ such that $(a_{\pi(1)}, \ldots, a_{\pi(\ell)}) \in T'$ if and only if $(a_1, \ldots, a_\ell) \in R$. Therefore we can derive R from T' using RR and TR.

On the other hand, let $R \in [\Gamma]_\Omega$. If $R \in \Gamma \cup \{\mathrm{eq}\}$ is n-ary, then the formula $R(x_1, \ldots, x_n)$ describes R. Now let $R_1, R_2 \in [\Gamma]_\Omega$ be n-ary. Then $\mathrm{RR}(R_1)$ and $\mathrm{TR}(R_1)$ can obviously be described by a formula in which the variables are permuted properly. $\mathrm{FVR}(R_1)$ can be expressed by the formula $S(x_1) \wedge R(x_2, \ldots, x_{n+1})$, where $S \stackrel{\text{def}}{=} \{0,1\}$ (remember that we are allowed to use all full constraints $\{0,1\}^i$). Finally, $R_1 \cap R_2$ can be described by the formula $R_1(x_1, \ldots, x_n) \wedge R_2(x_1, \ldots, x_n)$. ❑

So relations that can be expressed with constraints are very close to the elements of co-clones. Therefore $\mathrm{CF}(\Gamma)$ is often called the *weak co-clone* of Γ. The differences between

co-clone and weak co-clone are obviously due to the operators PE and PRO. The operator PRO is the more powerful one in many contexts, since the equality of two variables can be expressed as follows: If $\mathrm{eq}(x, y)$ is used in a conjunct of a constraint formula, remove it, and replace every occurrence of y in the rest of the formula by x. The new constraint formula describes a relation of smaller arity, so it is syntactically different, but since all tuples of the original constraint always have the same value at positions x and y, the difference can be neglected for most applications. Therefore we introduce existentially quantified constraint formulas.

Definition 3.11. *Let* Γ *be a set of Boolean constraints and* $F(x_1, \ldots, x_n, y_1, \ldots, y_m)$ *be a* Γ*-constraint formula that describes the constraint* R*. Then*

$$G \stackrel{\text{def}}{=} \exists x_1 \ldots \exists x_n F(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

is an existentially quantified Γ-constraint formula*. It describes the relation*

$$\{(b_1, \ldots, b_m) : \text{ there are } a_1, \ldots, a_n \in \{0,1\} \text{ such that } (a_1, \ldots, a_n, b_1, \ldots, b_m) \in R\}.$$

Let $\mathrm{EQCF}(\Gamma)$ *be the set of all relations that can be expressed by an existentially quantified* Γ*-constraint formula.*

The (existentially quantified) constraint formulas F_1 and F_2 are *equivalent* if they describe the same relation. We write $F_1 \equiv F_2$;

Proposition 3.12. *For every set of Boolean relations* Γ *holds* $\mathrm{EQCF}(\Gamma \cup \{\mathrm{eq}\}) = \langle\Gamma\rangle$.

Proof. The inclusion from left to right follows from Proposition 3.10 and the existence of PRO in $\mathrm{SUP}_{\mathrm{R}}$. For the other direction, it suffices to note that $\mathrm{EQCF}(\Gamma \cup \{\mathrm{eq}\})$ is closed under PRO. ❑

The similarities between weak Boolean co-clones and Boolean co-clones are often sufficient to simplify proofs about constraints very much by utilizing properties of co-clones. As an example, have a look at the following problems.

Problem:	Constraint Satisfaction Problem, $\mathrm{CSP}(\Gamma)$
Input:	A Γ-constraint formula F
Question:	Does the relation described by F contain at least one element? With other words, is F satisfiable?

There is a well-known dichotomy theorem by T. Schaefer for the computational complexity of the CSP problem [Sch78]. The claim of the theorem is that for every set of Boolean constraints Γ, the problem $\mathrm{CSP}(\Gamma)$ is either NP-complete or in P. Schaefer characterized all sets of constraints Γ for which $\mathrm{CSP}(\Gamma) \in \mathrm{P}$ as follows. Let R be a constraint of arity n.

– R is called *a-valid* for $a \in \{0,1\}$ if and only if $(a, \ldots, a) \in R$.

- R is called *Horn* (*anti-Horn*) if and only if it can be described by a propositional formula in conjunctive normal form, where every conjunction contains at most one positive (negative) literal.
- R is called *bijunctive* if and only if it can be described by a propositional formula in conjunctive normal form, where every conjunct contains at most two literals.
- R is called *affine* if and only if it can be described by a conjunction, where every conjunct is a formula over $\oplus$.
- R is called *Schaefer* if R is one of the above.

A set of constraints Γ is called 0-valid, 1-valid, Horn, etc., if every $R \in \Gamma$ is 0-valid, 1-valid, Horn, etc., respectively.

Theorem 3.13 ([Sch78]). *If a set of Boolean constraints* Γ *is Schaefer, then* CSP(Γ) *is in* P*. In all other cases,* CSP(Γ) *is* NP*-complete.*

The original proof of this theorem is very complicated and technical. The easier part of the proof is to show that the CSP problem is easy for Schaefer formulas. But it is not at all clear, why Schaefer formulas are the only ones, for which the problem is easy and to prove so is the hard part. We will see in the following, that this task becomes easier when making use of the structure of the Boolean co-clones. This proof is due to P. Kolaitis but is implicit in [JCG97, JCG99, Dal00].

For that, we define the constraint satisfaction problems for existentially quantified constraints.

Problem:	Constraint Satisfaction Problem for Existentially Quantified Constraints, ECSP(Γ)
Input:	An existentially quantified Γ-constraint formula F
Question:	Does the relation described by F contain at least one element? With other words, is F satisfiable?

In the next proposition we show that, with a little modification, the two constraint satisfaction problems have an equivalent computational complexity. We modify the ECSP-problem insofar, as we always allow the eq-relation. Then we can use Proposition 3.12 and therefore the structure of the Boolean co-clones.

Proposition 3.14. *For all sets* Γ *of Boolean relations,* CSP(Γ) $\equiv_m^p$ ECSP($\Gamma \cup \{\text{eq}\}$) *holds.*

Proof. The direction from left to right is obvious. For the other direction, let $F = \exists x_1 \ldots \exists x_n G(x_1, \ldots, x_n, y_1, \ldots, y_m)$ be a quantified $(\Gamma \cup \{\text{eq}\})$-constraint formula, where G a is $(\Gamma \cup \{\text{eq}\})$-constraint formula. Obviously, F is satisfiable if and only if G is. Let G' be the Γ-constraint formula that can be derived from G by removing every conjunct of the form $\text{eq}(x_i, x_j)$ from G and replacing every occurrence of x_j in G by x_i. If there are no other conjuncts in G, let $G' \stackrel{\text{def}}{=} S(x)$, where $S = \{0, 1\}$. Then G' is satisfiable if and only if G is. ❑

So it suffices to prove Theorem 3.13 for existentially quantified constraint formulas. We know that we can express with such formulas over a set $\Gamma \cup \{\text{eq}\}$ exactly the relations from $\langle\Gamma\rangle$ which is a Boolean co-clone. We want to use the structure of the Boolean co-clones. Since we know the structure of the lattice of Boolean clones and because of Proposition 3.6, we know the structure of Boolean co-clones, too: It is the same with reversed inclusion structure. For every clone A, we denote $\text{Inv}(A)$ with IA for **i**nvariant of A and, additionally, $\text{BR} \stackrel{\text{def}}{=} \text{II}_2$. The structure of Boolean co-clones is given in Figure 3.4.

Let us now have a look at the Schaefer classes.

Proposition 3.15 ([CKS01]). *Let Γ be a set of constraints.*

1. *Γ is 0-valid if and only if Γ is closed under c_a, the 0-ary constant function a.*
2. *Γ is Horn if and only if it is closed under* And*, and it is anti-Horn if and only if it is closed under* Or*.*
3. *Γ is bijunctive if and only if it is closed under the 3-ary majority function $xy \vee xz \vee yz$.*
4. *Γ is affine if and only if it is closed under the linear function $x \oplus y \oplus z$.*

Observe, that $\{c_a\}$, $\{\text{And}\}$, $\{\text{Or}\}$, $\{xy \vee xz \vee yz\}$, and $\{x \oplus y \oplus z\}$ are bases of the Boolean clones I_a, E_2, V_2, D_2, and L_2, respectively. Therefore the Schaefer classes are descriptions of Boolean co-clones! They are even descriptions of dual-atoms in the lattice $\mathcal{L}(\text{BR})$. The remaining dual-atom of this lattice is IN_2. It contains all relations R such that $\alpha \in R$ if and only if $\overline{\alpha} \in R$, since these relations have to be closed under negation. We call such constraints *complementive*. If we consult Figure 3.4, we see that every set of constraints is contained in one of the Schaefer classes except for those which describe a base for IN_2 or BR. So, it remains to show that $\text{CSP}(\Gamma)$ is NP-complete, if Γ describes a base for IN_2 or even for BR.

If Γ describes a base for BR, then we can express every possible 3-ary clause as an existential quantified constraint formula, i.e., we can express all of $x \vee y \vee z$, $x \vee y \vee \overline{z}$, …, $\overline{x} \vee \overline{y} \vee \overline{z}$. Because of Proposition 3.14 and since the equality relation can be removed as described above, we can reduce 3-SAT to $\text{CSP}(\Gamma)$.

Let NAE-3-SAT be the problem to satisfy a 3-CNF formula such that in each clause there is a literal that is not satisfied. Schaefer showed that this problem is NP-complete [Sch78] in his proof of Theorem 3.13. If Γ is complete for IN_2, then the relation $\text{R}_{\text{nae}} \stackrel{\text{def}}{=} \{0,1\}^3 - \{(0,0,0),(1,1,1)\}$ is in $\langle\Gamma\rangle$ since R_{nae} is complementive. This relation can be used to reduce NAE-3-SAT to $\text{CSP}(\Gamma)$.

This finishes the alternative proof of Schaefer's Theorem. The advantage in this approach is that we can make use of the known structure of the Boolean co-clones thus avoiding a lot of cumbersome case-distinctions. Also, we get a better impression where the essential parts of Schaefer's proof lie: The upper bounds for the easy cases and the lower bound for NAE-3-SAT.

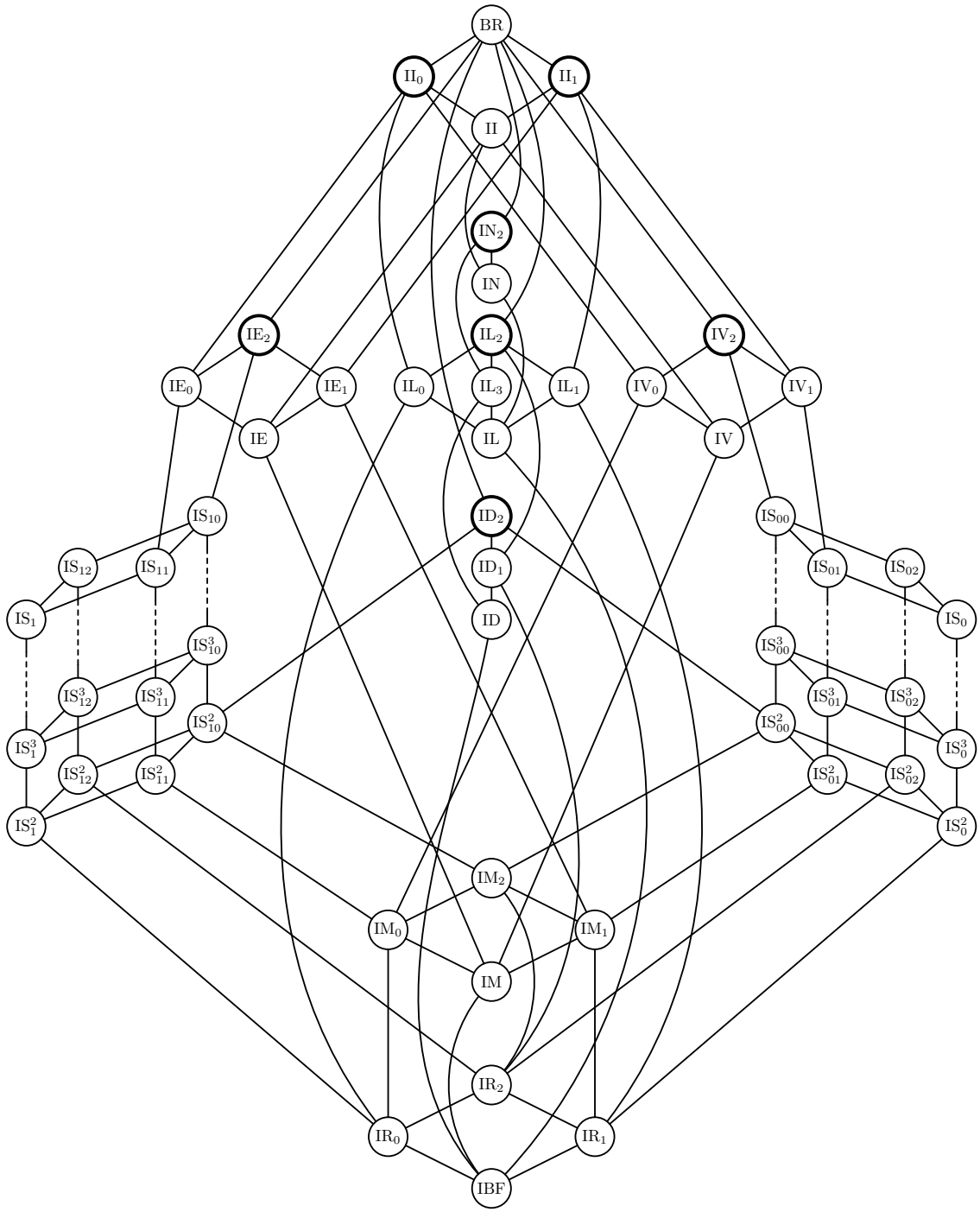


Fig. 3.4: Lattice $\mathcal{L}(\mathrm{BR})$ of Boolean Co-Clones

3.4 Minimal Bases for Boolean Co-Clones

We want to give a list of bases for the Boolean co-clones, as we did for Boolean clones in Figure 3.2. Based on Proposition 2.2, we can check whether a set Γ of Boolean relations is a base of a co-clone A:

Algorithm 3.16. First, we test, whether Γ is in A. Since we know a finite base B of $\mathrm{Pol}(A)$, this can be done by checking whether every relation in Γ is closed under every function in B. Then we check whether Γ is contained in a dual-atom of $\mathcal{L}(A)$. Finally, Γ is a base of A if and only if the first test is successful and the second is not.

We introduce a duality for Boolean relations as we did for Boolean functions, since it will help us to find bases in a faster way.

Definition 3.17. *The* dual Boolean relation *of the n-ary Boolean relation R is defined as*

$$\mathrm{dual}(R) \stackrel{\mathrm{def}}{=} \{(\overline{a_1}, \ldots, \overline{a_n}) : (a_1, \ldots, a_n) \in R\}.$$

For a set of Boolean relations Γ, let $\mathrm{dual}(\Gamma) \stackrel{\mathrm{def}}{=} \{\mathrm{dual}(R) : R \in \Gamma\}$.

Let us look at some properties of duality for Boolean relations.

Proposition 3.18. *Let R be a Boolean relation, let Γ be a set of Boolean relations, and let f be a Boolean function.*

1. *$f \sim R$ if and only if* $\mathrm{dual}(f) \sim \mathrm{dual}(R)$.
2. $\mathrm{Pol}(\mathrm{dual}(\Gamma)) = \mathrm{dual}(\mathrm{Pol}(\Gamma))$.

Proof. The second point is immediate from the first one, which remains to prove. Let f be m-ary and let $\alpha_1, \ldots, \alpha_m \in R$. Then $\mathrm{dual}(f)(\overline{\alpha_1}, \ldots, \overline{\alpha_m}) = \overline{f(\alpha_1, \ldots, \alpha_n)} \in \mathrm{dual}(R)$, since $f \sim R$. This shows the direction from left to right; the other direction is true, since $\mathrm{dual}(\mathrm{dual}(R)) = R$ and $\mathrm{dual}(\mathrm{dual}(f)) = f$. ❑

Therefore the symmetry, we know from $\mathcal{L}(\mathrm{BF})$, can also be found in $\mathcal{L}(\mathrm{BR})$, and the dual of every co-clone can be found as a mirror image with respect to the symmetry axis going through BR and IBF in Figure 3.4. Furthermore, we need to find only bases for "one half" of the lattice, since the other half can be derived by duality.

Proposition 3.19. *If Γ_1 and Γ_2 are bases for co-clones B_1 and B_2, then $\Gamma_1 \cup \Gamma_2$ is a base of the join $C \stackrel{\mathrm{def}}{=} \langle B_1 \cup B_2 \rangle$ of B_1 and B_2.*

Proof. Obviously, $\Gamma_1 \cup \Gamma_2 \subseteq C$. Therefore either $A \stackrel{\mathrm{def}}{=} \langle \Gamma_1 \cup \Gamma_2 \rangle = C$ or A is a sub-co-clone of C. Since there is a relation in Γ_1 that is not in Γ_2 and vice versa, A is a co-clone containing both, B_1 and B_2, i.e., $B_1 \subsetneq A$ and $B_2 \subsetneq A$. Therefore $A \neq C$ is a contradiction to the minimality of C. ❑

This enables us to derive bases for all co-clones from the bases of just a few ones. We define some special relations for $m, n \in \mathbb{N}$.

$$\begin{aligned}
\mathrm{OR}^m &\stackrel{\text{def}}{=} \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i > 0\} = \{0,1\}^m - \{(0,\ldots,0)\} \\
\mathrm{AND}^m &\stackrel{\text{def}}{=} \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i = m\} = \{(1,\ldots,1)\} \\
\mathrm{NOR}^m &\stackrel{\text{def}}{=} \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i = 0\} = \{(0,\ldots,0)\} \\
\mathrm{NAND}^m &\stackrel{\text{def}}{=} \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i \neq m\} = \{0,1\}^m - \{(0,\ldots,0)\} \\
n\text{-IN-}m &\stackrel{\text{def}}{=} \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i = n\} \\
\mathrm{EVEN}^m &\stackrel{\text{def}}{=} \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i \text{ is even}\} \\
\mathrm{ODD}^m &\stackrel{\text{def}}{=} \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i \text{ is odd}\} \\
\mathrm{NAE}^m &\stackrel{\text{def}}{=} \{0,1\}^m - \{(0,\ldots,0),(1,\ldots,1)\}
\end{aligned}$$

Furthermore, we define the relation

$$\mathrm{DUP} \stackrel{\text{def}}{=} \{0,1\}^3 - \{(0,1,0),(1,0,1)\}$$

Proposition 3.20. *Let $m \geq 2$.*

$$\begin{aligned}
\langle \mathrm{OR}^m \rangle &= \mathrm{IS}_0^m \\
\langle x \rangle &= \mathrm{IR}_1 \\
\langle x \oplus y \rangle &= \mathrm{ID} \\
\langle \mathrm{EVEN}^4 \rangle &= \mathrm{IL} \\
\langle x \to y \rangle &= \mathrm{IM} \\
\langle \mathrm{DUP} \rangle &= \mathrm{IN} \\
\langle x \vee y \vee \overline{z} \rangle &= \mathrm{IV}
\end{aligned}$$

Proof. Let f be a k-ary Boolean function. First, let $f \in \mathrm{IS}_0^m$. By definition, if $\alpha_1,\ldots,\alpha_m \in f^{-1}(\{0\})$ then $\{\alpha_1,\ldots,\alpha_m\} \subseteq \{0,1\}^{i-1} \times \{0\} \times \{0,1\}^{k-i}$ for some i. This is the same as saying that if $f(\alpha_1[1],\ldots,\alpha_1[k]) = 0$, $f(\alpha_2[1],\ldots,\alpha_2[k]) = 0,\ldots,f(\alpha_m,[1],\ldots,\alpha_m[k]) = 0$ then there is some i such that $(\alpha_1[i],\alpha_2[i],\ldots,\alpha_m[i]) = (0,0,\ldots,0)$. Hence, $f \sim \mathrm{OR}^m$.

Now, let $f \notin \mathrm{IS}_0^m$. Then there are $\alpha_1,\ldots,\alpha_m \in \{0,1\}^k$ such that for all $i \in \{1,\ldots,m\}$ we have $f(\alpha_i) = 0$ and for all $j \in \{1,\ldots,k\}$ there is an $i \in \{1,\ldots,m\}$ such that $\alpha_i[j] = 1$. Since $\{(\alpha_1[i],\alpha_2[i],\ldots,\alpha_m[i]) : 1 \leq i \leq k\} \subseteq \mathrm{OR}^m$, we have $f \not\sim \mathrm{OR}^m$.

The remaining points can easily be verified using Algorithm 3.16. ❑

These basic cases suffice to find bases for all Boolean co-clones, using duality and Proposition 3.19, using Figure 3.4. In Figure 3.5 we indicate in the third column how the base could be derived from the above basic ones.

This systematic approach yields quite complicated bases for some co-clones as, e.g., the important ones IN_2 and BR. So we additionally give at least one base of minimal order for each co-clone. That these are in fact bases can be verified using Algorithm 3.16.

To determine the order of a co-clone, we proceed as follows. First, note that $\langle\emptyset\rangle = \mathrm{IBF}$, $\langle x\rangle = \mathrm{IR}_1$, $\langle\overline{x}\rangle = \mathrm{IR}_0$, and $\langle x, \overline{x}\rangle = \mathrm{IR}$). Since we have thus identified all co-clones that have order of at most one, every other co-clone has to have an order of at least two. The next proposition tells us something about the nature of co-clones of order two.

Proposition 3.21. *If A is not a sub-co-clone of ID_2, and Γ is a set of relations containing just relations with arity less then 3, then $\langle\Gamma\rangle \neq A$.*

Proof. We know from Proposition 3.15 that exactly the bijunctive relations are closed under $\{xy \vee xz \vee yz\}$, which is a base of D_2. Hence ID_2 coincides with the class of relations that are bijunctive and can hence be expressed by 2-CNF formulas. Every 2-ary relation can be expressed by such a formula and if A had a base with just relations of arity 2 or less, then $A \subseteq \mathrm{ID}_2$ would hold, which is a contradiction. ❑

Hence every non-bijunctive co-clone has order greater than two. It turns out, that we find a base of order three for nearly all of them. Propositions 3.22 and 3.23 deal with the remaining cases.

Proposition 3.22. *The order of* IL *and* IL_3 *is 4.*

Proof. Observe, that $\{\mathrm{EVEN}^4\}$ is a base for IL, and $\{\mathrm{ODD}^4\}$ is a base for IL_3. Therefore 4 is an upper bound for the orders of IL and IL_3. Since all 2-ary relations are bijunctive, it suffices to show in the case of IL that all 3-ary relations in IL are also in IBF and in the case of IL_3 that all 3-ary relations in IL_3 are in ID. This is an easy exercise, since we have to check only a small number of relations: If $R \in \mathrm{IL}_3$, then R is complementive and affine. Since R is complementive, one half of the possible elements of R is determined by the other half. Hence we have to check the constraints with the following characteristic strings:

$$\begin{array}{c} 0000\ 0000 \\ 0001\ 1000 \\ 0010\ 0100 \\ \vdots \\ 1110\ 0111 \\ 1111\ 1111 \end{array}$$

Using Algorithm 3.16 we can see that all these constraints are in IBF or ID which are both bijunctive co-clones. ❑

Co-Clone	**O.**	**Remark**	**Base(s)**
BR	3	$\mathrm{IL} \cup \mathrm{IM} \cup \mathrm{IR}_2$	$\{\mathrm{EVEN}^4, x \to y, x, \overline{x}\}, \{\text{1-IN-3}\}$,$\{x \to (y \oplus z)\}$
II_1	3	$\mathrm{IL} \cup \mathrm{IM} \cup \mathrm{IR}_1$	$\{\mathrm{EVEN}^4, x \to y, x\}$,$\{x \vee (x \oplus z)\}$
II_0	3	$\mathrm{IL} \cup \mathrm{IM} \cup \mathrm{IR}_0$	$\{\mathrm{EVEN}^4, x \to y, \overline{x}\}$,$\{\mathrm{DUP}, x \to y\}$
II	3	$\mathrm{IL} \cup \mathrm{IM}$	$\{\mathrm{EVEN}^4, x \to y\}$
IN_2	3	$\mathrm{IN} \cup \mathrm{IL}_3$	$\{\mathrm{DUP}, \mathrm{EVEN}^4, x \oplus y\}, \{\mathrm{NAE}^3\}$
IN	3		$\{\mathrm{DUP}\}$
IE_2	3	dual of IV_2	$\{\overline{x} \vee \overline{y} \vee z, x, \overline{x}\}$
IE_0	3	dual of IV_1	$\{\overline{x} \vee \overline{y} \vee z, \overline{x}\}$
IE_1	3	dual of IV_0	$\{\overline{x} \vee \overline{y} \vee z, x\}$
IE	3	dual of IV	$\{\overline{x} \vee \overline{y} \vee z\}$
IV_2	3	$\mathrm{IV} \cup \mathrm{IR}_2$	$\{x \vee y \vee \overline{z}, x, \overline{x}\}$
IV_1	3	$\mathrm{IV} \cup \mathrm{IR}_1$	$\{x \vee y \vee \overline{z}, x\}$
IV_0	3	$\mathrm{IV} \cup \mathrm{IR}_0$	$\{x \vee y \vee \overline{z}, \overline{x}\}$
IV	3		$\{x \vee y \vee \overline{z}\}$
IL_3	4	$\mathrm{IL} \cup \mathrm{ID}$	$\{\mathrm{EVEN}^4, x \oplus y\}$,$\{\mathrm{ODD}^4\}$
IL_2	3	$\mathrm{IL} \cup \mathrm{IR}_2$	$\{\mathrm{EVEN}^4, x, \overline{x}\}$, every $\{\mathrm{EVEN}^n, \{(1)\}\}$, $n \geq 3$ is odd
IL_1	3	$\mathrm{IL} \cup \mathrm{IR}_1$	$\{\mathrm{EVEN}^4, x\}$,$\{\mathrm{ODD}^3\}$
IL_0	3	$\mathrm{IL} \cup \mathrm{IR}_0$	$\{\mathrm{EVEN}^4, \overline{x}\}$,$\{\mathrm{EVEN}^3\}$
IL	4		$\{\mathrm{EVEN}^4\}$
ID_2	2	$\mathrm{ID} \cup \mathrm{IM}$	$\{x \oplus y, x \to y\}$,$\{x\overline{y} \vee \overline{x}yz\}$
ID_1	2	$\mathrm{ID} \cup \mathrm{IR}_1$	$\{x \oplus y, x\}$, every $R \in \{\{(a_1, a_2, a_3), (b_1, b_2, b_3)\} \mid \exists c \in \{1, 2\}$ such that $\Sigma_{i=1}^{3} a_i = \Sigma_{i=1}^{3} b_i = c\}$
ID	2		$\{x \oplus y\}$
IS_{10}	∞	dual of IS_{00}	$\{\mathrm{NAND}^m \mid m \geq 2\} \cup \{x, \overline{x}, x \to y\}$
IS_{10}^m	m	dual of IS_{00}^m	$\{\mathrm{NAND}^m, x, \overline{x}, x \to y\}$
IS_{11}	∞	dual of IS_{01}	$\{\mathrm{NAND}^m \mid m \geq 2\} \cup \{x \to y\}$
IS_{11}^m	m	dual of IS_{01}^m	$\{\mathrm{NAND}^m, x \to y\}$
IS_{12}	∞	dual of IS_{02}	$\{\mathrm{NAND}^m \mid m \geq 2\} \cup \{x, \overline{x}\}$
IS_{12}^m	m	dual of IS_{02}^m	$\{\mathrm{NAND}^m, x, \overline{x}\}$
IS_{00}	∞	$\mathrm{IS}_0 \cup \mathrm{IR}_2 \cup \mathrm{IM}$	$\{\mathrm{OR}^m \mid m \geq 2\} \cup \{x, \overline{x}, x \to y\}$
IS_{00}^m	m	$\mathrm{IS}_0^n \cup \mathrm{IR}_2 \cup \mathrm{IM}$	$\{\mathrm{OR}^m, x, \overline{x}, x \to y\}$
IS_{01}	∞	$\mathrm{IS}_0 \cup \mathrm{IM}$	$\{\mathrm{OR}^m \mid m \geq 2\} \cup \{x \to y\}$
IS_{01}^m	m	$\mathrm{IS}_0^m \cup \mathrm{IM}$	$\{\mathrm{OR}^m, x \to y\}$
IS_{02}	∞	$\mathrm{IS}_0 \cup \mathrm{IR}_2$	$\{\mathrm{OR}^m \mid m \geq 2\} \cup \{x, \overline{x}\}$
IS_{02}^m	m	$\mathrm{IS}_0^n \cap \mathrm{IR}_2$	$\{\mathrm{OR}^m, x, \overline{x}\}$
IS_1	∞	dual of IS_0	$\{\mathrm{NAND}^m \mid m \geq 2\}$
IS_0	∞	$\cup_{m \geq 2} \mathrm{IS}_0^m$	$\{\mathrm{OR}^m \mid m \geq 2\}$
IS_1^m	m	dual of IS_0^m	$\{\mathrm{NAND}^m\}$
IS_0^m	m		$\{\mathrm{OR}^m\}$
IM_2	2	$\mathrm{IM} \cup \mathrm{IR}_2$	$\{x \to y, x, \overline{x}\}$,$\{x \to y, \overline{x \to y}\}$, $\{x\overline{y} \wedge (u \to v)\}$
IM_0	2	$\mathrm{IM} \cup \mathrm{IR}_0$	$\{x \to y, \overline{x}\}$,$\{\overline{x} \wedge (y \to z)\}$
IM_1	2	$\mathrm{IM} \cup \mathrm{IR}_1$	$\{x \to y, x\}$,$\{x \wedge (y \to z)\}$
IM	2		$\{x \to y\}$
IR_2	1	$\mathrm{IR}_0 \cup \mathrm{IR}_1$	$\{x, \overline{x}\}$,$\{x\overline{y}\}$
IR_1	1		$\{x\}$
IR_0	1	dual of IR_1	$\{\overline{x}\}$
IBF	0		$\{\mathrm{eq}\}$, $\{\emptyset\}$

Fig. 3.5: Bases for all Boolean co-clones

Proposition 3.23. *For $n \geq 2$ and $a \in \{0,1\}$, the orders of IS_a^n, IS_{a2}^n, IS_{a1}^n, and IS_{a0}^n are n.*

Proof. It suffices to show the claim for $a = 1$, because the case $a = 0$ follows by duality. We define $h_n(x_1, \ldots, x_{n+1}) \stackrel{\text{def}}{=} \bigvee_{i=1}^{n+1} x_1 \wedge x_2 \wedge \cdots \wedge x_{i-1} \wedge x_{i+1} \wedge \cdots \wedge x_{n+1}$ to be the $n+1$-ary function that maps inputs to 1 if and only if at least n of the inputs are set to 1 (so h_n is the function version of n-IN-$(n+1)$). We show that, for $2 \leq m < n$, every m-ary relation that is closed under h_n is already closed under h_{n-1}. From this follows that if a relation of arity less than n is in IS_1^n then it is in fact already included in IS_1^{n-1} since for all $n \geq 2$ the set $\{x \wedge \overline{y}, h_n\}$ is a base for the clone S_1^n.

Let R be a relation with arity m where $2 \leq m < n$ that is closed under h_n and suppose it were not closed under h_{n-1}. Then there are $\alpha_1, \ldots, \alpha_n \in R$ such that $\delta \stackrel{\text{def}}{=} h_{n-1}(\alpha_1, \ldots, \alpha_n) \notin R$. For $i \in \{1, \ldots, n\}$, let $\beta_i \stackrel{\text{def}}{=} h_n(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i, \alpha_i, \alpha_{i+1}, \ldots, \alpha_n)$. Since h_n preserves R, all of the β_i are in R and therefore for all of them holds $\beta_i \neq \delta$. We have a look at the coordinates of β_i. For that let $k \in \{1, \ldots, m\}$. If $\alpha_i[k] = 1$, then $\beta_i[k] = \delta[k]$. If $\alpha_i[k] = 0$ and there is a $j \neq i$ such that $\alpha_j[k] = 0$ then again $\beta_i[k] = \delta[k]$. But if $\alpha_i[k] = 0$ and $\alpha_j[k] = 1$ for all $j \neq i$, then $\beta_i[k] = 0$ and $\delta[k] = 1$.

So, if $\beta_i[k]$ differs from $\delta[k]$ then $\alpha_i[k] = 0$ and $\alpha_j[k] = 1$ for all $j \neq i$. Therefore for all $j \neq i$ we have $\beta_j[k] = \delta[k]$. Following the pigeon hole principle, there has to be a j such that $\beta_j = \delta$, which is a contradiction.

The proofs for IS_{12}^n, IS_{11}^n, and IS_{10}^n are analogous. ❑

Corollary 3.24. *There is no finite set of constraints Γ such that*

$$\langle\Gamma\rangle \in \{\mathrm{IS}_0, \mathrm{IS}_1, \mathrm{IS}_{00}, \mathrm{IS}_{01}, \mathrm{IS}_{02}, \mathrm{IS}_{10}, \mathrm{IS}_{11}, \mathrm{IS}_{12}\}.$$

Proof. Follows from Proposition 2.1, since $\mathcal{L}(\langle\Gamma\rangle)$ is not dual-atomar. ❑

Finally, note that for every co-clone for which there is a finite base, there is a base that contains just one relation.

Proposition 3.25. *Let $\Gamma = \{R_1, R_2, \ldots, R_k\}$ be a base of a co-clone A. Then $R \stackrel{\text{def}}{=} \{R_1 \times R_2 \times \cdots \times R_k\}$ is a base of A.*

Proof. Obviously, R is closed under all functions from $\mathrm{Pol}(A)$, so $R \in A$. Using permutation of variables and projection, we can derive from R all R_i for $1 \leq i \leq k$, so all relations of A can be generated. ❑

4. Complexity Classifications for Problems on the Boolean Domain

Among the most important problems in complexity theory are the satisfiability problems for propositional formulas SAT and Boolean circuits CSAT. The proof by S. Cook [Coo71] of the NP-completeness of the CSAT problem is established by building a circuit that simulates the computation of a polynomial-time bounded Turing machine. In 1979, H. R. Lewis [Lew79] made the observation that these circuits do not need to be built over a complete set of Boolean functions. He showed that NP-completeness of CSAT can be proven with circuits over a set of Boolean functions B if and only if the function $x \wedge \overline{y}$ can be expressed with a circuit over B. In 1999, S. Reith and K. Wagner published a connection of Lewis's result with universal algebra [RW00, Rei01]: The set $\{x \wedge \overline{y}\}$ is a base of the clone S_1 and for S_1-circuits, i.e., circuits that are built over a set of Boolean functions B with $[B] = \mathrm{S}_1$, the satisfiability problem is NP-hard, already. It is not too difficult to see, that for all other circuits the respective CSAT-problem is in P. For example, if the circuits are build over a set of functions that is complete for a sub-clone of R_1, then every function described by such a circuit is 1-reproducing and hence satisfiable.

In this chapter we study the complexity of similar problems. We give a formal definition of CSAT.

Problem:	Circuit Satisfiability Problem for B-Circuits, CSAT(B)
Input:	A B circuit C with arity n for some $n \in \mathbb{N}$
Question:	Is there an $\alpha \in \{0,1\}^n$ such that $C(\alpha) = 1$?

Theorem 4.1 ([Lew79, RW00]). *If B is a set of Boolean functions such that $\mathrm{S}_1 \subseteq [B]$, then* CSAT($B$) *is* NP*-complete otherwise* CSAT(B) $\in$ P*.*

4.1 Satisfiability with Exceptions

As mentioned, CSAT(B) is trivial if $[B] \subseteq \mathrm{R}_1$. However, the tractability rests on just one tuple: If we would ask whether there is an $\alpha \neq (1, \ldots, 1)$ that satisfies such a B-circuit, it is not at all clear that this problem would be easy. We define a more general version that can be difficult even so a circuit is obviously satisfiable by a finite number of assignments.

Problem:	Circuit Satisfiability Problem with Selected Exceptions, SelectSAT(B)
Input:	A B-circuit C of arity n and tuples $\alpha_1, \ldots, \alpha_m$ ($m \geq 0$)
Question:	Is there a $\beta \in \{0,1\}^n - \{\alpha_1, \ldots, \alpha_m\}$ such that $C(\beta) = 1$?

The following theorem shows that the situation does not change much: The normal B-circuit satisfiability problem is NP-complete if and only if $\mathrm{S}_1 \subseteq [B]$, whereas the problem SelectSAT(B) is NP-complete if and only if $\mathrm{S}_{12} \subseteq [B]$. Therefore the tractability of CSAT(B) if $B \subseteq \mathrm{R}_1$ does not solely depend on the fact that B is 1-reproducing.

Theorem 4.2. *Let $B \subseteq$ BF be a set of Boolean functions. If $\mathrm{S}_{12} \subseteq [B]$ then* SelectSAT(B) *is* NP*-complete. In all other cases* SelectSAT(B) $\in$ P*.*

Proof. For that, we need a restricted version of SelectSAT(B):

Problem: SAT*(B)
Input: A B-circuit C with arity $n \geq 0$
Question: Is there an $\alpha \in \{0,1\}^n - \{(1,\ldots,1)\}$ such that $C(\alpha) = 1$?

Claim 4.3. If $\mathrm{S}_{12} \subseteq [B]$ then SAT*(B) is NP-hard.

Proof: Looking at Figure 3.1, we see that $[\mathrm{S}_{12} \cup \{c_0\}] = \mathrm{S}_1$, hence with Theorem 4.1, CSAT($\mathrm{S}_{12} \cup \{c_0\}$) is NP-complete. We will now reduce CSAT($\mathrm{S}_{12} \cup \{c_0\}$) to SAT*($\mathrm{S}_{12}$). Given a $(B \cup \{c_0\})$-circuit C on variables $\{x_1,\ldots,x_n\}$, we use a new variable x as a replacement for the constant c_0. Thus we obtain an S_{12}-circuit C' that behaves as C does, for all inputs that set x to 0. Observe, that the Boolean function $x \wedge (y \vee \bar{z})$ belongs to S_{12}. Therefore we can build an S_{12}-circuit $\hat{C}$ which is equivalent to

$$C'(x_1,\ldots,x_n,x) \wedge \bigwedge_{i=1}^{n} (x_i \vee \bar{x})$$

and that is at most polynomially larger than C'. Observe that C has a satisfying assignment if and only if there is an input different from $(1,\ldots,1)$ that makes $\hat{C}$ output 1. We conclude that SAT*(S_{12}) is NP-hard. □

Now, let $\mathrm{S}_{12} \subseteq [B]$. We easily reduce SAT*($\mathrm{S}_{12}$) to SelectSAT($B$), because SAT*($B$) is a restriction of SelectSAT(B). It remains to prove that in all remaining cases, i.e., if $B \subseteq A$ for an $A \in \{\mathrm{M}, \mathrm{S}_0^2, \mathrm{D}, \mathrm{L}\}$, then SelectSAT($B$) is in P.

If $B \subseteq \mathrm{M}$, then for every satisfying assignment α of a B-circuit C holds that all β with $\beta \geq \alpha$ are also satisfying. We use this in the following polynomial-time algorithm that decides for a SelectSAT(B) instance $(C, \alpha_1, \ldots, \alpha_k)$ whether C has a satisfying assignment besides $\alpha_1, \ldots, \alpha_k$:

```
function sSAT(C, α_1, ..., α_k)
  return sSATbelow(1^n, C, α_1, ..., α_k)

function sSATbelow(β, C, α_1, ..., α_k)
  if C(β) = 1 then
    if β ∉ {α_1, ..., α_k} return true
```

```
  else
    go through all γ ∈ {0,1}^n with hamming-distance(β,γ)=1 and γ ≤ β
      if (sSATbelow(γ, C, α_1, ..., α_k) = true) return true
return false
```

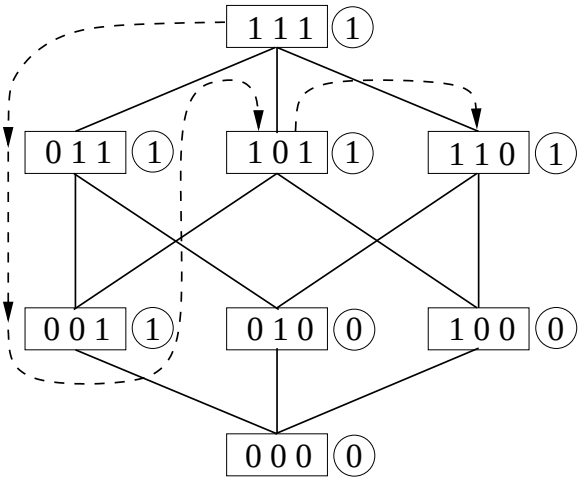


Fig. 4.1: An example of possible assignments for a 3-ary Boolean function (boxes) together with possible function values of a monotonic function (circles). Since all satisfying assignments are grouped in the "upper" part of the lattice, due to the monotonicity of the function, a constant number of satisfying assignments can be found very fast. The dotted line illustrates a possible search path of the algorithm.

See Figure 4.1 for an example of the work of the algorithm. Remember, that $\gamma \leq \beta$ if and only if γ is coordinate-wise less than or equal to β. The first `if`-block in `sSATbelow` is reached only if β is a satisfying assignment and for every β there are at most n assignments γ that are smaller than β and have hamming-distance 1. So, if C has $c \leq k$ satisfying assignments, the algorithm needs no more than $c \cdot n$ calls to `sSATbelow` to find all of them and to verify that there are no further satisfying assignments. If there are more than k satisfying assignments, the algorithm will stop accepting after no more than $k \cdot n + 1$ calls to `sSATbelow`.

If $B \subseteq \mathrm{L}$, we can use the linear normal form $c_0 \oplus \bigoplus_{i=1}^{n} c_i x_i$, which exists for every linear Boolean function and can be found quickly, to solve SelectSAT(B) efficiently.

If $B \subseteq \mathrm{D}$ then all B circuits are trivially satisfiable, since for every assignment α $f(\alpha) \neq f(\bar{\alpha})$, if f is self-dual.

Finally, if $B \subseteq \mathrm{S}_0^2$ then each n-ary B-circuit has at least 2^{n-1} satisfying assignments: No 0-separating set $M \subseteq \{0,1\}^n$ of degree 2 can contain α and $\bar{\alpha}$. ❑

4.2 Frozen Variables

An important aspect in connection with satisfiability is that of *frozen variables*. A variable of a satisfiable Boolean circuit is *frozen* if it has the same value in every assignment that is satisfying for the circuit. The problem of frozen variables is of importance in all areas, where large amounts of data have to be stored, as for example, in databases. If there is an attribute that takes the same value for all objects stored in a database, then this attribute is redundant and can be removed.

A. Krokhin and P. Jonsson examined the complexity of the problem of finding frozen variables in constraint satisfaction problems [JK04]. They show that the problem for general constraint satisfaction problems is complete for the complexity class DP. We examine variations of this problem for Boolean circuits. First, let us give an exact definition of a frozen variable.

Definition 4.4. *Let C be a Boolean circuit with variables $x_1, \ldots, x_n$ and let $1 \leq i \leq n$. The variable x_i is a* frozen variable of C *if and only if*

$$\#\{\alpha[i] : \alpha \in \{0,1\}^n \text{ and } C(\alpha) = 1\} = 1.$$

With other words, x_i is a frozen variable of C if and only if C is satisfiable and there is an $a \in \{0,1\}$ such that for every α with $C(\alpha) = 1$ holds $\alpha[i] = a$. If $V \subseteq \{x_1, \ldots, x_n\}$ then V is *frozen in* C if and only if every $x \in V$ is a frozen variable of C.

Problem:	Frozen Variables Problem for Boolean Circuits, FV(B)
Input:	A B-circuit C with input variables V, and $V' \subseteq V$
Question:	Is V' frozen in C?

Note, that the frozen variables problem is a generalization of the well-known unique satisfiability problem. We define this problem for B-circuits.

Problem:	Unique Satisfiability Problem for B-circuits, UNIQUE-SAT(B)
Input:	A B-circuit C over the set of variables V
Question:	Is V frozen in C, i.e., does C have exactly one satisfying assignment?

It is rather obvious, that UNIQUE-SAT(BF) is in DP: An n-ary circuit C is in UNIQUE-SAT(BF) if and only if

$$\begin{array}{l} \text{1. there is an } \alpha \in \{0,1\}^n \text{ such that } C(\alpha) = 1 \text{ and} \\ \text{2. for all } \alpha_1, \alpha_2 \in \{0,1\}^n \text{ holds: If } C(\alpha_1) = C(\alpha_2) = 1 \text{ then } \alpha_1 = \alpha_2. \end{array} \tag{4.1}$$

Then 1. is an NP condition and 2. is a coNP-condition. However, there is an oracle against UNIQUE-SAT(BF) being DP-complete [BG82].

Instead of testing whether all of the variables of a given set are frozen in a Boolean circuit, we could also ask whether there is such a variable at all. This is the idea of the following, existential version of the frozen variable problem.

Problem: The Existential Frozen Variable Problem for B-circuits, $\exists\mathrm{FV}(B)$
Input: A B-circuit C
Question: Is there a frozen variable in C?

Per definition, a variable can only be frozen in a circuit C, if that circuit can be satisfied. This adds a lot to the complexity of the frozen variable problem. If our main motivation is to just find frozen variables without caring whether the given circuit is actually satisfiable, we are interested in the following additional problem.

Problem: The Auditing Problem for B-Circuits, $\mathrm{AUDIT}(B)$
Input: A B-circuit C
Question: Does C have a frozen variable or is C unsatisfiable?

The goal of this section is to classify the complexity of all these problems with respect to all possible clones B. Let us begin with some upper bounds.

Proposition 4.5. *For every set B of Boolean functions holds*

1. $\mathrm{FV}(B)$ *is in* DP.
2. $\mathrm{AUDIT}(B)$ *is in* coNP.
3. $\exists\mathrm{FV}(B)$ *is in* DP.
4. $\mathrm{UNIQUE\text{-}SAT}(B)$ *is in* DP.

Proof. Let C be an n-ary B-circuit over a set of variables V and let $V' \subseteq V$. For the first point, note that V' is frozen in C if and only if

1. C is satisfiable and
2. for all $\alpha_0, \alpha_1 \in \{0,1\}^n$ holds that if $C(\alpha_0) = C(\alpha_1) = 1$ then for all i with $x_i \in V'$ we have $\alpha_0[i] = \alpha_1[i]$.

The first condition is in NP and the second one is in coNP, hence $\mathrm{FV}(B)$ is in DP. For the second point note that $C \in \overline{\mathrm{AUDIT}(B)}$ if and only if there are $\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n \in \{0,1\}^n$ such that $C(\alpha_i) = C(\beta_i) = 1$ and $\alpha_i[i] \neq \beta_i[i]$ for $1 \leq i \leq n$. This is an NP-condition.

The third point follows, since $\exists\mathrm{FV}(B)$ is a conjunction of $\mathrm{AUDIT}(B)$ and $\mathrm{CSAT}(B)$. We already saw the fourth point in (4.1). ❑

Therefore, we have upper bounds for our problems. Let us now search for lower bounds, and start with $\mathrm{FV}(B)$ and $\exists\mathrm{FV}(B)$.

Lemma 4.6. *Let $B \subseteq \mathrm{BF}$ be a set of Boolean functions. If $\mathrm{D}_1 \subseteq [B] \subseteq \mathrm{D}$ or $\mathrm{S}_{02} \subseteq [B] \subseteq \mathrm{R}_1$, then $\exists\mathrm{FV}(B)$ and $\mathrm{FV}(B)$ are* coNP*-complete.*

Proof. Observe that for all B that satisfy the conditions above all B-circuits are trivially satisfiable. Therefore the only difficulty that remains is to test whether there are frozen variables, and hence $\mathrm{FV}(B)$ and $\exists\mathrm{FV}(B)$ are in coNP.

Let $\mathrm{S}_{02} \subseteq [B] \subseteq \mathrm{R}_1$. We will reduce $\overline{\mathrm{CSAT}(\mathrm{R}_0)}$ to $\exists\mathrm{FV}(B)$ and $\mathrm{FV}(B)$. This gives the needed coNP-hardness, since $\overline{\mathrm{CSAT}(\mathrm{R}_0)}$ is coNP-complete, see Theorem 4.1. For that let C

be an R_0-circuit over $\{x \vee (y \wedge \overline{z}), 0\}$, which is a base of R_0. We build a new circuit C' from C by taking a variable x that does not occur in C and by replacing every occurrence of 0 in C with x. Then $C' \vee x$ is a S_{02}-circuit, because $\{\text{OR}, x \vee (y \wedge \overline{z})\} \subseteq [S_{02}]$ (see Figure 3.2: $\text{OR} \in V_2 \subset S_{02}$ and $x \vee (y \wedge \bar{z})$ is a base of S_{02}). Since $C' \vee x$ can be satisfied by every assignment that sets x to 1, the only possibly frozen variable in $C' \vee x$ is x. Furthermore, x is a frozen variable in $C' \vee x$ if and only if C is not satisfiable.

Now let $D_1 \subseteq [B] \subseteq D$. We will reduce the coNP-complete problem $\text{EQ}(D_1)$ [Rei03] to $\exists\,\text{FV}(B)$ and $\text{FV}(B)$. For a clone B, this problem is defined as follows:

Problem:	Circuit Equivalence Problem for B-circuits, $\text{EQ}(B)$
Input:	Two B-circuits C_1 and C_2
Question:	Do C_1 and C_2 describe the same Boolean function?

Let C_1 and C_2 be two n-ary D_1-circuits. Let x be a variable that occurs neitherin C_1 nor in C_2 and let $C' \stackrel{\text{def}}{=} x \oplus C_1 \oplus C_2$. Since $x \oplus y \oplus z$ is a function in $L_2 \subseteq D_1$ (see Figures 3.1, 3.2), C' is a D_1-circuit. Note that $C''(x, y, z) = xy \vee xz \vee yz$ is a D_1-circuit, too. The reduction g works as follows:

$$g(C_1, C_2) \stackrel{\text{def}}{=} \begin{cases} C'' & \text{, if } C_1(0^n) \neq C_2(0^n) \text{ or } C_1(1^n) \neq C_2(1^n) \\ C' & \text{otherwise} \end{cases}$$

We claim that $C_1 \equiv C_2$ holds if and only if $g(C_1, C_2) \in \exists\,\text{FV}(D_1)$ if and only if $(g(C_1, C_2), \{x\}) \in \text{FV}(B)$. For that let $C_1 \equiv C_2$. Then $g(C_1, C_2) = C'$. Since $C_1 \oplus C_2 \equiv 0$ the circuit C' is satisfied if and only if x is satisfied. Thus x is a frozen variable and therefore $g(C_1, C_2) \in \exists\,\text{FV}(B)$ and $(g(C_1, C_2), \{x\}) \in \text{FV}(B)$.

On the other hand, if $C_1 \not\equiv C_2$ then we have two cases. If $C_1(0^n) \neq C_2(0^n)$ or $C_1(1^n) \neq C_2(1^n)$ then $g(C_1, C_2) = C''$, which is a circuit without frozen variables. If $C_1(0^n) = C_2(0^n)$ and $C_1(1^n) = C_2(1^n)$ then $g(C_1, C_2) = C'$. Since $C_1 \not\equiv C_2$, there is an assignment $\alpha \in \{0,1\}^n$ such that $C_1(\alpha) \neq C_2(\alpha)$. Therefore $1 = 0 \oplus C_1(\alpha) \oplus C_2(\alpha) = 1 \oplus C_1(0^n) \oplus C_2(0^n) = 1 \oplus C_1(1^n) \oplus C_2(1^n)$ and there is no frozen variable in C'. ❑

For the next theorem, we need the following standard argument about how to build DP-complete sets from NP-complete sets.

Lemma 4.7. *Let Σ be a finite alphabet. If $A \subseteq \Sigma^*$ is* NP*-complete, then the set* $\text{dp}(A) \stackrel{\text{def}}{=} (A \times \Sigma^*) \triangle (\Sigma^* \times A)$ *is* DP*-complete.*

Proof. Since both, $(A \times \Sigma^*)$ and $(\Sigma^* \times A)$ are in NP, the set $\text{dp}(A)$ is in DP by the definition of DP. Let $B = C \triangle D$ be in DP where $C, D \in \text{NP}$. Since A is NP-complete, there are functions $f, g \in \text{FP}$ that reduce C to A and D to A, respectively. Let $h(x, y) \stackrel{\text{def}}{=} (f(x), g(y))$. Then $h \in \text{FP}$ and for all (x, y) holds $(x, y) \in B$ if and only if $h((x, y)) \in A$. So $\text{dp}(A)$ is DP-complete. ❑

Theorem 4.8. *Let $B \subseteq \mathrm{BF}$ be a set of Boolean functions.*

1. *If $B \subseteq \mathrm{L}$, $B \subseteq \mathrm{M}$, or $[B] = \mathrm{S}_{12}$ then $\exists\mathrm{FV}(B)$ is in* P,
2. *$\exists\mathrm{FV}(\mathrm{S}_1)$ is* NP*-complete,*
3. *if $\mathrm{S}_1 \subset [B]$ then $\exists\mathrm{FV}(B)$ is* DP*-complete,*
4. *and in all other cases $\exists\mathrm{FV}(B)$ is* coNP*-complete.*

Proof. 1. If $B \subseteq \mathrm{L}$ then in a B-circuit there is a frozen variable if and only if exactly one of the variables of its linear normal form has a coefficient of 1.
Let $B \subseteq \mathrm{M}$ and let $C(x_1, \ldots, x_n)$ be a B-circuit. The variable x_i is frozen if and only if C is satisfiable and $C(\beta_i) = 0$ for $\beta_i \stackrel{\text{def}}{=} 1^{i-1}01^{n-i}$: If $C(\beta_i) = 0$, then $C(\alpha) = 0$ for every tuple $\alpha \in \{0,1\}^{i-1} \times 0 \times \{0,1\}^{n-i}$, since $\alpha \leq \beta_i$ and C is monotonic. Hence x_i is frozen. On the other hand, if $C(\beta_i) = 1$, then x_i is not a frozen variable, because $C(1^n) = 1$. Moreover, the satisfiability of a monotonic C can be easily tested by computing $C(1^n)$.
If $B = \mathrm{S}_{12} = \mathrm{R}_1 \cap \mathrm{S}_1$ then every B-circuit C is satisfiable and has a frozen variable because C is 1-separating.

2. Note, that an S_1-circuit has a frozen variable if and only if it is satisfiable. Therefore, $\exists\mathrm{FV}(\mathrm{S}_1) \equiv \mathrm{CSAT}(\mathrm{S}_1)$ and the claim is proved with Theorem 4.1.

3. Let B be such that $\mathrm{S}_1 \subset [B]$. Since $\mathrm{CSAT}(\mathrm{S}_1)$ is NP-complete (Theorem 4.1) the set $\mathrm{dp}(\mathrm{CSAT}(\mathrm{S}_1))$ is DP complete, due to Lemma 4.7. We reduce $\mathrm{dp}(\mathrm{CSAT}(\mathrm{S}_1))$ to $\exists\mathrm{FV}(B)$. For $k \geq 2$, let h_k be the Boolean function such that $h_k(x_1, \ldots, x_k + 1) = 1$ if and only if $\Sigma_{i=1}^{k+1} x_i \geq n$. Since $\mathrm{S}_1 \subset B$, there is a $k \geq 2$ such that h_k is in $[B]$ (see Figures 3.2 and 3.1). For a pair of S_1-circuits C_1' and C_2', which are $m-$ and $n-$ary respectively, let C_1 (C_2, resp.) be the polynomial-time computable B-circuit, which is equivalent to C_1' (C_2', resp.). Now define $C \stackrel{\text{def}}{=} h_k(C_1, C_2, x_1, \ldots, x_{k-1})$, where each x_i is a variable neither occurring in C_1 nor in C_2. Clearly C can be encoded as a B-circuit of polynomial size with respect to $|C_1'| + |C_2'|$. Next we show that this transformation gives the needed reduction.
If $C_1', C_2' \in \mathrm{CSAT}(\mathrm{S}_1)$ then there are assignments $\alpha \in \{0,1\}^m$ and $\beta \in \{0,1\}^n$ such that $C_1(\alpha) = C_2(\beta) = 1$. Then none of the variables of C_1 is frozen, since $C(\gamma\beta 1^{k-1}) = 1$ for all $\gamma \in \{0,1\}^m$. The same argumentation holds for all variables in C_2. Furthermore for all $1 \leq i \leq k-1$ it holds that x_i is not frozen in C, since $C(\alpha\beta 1^{i-1}01^{k-i-1}) = C(\alpha\beta 1^{k-1}) = 1$.
If $C_1', C_2' \in \overline{\mathrm{CSAT}(\mathrm{S}_1)}$, then C is not satisfiable and therefore has no frozen variables. So let without loss of generality $C_1 \in \mathrm{CSAT}(\mathrm{S}_1)$ and $C_2 \in \overline{\mathrm{CSAT}(\mathrm{S}_1)}$. Then $C \equiv C_1 \wedge x_1 \wedge \cdots \wedge x_{k-1}$ and obviously at least all of the x_i's are frozen.

4. If $\mathrm{D}_1 \subseteq [B] \subseteq \mathrm{D}$ or $\mathrm{S}_{02} \subseteq [B] \subseteq \mathrm{R}_1$, then $\exists\mathrm{FV}(B)$ is coNP-complete because of Lemma 4.6. The only remaining case is $\mathrm{S}_{12} \subset [B] \subset \mathrm{R}_2$. Then B-circuits are trivially satisfiable and therefore $\exists\mathrm{FV}(B) \in$ coNP. The proof of the lower bound is similar to Case 3, but this time the reduction starts with $\overline{\mathrm{SAT}^*(\mathrm{S}_{12})}$, which is coNP-complete (see proof of Theorem 4.2). Since $\mathrm{S}_{12} \subset [B]$ there is a $k \geq 2$ such that h_k is in $[B]$. Let $C(x_1, \ldots x_n)$ be an S_{12}-circuit and let

$$D(x_1,\ldots,x_n,y_1,\ldots,y_k) \stackrel{\text{def}}{=} h_k(C(x_1,\ldots,x_n),y_1,\ldots,y_k) \wedge ((\bigwedge_{i=1}^{k} y_i) \vee \neg(\bigwedge_{j=1}^{n} x_n)),$$

where all the y_i are variables not already occurring in C. Observe that D can be encoded as B-circuit, because $\{\text{AND}, x \wedge (y \vee \overline{z})\} \subset \mathrm{S}_{12} \subseteq [B]$.
If $C \in \overline{\mathrm{SAT}^*(\mathrm{S}_{12})}$ then D is satisfiable only by setting y_i to 1 for all $1 \leq i \leq k$, hence all y_i are frozen.
On the other hand, if there is an $\alpha \in \{0,1\}^n$ such that $\alpha \neq (1,\ldots,1)$ and $C(\alpha) = 1$ then none of the x_i's is frozen ($1 \leq i \leq n$), since D can be satisfied by just setting all the y_i's to 1. Furthermore, for each $j \in \{1,\ldots,k\}$ holds $D(\alpha 1^{j-1}01^{k-j}) = D(\alpha 1^k) = 1$, hence none of the y_j's is frozen.

❑

Figure 4.2 gives an overview of Theorem 4.8. So, some interesting properties of Boolean circuits are natural to some Boolean clones: 1-separating functions have a frozen variable if and only if they are satisfiable which makes their complexity-classification especially easy. This is a good example, that it is worth our while to study the algebraic structure of objects in order to gain knowledge about problems that are based on them.

Since we require instances of $\mathrm{FV}(\mathrm{S}_1)$ to have a whole set of frozen variables, which may not even contain the one which is guaranteed to be frozen, the 1-separability cannot help us in this case.

Theorem 4.9. *Let $B \subseteq \mathrm{BF}$ be a set of Boolean functions.*

1. *If $B \subseteq \mathrm{M}$ or $B \subseteq \mathrm{L}$, then $\mathrm{FV}(B)$ is tractable,*
2. *else if $\mathrm{S}_1 \subseteq [B]$, then $\mathrm{FV}(B)$ is DP-complete,*
3. *else $\mathrm{FV}(B)$ is coNP-complete.*

Proof. If $B \subseteq \mathrm{M}$ or $B \subseteq \mathrm{L}$, then the argumentation from Theorem 4.8 holds. Furthermore, we have seen in Lemma 4.6 that if $\mathrm{D}_1 \subseteq [B] \subseteq \mathrm{D}$ or $\mathrm{S}_{02} \subseteq [B] \subseteq \mathrm{R}_1$, then $\mathrm{FV}(B)$ is coNP-complete.

Hence, the coNP-hardness of $\mathrm{FV}(B)$ for B such that $\mathrm{S}_{12} \subseteq [B]$ and the DP-hardness of $\mathrm{FV}(B)$ for all B such that $\mathrm{S}_1 \subseteq [B]$ remains to be proven. We reduce $\mathrm{FV}(\mathrm{R}_1)$ to $\mathrm{FV}(\mathrm{S}_{12})$ ($\mathrm{FV}(\mathrm{BF})$, which is DP-complete [JK04], to $\mathrm{FV}(\mathrm{S}_1)$, resp.). For that, let C be a circuit over the R_1-complete basis $\{x \wedge (y \vee \overline{z}), 1\}$ (over the complete basis $\{x \wedge \overline{y}, 1\}$, resp.) and let V be the set of variables used in C. Build a circuit C' by taking a variable x that is not contained in C and replace every occurrence of 1 in C by x. Then $C' \wedge x$ is an S_{12}-circuit (an S_1-circuit, resp.), which can only be satisfied by assignments that set x to 1. For all these assignments, $C' \wedge x$ is satisfied if and only if C is satisfied. Therefore $(C,V) \in \mathrm{FV}(\mathrm{R}_1)$ ($(C,V) \in \mathrm{FV}(\mathrm{BF})$, resp.) if and only if $(C' \wedge x, V) \in \mathrm{FV}(\mathrm{S}_{12})$ ($(C' \wedge x, V) \in \mathrm{FV}(\mathrm{S}_1)$, resp.). ❑

Figure 4.3 gives an overview for Theorem 4.9. After what we have seen so far, the following result is the expected one, since $\mathrm{AUDIT}(B)$ is an easier variant of $\exists\,\mathrm{FV}(B)$ where we leave away the NP-complete question of whether a circuit is satisfiable.

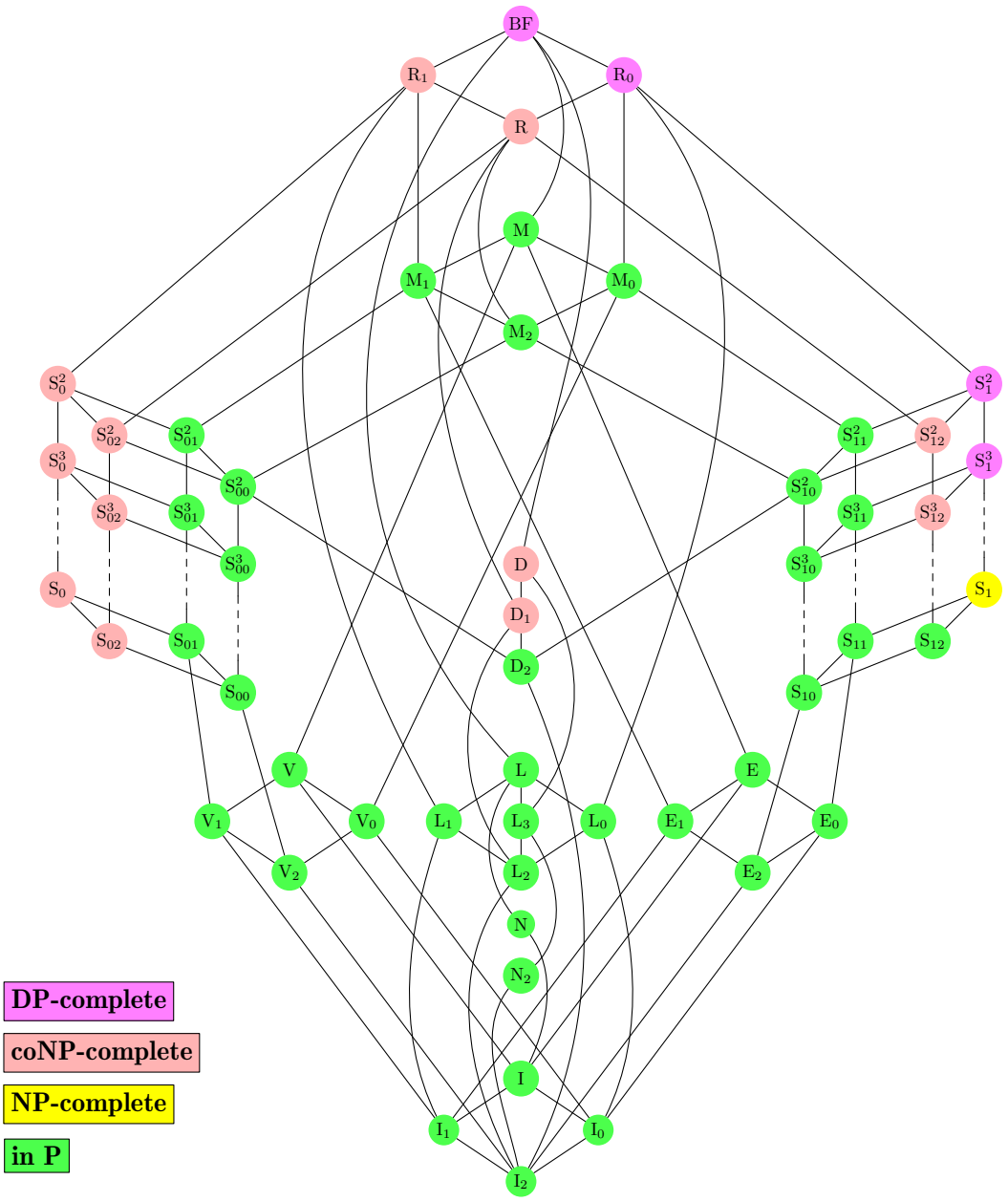


Fig. 4.2: The Complexity of $\exists\,\mathrm{FV}(B)$.

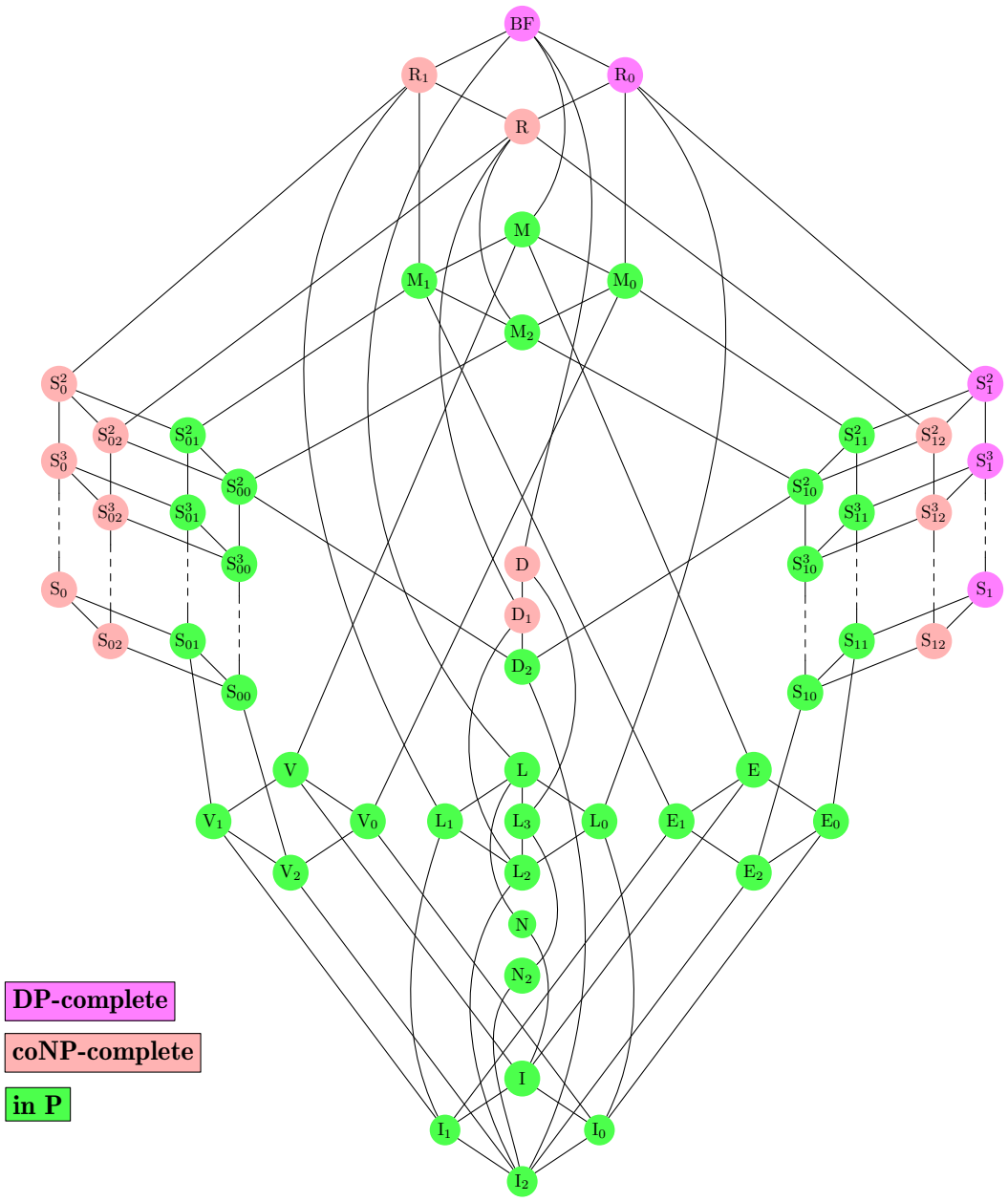


Fig. 4.3: The Complexity of FV(B).

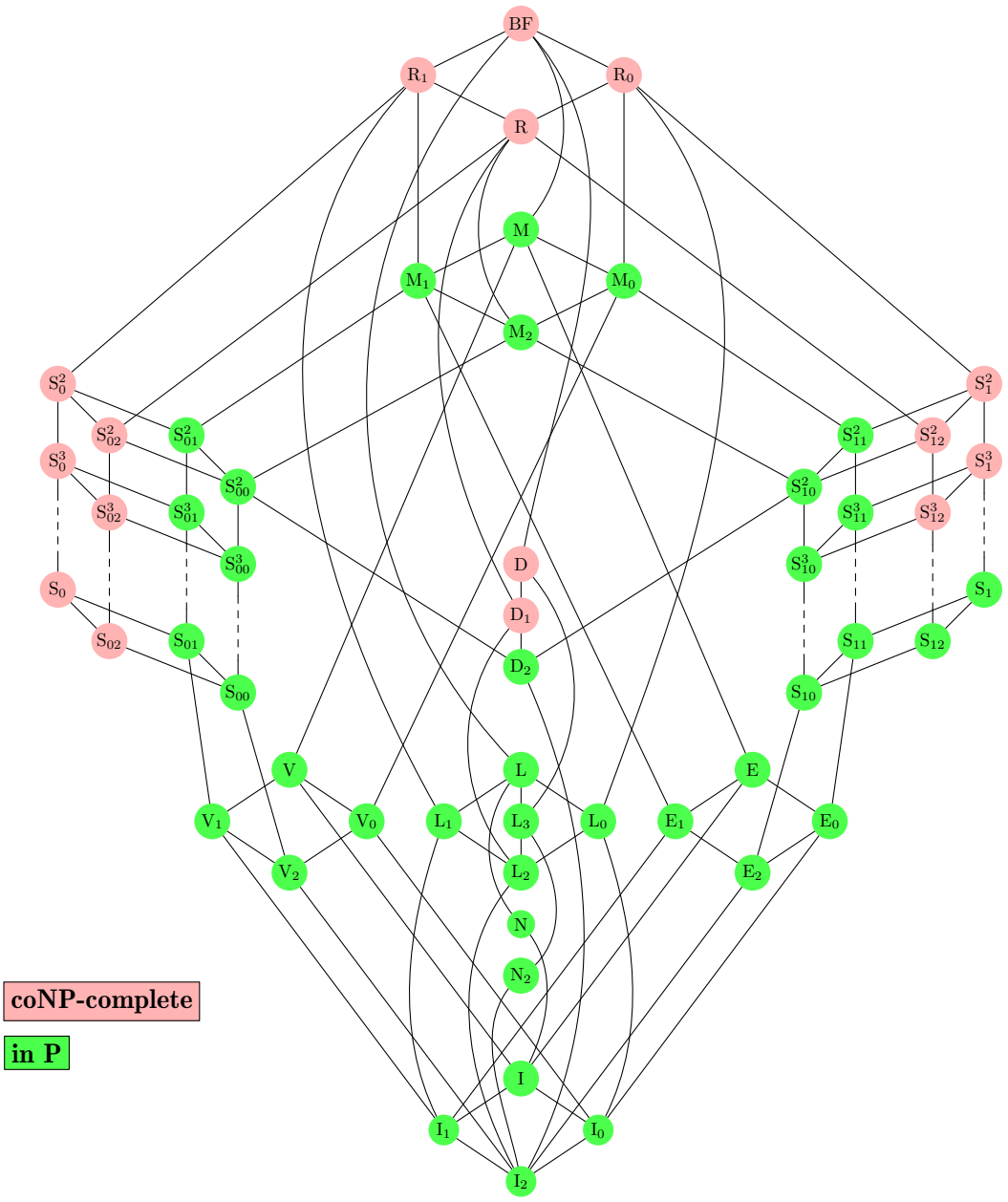


Fig. 4.4: The Complexity of AUDIT(B).

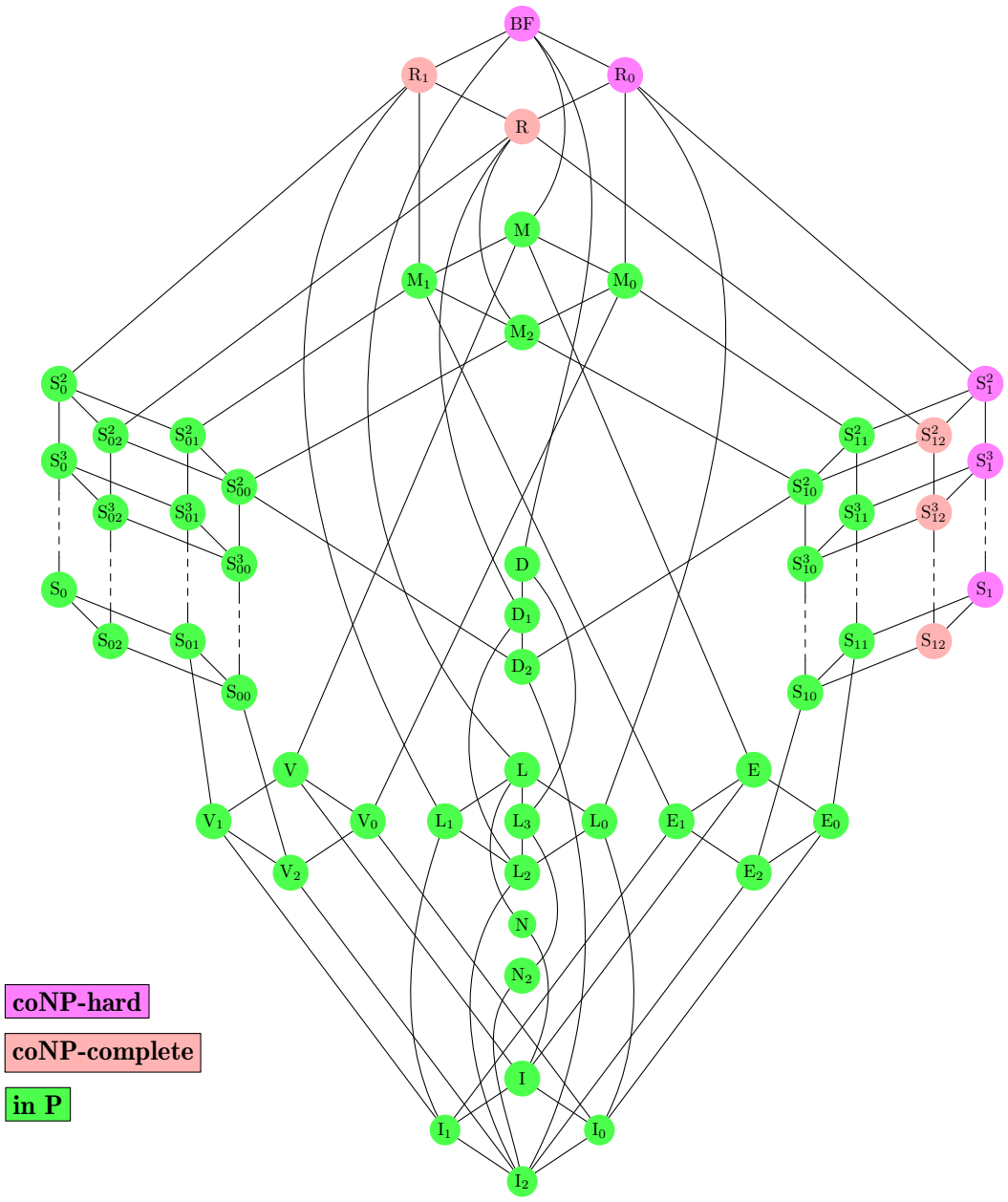


Fig. 4.5: The Complexity of UNIQUE-SAT(B).

Theorem 4.10. *Let $B \subseteq \mathrm{BF}$ be a set of Boolean functions.*

1. *If $B \subseteq \mathrm{M}$ or $B \subseteq \mathrm{L}$ or $B \subseteq \mathrm{S}_1$, then* $\mathrm{AUDIT}(B)$ *is tractable.*
2. *In all other cases,* $\mathrm{AUDIT}(B)$ *is* coNP*-complete.*

Proof. First, observe that if $\mathrm{CSAT}(B) \in \mathrm{P}$, then $\mathrm{AUDIT}(B) \equiv^{\mathrm{p}}_{\mathrm{m}} \exists\mathrm{FV}(B)$.

1. Since $\mathrm{CSAT}(B) \in \mathrm{P}$ if $[B] \subseteq \mathrm{L}$ or $[B] \subseteq \mathrm{M}$ (see Theorem 4.1), the claim for such B follows from the above observation and Theorem 4.8. If $B \subseteq \mathrm{S}_1$, it is either not satisfiable or has always a frozen variable.
2. If $B \subseteq \mathrm{R}_1$ or $B \subseteq \mathrm{D}$, every B-circuit is trivially satisfiable. Hence, we can use the above observation and Theorem 4.8, again. It remains to show that $\mathrm{AUDIT}(B)$ is coNP-hard if $\mathrm{S}_1 \subset [B]$. We will reduce $\overline{\mathrm{CSAT}(\mathrm{S}_1)}$, which is coNP-hard (see Theorem 4.1), to $\mathrm{AUDIT}(B)$. The proof runs along the same lines as in Theorem 4.8. Take an S_1-circuit C' and let C be a B-circuit with is equivalent to C'. Since $\mathrm{S}_1 \subset [B]$ there is an $h_k \in [B]$ for some $k \geq 2$. Define $C'' \stackrel{\mathrm{def}}{=} h_k(C, x_1, \ldots, x_k)$, where x_i is a variable not occurring in C $(1 \leq i \leq k)$. If C' is not satisfiable, x_i is frozen $(1 \leq i \leq k)$. If C' is satisfiable by an assignment α none of the variables in C' is frozen, since C' can be satisfied by setting all the x_i's to 1. Furthermore x_i is not frozen for $1 \leq i \leq k$, because $C'(\alpha 1^{i-1}01^{k-i}) = C'(\alpha 1^k) = 1$.

❑

Figure 4.4 gives an overview for Theorem 4.10. Hence, we still have to classify the UNIQUE-SAT(B)-problem.

Theorem 4.11. *Let $B \subseteq \mathrm{BF}$ be a set of Boolean functions.*

1. *If $\mathrm{S}_1 \subseteq [B]$, then* UNIQUE-SAT$(B) \equiv^{\mathrm{p}}_{\mathrm{m}}$ UNIQUE-SAT(BF)
2. *else if $\mathrm{S}_{12} \subseteq [B] \subseteq \mathrm{R}_1$, then* UNIQUE-SAT$(B)$ *is* coNP*-complete*
3. *else* UNIQUE-SAT(B) *is tractable.*

Proof. 1. Trivially UNIQUE-SAT$(B) \leq^{\mathrm{p}}_{\mathrm{m}}$ UNIQUE-SAT(BF) holds. Now let C be a Boolean circuit over the complete base $\{x \wedge \overline{y}, 1\}$, let x be a variable not occurring in C, let C'' be an S_1-circuit that is derived from C by replacing every occurrence of 1 in C by x, and let $C' \stackrel{\mathrm{def}}{=} C'' \wedge x$. Observe, that C' is an S_1-circuit with the same number of satisfying assignments as C.

2. If $B \subseteq \mathrm{R}_2$, we have UNIQUE-SAT$(B) \leq^{\mathrm{p}}_{\mathrm{m}} \overline{\mathrm{SAT}^*(B)} \leq^{\mathrm{p}}_{\mathrm{m}} \overline{\mathrm{SelectSAT}(B)}$. So, since for all $B \subseteq \mathrm{R}_1$ holds UNIQUE-SAT$(B) \in$ coNP, the coNP-completeness for all B with $\mathrm{S}_{12} \subseteq [B] \subseteq \mathrm{R}_1$ follows by Theorem 4.2.
3. For all $B \subseteq \mathrm{S}^2_0$ or $B \subseteq \mathrm{D}$ the claim holds because every n-ary function from B has at least 2^{n-1} satisfying assignments. If $B \subseteq \mathrm{M}$, then an n-ary B-circuit has more than one satisfying assignment if and only if there is an $i \in \{1, \ldots, n\}$ such that $C(1^{i-1}01^{n-i}) = 1$. If $B \subseteq \mathrm{L}$, then the number of satisfying assignments for every B-circuit can easily be determined using its linear normal form.

❑

Figure 4.5 gives an overview for Theorem 4.11

4.3 Quantified Boolean Constraint Satisfaction with Bounded Alternations

In the last sections we classified of the complexity of different problems over Boolean circuits with respect to the Boolean functions that occur in their gates. We made heavy use of the Boolean clones and were thus able to give short and simple proofs. In this section we have a look at a problem from the world of Boolean constraints. Recently, E. Hemaspaandra studied the complexity of satisfiability of certain constraint formulas with quantifiers where the number of alternations between the quantifiers is bounded by a constant [Hem04]. She gave a full classification for these problems with respect to the types of constraints that are allowed in the constraint formulas.

In this section, we reproof the results of Hemaspaandra. However, we will make use of the structure of Boolean co-clones which will simplify the proofs considerably.

Definition 4.12. *Let Γ be a set of Boolean constraints and let $F(x_1, \ldots, x_n) \in \mathrm{CF}(\Gamma)$. Let $Q_1, \ldots, Q_n$ be quantifiers from $\{\exists, \forall\}$. We call $G \stackrel{\mathrm{def}}{=} Q_1 x_1 Q_2 x_2 \ldots Q_n x_n F$ a* fully quantified Γ-formula. *Let k be the number of quantifier alternations in G. We call G a $\Sigma_k(\Gamma)$-*formula *if $Q_1 = \exists$ and a $\Pi_k(\Gamma)$-*formula *otherwise.*

We call G a $\Xi_k(\Gamma)$-formula if either k is odd and G is a $\Sigma_k(\Gamma)$-formula or k is even and G is a $\Pi_k(\Gamma)$-formula. With other words, G is a $\Xi_k(\Gamma)$ formula if it is a fully quantified Γ-formula with k alternations of quantifiers and the rightmost quantifier is $\exists$.

Definition 4.13. *For every $k \in \mathbb{N}$, and every finite set of Boolean relations Γ, we define the following problem.*

Problem:	*Quantified Boolean Constraint Satisfaction Problem,* $\mathrm{QSAT}_k(\Gamma)$
Input:	*A fully quantified $\Xi_k(\Gamma)$-formula φ*
Question:	*Is φ true?*

Why are we interested in $\Xi_k(\Gamma)$ formulas, of all things? Since in $\Xi_k(\Gamma)$-formulas the rightmost quantifier is always an existential one, a $\Xi_k(\Gamma)$-formula is in fact a quantified $\mathrm{EQCF}(\Gamma)$-formula. We have seen in Section 3.3 that there is a strong connection between this type of formulas and Boolean co-clones that we can use here. We remark, that the result of Hemaspaandra is over the same type of constraint formulas (although her definition differs). She also makes use of Schaefer's classification of Boolean constraints, which we have seen to be implied by Boolean co-clones. So, Hemaspaandra makes use of some universal algebra in her paper, without explicitly stating it. In the following, let $\Xi_k^{\mathrm{p}} \stackrel{\mathrm{def}}{=} \Sigma_k^{\mathrm{p}}$ if k is odd and $\Xi_k^{\mathrm{p}} \stackrel{\mathrm{def}}{=} \Pi_k^{\mathrm{p}}$, otherwise.

In order to utilize the co-clone structure of $\mathcal{L}(\mathrm{BR})$, we have to establish a link between the lattice and $\mathrm{QSAT}_k(\Gamma)$.

Lemma 4.14. *Let Γ_1 and Γ_2 be sets of relations over $\{0,1\}$ such that Γ_1 is finite and $\langle\Gamma_1\rangle \subseteq \langle\Gamma_2\rangle$. Then for all $k \geq 1$ we have $\mathrm{QSAT}_k(\Gamma_1) \leq_{\mathrm{m}}^{\mathrm{p}} \mathrm{QSAT}_k(\Gamma_2)$.*

Proof. The idea of the reduction is to replace every occurrence of a relation from Γ_1 with an existential quantified constraint formula over Γ_2.

According to Proposition 3.12, we can express relations from Γ_1 using relations from Γ_2 plus existential quantifiers and the eq-relation. For every $\Xi_k(\Gamma_1)$-formula F an equivalent $\Xi_k(\Gamma_2 \cup \{\text{eq}\})$-formula F' can be computed in polynomial time:

Let $F = Q_1 x_1 \cdots Q_m x_m G$, where G is in $\text{CF}(\Gamma_1)$. Then there is a formula $\exists y_1 \cdots \exists y_\ell G' \in \text{EQCF}(\Gamma_2)$ where $\ell \in \mathbb{N}$ and $y_1, \ldots, y_\ell$ are new variables and that formula describes the same relation as G. We can compute this new formula in polynomial time, since we can replace every atomar relation in G by an existential quantified formula from $\text{EQCF}(\Gamma_2)$. Then $F' \stackrel{\text{def}}{=} Q_1 x_1 \cdots Q_m x_m \exists y_1 \cdots \exists y_\ell G'$ describes the same relation as F and is in $\Xi_k(\Gamma_2 \cup \{\text{eq}\})$.

Now, F' may contain eq-constraints, which we have to remove in the case that $\text{eq} \notin \Gamma_2$. To do this, we go through all maximal sets of variables $\{x_1, \ldots, x_n\}$ such that $\text{eq}(x_1, x_2), \text{eq}(x_2, x_3), \ldots, \text{eq}(x_{n-1}, x_n)$ are clauses of F'. Observe, that we can identify these sets in polynomial time: This is as difficult as finding the connected parts in an undirected graph. For each of these sets, we do the following.

Case 1. There exist $i, j \in \{1, \ldots, n\}$ such that x_i and x_j are both universally quantified. Then F' is true if and only if $\forall x_i \forall x_j \text{eq}(x_i, x_j)$ is true, i.e., F' is false.
Case 2. All variables x_i of the set are existentially quantified. In this case we replace all variables in $\{x_1, \ldots, x_n\}$ with x_1 and delete the corresponding equality constraints and the quantifiers for $x_2, \ldots, x_n$. The resulting formula is equivalent to F'.
Case 3. There is an $i \in \{1, \ldots, n\}$ such that x_i is universally quantified and x_j is existentially quantified for all $j \in \{1, \ldots, n\} - \{i\}$. If there is such a j such that x_j is quantified before x_i, then F' can be true if and only if there is an element in $\{0, 1\}$ that is equal to all elements in $\{0, 1\}$, which is impossible. So let x_i be quantified before the other variables of the set. Then we can just replace the variables from $\{x_1, \ldots, x_n\} - \{x_i\}$ with x_i, delete the equality constraints and the quantifiers belonging to them, and get an equivalent formula.

This can be done in polynomial time. ❑

Theorem 4.15. $\text{QSAT}_k(\{\text{NAE}^3\})$ *is* $\leq^{\text{p}}_{\text{m}}$*-complete for* Ξ^{p}_k *if* $k \geq 1$.

Proof. It is obvious, that $\text{QSAT}_k(\Gamma)$ is in Ξ_k for all sets of relations Γ and all $k \geq 0$, since the constraint formulas can be viewed as normal formulas. Due to Lemma 4.14 and since $\langle\{\text{1-IN-3}\}\rangle = \text{BR}$, the problem $\text{QSAT}_k(\{\text{1-IN-3}\})$ is as hard as the general satisfiability problem for Ξ_k-formulas, which is known to be complete for Ξ^{p}_k [Wra77]. We now show $\text{QSAT}_k(\{\text{1-IN-3}\}) \leq^{\text{p}}_{\text{m}} \text{QSAT}_k(\{\text{NAE}^3\})$.

Let $\varphi \stackrel{\text{def}}{=} Q_1 X_1 \cdots \forall X_{k-1} \exists X_k F$ be a $\Xi_k(\text{1-IN-3})$-formula, where

$$F \stackrel{\text{def}}{=} \bigwedge_{j=1}^{n} \text{1-IN-3}(x_{j_1}, x_{j_2}, x_{j_3})$$

and for $1 \leq i \leq k$ is $Q_i X_i$ an abbreviation of $Q_i x_{i1} \dots Q_i x_{im_i}$ if $X_i = \{x_{i1}, \dots, x_{im_i}\}$ is a subset of the variables in F and $Q_i \in \{\exists, \forall\}$. For each constraint 1-IN-3$(x_{j_1}, x_{j_2}, x_{j_3})$ of F, introduce a new constraint

$$\begin{aligned} C_j &\stackrel{\text{def}}{=} \mathrm{NAE}^3(x_{j_1}, x_{j_2}, t) \wedge \mathrm{NAE}^3(x_{j_1}, x_{j_3}, t) \wedge \mathrm{NAE}^3(x_{j_2}, x_{j_3}, t) \wedge \\ &\quad \mathrm{NAE}^3(x_{j_1}, x_{j_2}, x_{j_3}) \\ &\equiv \text{2-IN-4}((x_{j_1}, x_{j_2}, x_{j_3}, t), \end{aligned}$$

where t is a new variable. We define

$$\begin{aligned} F' &\stackrel{\text{def}}{=} C_1 \wedge \dots \wedge C_n, \\ \varphi' &\stackrel{\text{def}}{=} Q_1(X_1 \cup \{t\})\, Q_2 X_2 \cdots \forall X_{k-1} \exists X_k F', \\ \psi &\stackrel{\text{def}}{=} Q_1 X_1 \cdots \forall X_{k-1} \exists X_k F'(t = 0), \text{ and} \\ \psi' &\stackrel{\text{def}}{=} Q_1 X_1 \cdots \forall X_{k-1} \exists X_k F'(t = 1). \end{aligned}$$

Here, $F'(t = a)$ means the constraint formula that is derived from F' by replacing every occurrence of t in F' with the constant $a \in \{0, 1\}$. Observe, that $\psi' \equiv \varphi$, since 2-IN-4$((x, y, z, 1) =$ 1-IN-3(x, y, z).

Claim 4.16. ψ is true if and only if ψ' is true.

Proof: Let ψ be true. We use that F' describes a complementive relation to show that ψ' is true, too. For that, we need the notion of a quantifier function. Let $\xi = Q_1 X_1 \cdots Q_\ell X_\ell G$ be a fully quantified Γ-formula where $G \in \mathrm{CF}(\Gamma)$ and Γ is an arbitrary set of constraints. Let

$$A \stackrel{\text{def}}{=} \{I : I \text{ is a function from } V \text{ to } \{0, 1\} \text{ for a } V \subseteq \{x_1, x_2, \dots\}\}$$

be the set of all functions that assign Boolean values to finite sets of variables. Let

$$I_\forall^\xi \stackrel{\text{def}}{=} \{I : I \text{ maps all universally quantified variables from } \xi \text{ to } \{0, 1\}\} \subsetneq A$$

be the set of Boolean assignments for the universally quantified variables from ξ.

A function $f : A \to A$ is called *quantifier function for* ξ if and only if the following holds:

1. For all $I \in I_\forall^\xi$ the function $f(I)$ is an assignment for all existentially quantified variables of ξ and
2. if $I_1, I_2 \in I_\forall^\xi$ and y is an existentially quantified variable in ξ and $\{x_1, \dots, x_m\}$ is the set of universally quantified variables of ξ that appear left of y, then $f(I_1)(y) = f(y)(x_i)$. That means that for an existentially quantified variable y, the value of $f(I)(y)$ is determined by $\{I(x_j) : x_j$ is universally quantified and appears left of $y\}$.

The function f is called *satisfying quantifier function for* ξ if and only if for all $I \in I_\forall^\xi$ the assignment $I \cup f(I)$ is satisfying for G. Observe, that ξ is true if and only if there is a satisfying quantifier function for ξ.

For $I \in A$, we define the function $\bar{I} \in A$ as follows:

$$\overline{I}(x_i) \stackrel{\text{def}}{=} \begin{cases} 1 - I(x_i) & \text{, if } I(x_i) \text{ is defined} \\ \text{not defined} & \text{otherwise.} \end{cases}$$

Now let ψ be true. Then there is a satisfying quantifier function f for ψ. For all $I \in I_\forall^\psi$, let $f'(\overline{I}) \stackrel{\text{def}}{=} \overline{f(I)}$. Observe, that f' is a quantifier function for ψ', since ψ and ψ' share the same quantifier prefix. Then for all $I \in I_\forall^\psi = I_\forall^{\psi'}$ holds $\overline{I} \cup f'(\overline{I}) = \overline{I} \cup \overline{f(I)}$ satisfies $F'(t=0)$ since $I \cup f(I)$ satisfies $F'(t=1)$ and F' is complementive. The other direction is analogous. □

Since φ' is equivalent to $\psi \wedge \psi'$ if $Q_1 = \forall$ and φ' is equivalent to $\psi \vee \psi'$ if $Q_1 = \exists$, we can conclude

$$\varphi' \text{ is true } \Leftrightarrow \psi' \text{ is true } \Leftrightarrow \varphi \text{ is true,}$$

since 1-IN-3$(x,y,z) \equiv$ 2-IN-4$(x,y,z,1)$. ❑

Theorem 4.17. *Let Γ be a set of Boolean constraints. Then for all $k \geq 1$ the problem* $\mathrm{QSAT}_k(\Gamma)$ *is $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete for Ξ_k^{p} if* $\mathrm{IN} \subseteq \langle\Gamma\rangle$ *and in* P *otherwise.*

Proof. If $\mathrm{IN} \not\subseteq \langle\Gamma\rangle$ then $\langle\Gamma\rangle \subseteq B$ for a $B \in \{\mathrm{IE}_2, \mathrm{IV}_2, \mathrm{IL}_2, \mathrm{ID}_2\}$. Therefore B is Horn, anti-Horn, affine, or bijunctive. For all these cases the quantified constraint satisfaction problem with no bounds on the quantifier alternations is known to be in P [Sch78, Dal97, CKS01]. So the remaining cases are those where $\mathrm{IN} \subseteq \langle\Gamma\rangle$. For $k = 0$, the problem $\mathrm{QSAT}_k(\Gamma)$ coincides with $\mathrm{CSP}(\Gamma)$. We have seen in Section 3.3 that Schaefer's theorem (Theorem 3.13) already settles this case, so let $k \geq 1$. We reduce $\mathrm{QSAT}_k(\{\mathrm{NAE}^3\})$ to $\mathrm{QSAT}_k(\{\mathrm{DUP}\})$ which finishes the proof, since $\langle\{\mathrm{NAE}^3\}\rangle = \mathrm{IN}_2$ and $\langle\{\mathrm{DUP}\}\rangle = \mathrm{IN}$ and because of Theorem 4.15 and Lemma 4.14.

Let $\varphi = Q_1 X_1 \cdots \forall X_{k-1} \exists X_k F$ be a $\Xi_k(\{\mathrm{NAE}^3\})$-formula where

$$F = \bigwedge_{i=1}^{m} \mathrm{NAE}^3(x_{i1}, x_{i2}, x_{i3})$$

is a $\{\mathrm{NAE}^3\}$-constraint formula. For $1 \leq i \leq m$, let

$$C_i(x_{i1}, x_{i2}, x_{i3}, t) \stackrel{\text{def}}{=} \mathrm{DUP}(x_{i1}, t, x_{i2}) \vee \mathrm{DUP}(x_{i1}, t, x_{i3}) \vee \mathrm{DUP}(x_{i2}, t, x_{i3}),$$

where t is a new variable, and let

$$\varphi' \stackrel{\text{def}}{=} Q_1 X_1 \cdots \forall X_{k-1} \cup \{t\} \exists X_k \bigwedge_{i=1}^{m} C_i(x_{i1}, x_{i2}, x_{i3}, t).$$

Observe, that φ is true if and only if φ' is true, since $\mathrm{NAE}^3(x_{i1}, x_{i2}, x_{i3})$ is equivalent to $\forall t C_i(x_{i1}, x_{i2}, x_{i3}, t)$. ❑

5. Clones in Structural Complexity Theory

We have seen that the structures of universal algebra can be very helpful when proving complexity results on problems over the Boolean domain. In computer science, there are other sets that are of interest which are not always at first sight recognized as clones. Examples are the set of recursive functions, the set of functions computable in polynomial time, or the set of functions computable in logarithmic space. They are all clones in the lattice of functions on $\mathbb{N}$. Since the exact relationship between many important complexity classes is still unknown, every means that could shed light on these relationships should be studied. Therefore, we examine some properties of complexity classes that are in fact clones, in this chapter. The focus of our attention will be on the clone FP – the set of functions that are computable deterministically in polynomial time.

5.1 Identifying Complexity Clones

In this section, we will see that there are a lot of function-complexity classes, that are in fact clones. For all complexity classes we examine it is rather obvious that they are closed under $\{\mathrm{PI}, \mathrm{TF}, \mathrm{LVF}, \mathrm{RF}\}$. Therefore we have to check whether they are closed under substitution. Furthermore, since arguments can be easily paired, and paired arguments can easily be separated again in all complexity classes we are concerned with, it suffices to concentrate on unary functions.

Theorem 5.1. *The following function-complexity classes are clones:* FL*,* FP*,* FPSPACE*,* FNC^i *for all* $i \geq 1$*, and* FNC*.*

Proof. It is well known, that L is closed under log-space-reductions and P is closed under polynomial-time reductions. The same argumentation used in the proofs of these closures can be used to show that FL and FP are closed under substitution. For FPSPACE, note that we require for every $f \in \mathrm{FPSPACE}$ that $|f(x)| \leq |x|^k$ for some $k \in \mathbb{N}$ and all $x \in \Sigma^*$.

Let $i \geq 1$ and $f, g \in \mathrm{FNC}^i$. Since $g \in \mathrm{FNC}^i$ there is a $k_g \in \mathbb{N}$ and a polynomial p such that for all $n \in \mathbb{N}$ there is a circuit C_g^n such that for all $x \in \Sigma^n$ holds $C_g^n(x) = g(x)$ and $\mathrm{size}(C_g^n) \leq p(n)$ and $\mathrm{depth}(C_g^n) \leq k_g(\log n)^i$. Due to the size of C_g^n for all $x \in \Sigma^*$ holds $|g(x)| \leq p(n)$. Since $f \in \mathrm{FNC}^i$ there exists a polynomial q and a $k_f \in \mathbb{N}$ such that for all $n \in \mathbb{N}$ there is a circuit $C_f^{p(n)}$ such that for all $x \in \Sigma^{p(n)}$ holds $f(x) = C_f^{p(n)}(x)$, $\mathrm{size}(C_f^{p(n)}) \leq q(p(n))$ and $\mathrm{depth}(C_f^{p(n)}) \leq k_f(\log p(n))^i \leq_{\mathrm{ae}} c(\log n)^i$ for a suitable constant

$c \in \mathbb{N}$. So if we build a new circuit by sequentializing C_g^n and $C_f^{p(n)}$, i.e., we build a new circuit by connecting the outputs of C_f^n with the inputs of C_g^n, we get a circuit of polynomial size and depth $\mathrm{O}((\log n)^i)$ that calculates $g \circ f$. It is easy to see, that if f and g can be realized by $\mathrm{U_L}$-uniform families of circuits, so can be $g \circ f$.

❑

5.2 An Example of a Hierarchy of Complexity Clones Below FP

We are especially interested in clones in $\mathcal{L}(\mathrm{FP})$ that are time-complexity classes. In the following, we will see that there are countably many such complexity classes that are clones below FP. We know, that $\mathrm{FDTIME}(n^{\mathrm{O}(1)}) = \mathrm{FP}$ is a clone and it is easy to see, that $\mathrm{FDTIME}(n(\log n)^{O(1)})$ is a clone, too. Are there time-complexity classes, that are clones, between these two classes? We define an in-between hierarchy of time-complexity classes containing countably many different clones. First, we make the following observation about time-complexity classes and them being closed under substitution.

Proposition 5.2. *Let $\mathcal{K} \subseteq \mathcal{F}^1(\mathbb{N})$ be a class of functions on $\mathbb{N}$. Then* $\mathrm{FDTIME}(\mathcal{K})$ *is closed under substitution if for all $t_1, t_2 \in \mathcal{K}$ there is a $t_3 \in \mathcal{K}$ such that $t_2 + t_1 \circ t_2 \leq_{\mathrm{ae}} t_3$.*

Proof. Let $f, g \in \mathrm{FDTIME}(\mathcal{K})$ such that f can be computed deterministically in time t_1 and g can be computed deterministically in time t_2. Then for all $x \in \Sigma^*$ holds $|g(x)| \leq t_2(|x|)$. Therefore $f \circ g$ can be computed in time $t_2(|x|) + t_1(t_2(|x|)) \leq_{\mathrm{ae}} t_3(|x|) \in \mathcal{K}$. Hence $f \circ g \in \mathrm{FDTIME}(\mathcal{K})$. ❑

Definition 5.3. *Let* $\lg(x) \stackrel{\mathrm{def}}{=} \log x$ *if* $x > 0$ *and* $\lg(0) \stackrel{\mathrm{def}}{=} 0$. *Let for* $i \geq 1$ *and* $k \geq 0$,

$$\begin{aligned}
\mathrm{ld}(0, n) &\stackrel{\mathrm{def}}{=} n, \\
\mathrm{ld}(i, n) &\stackrel{\mathrm{def}}{=} \lg \mathrm{ld}(i-1, n), \\
\mathrm{tp}_k(a_1, a_2, \ldots, a_i) &\stackrel{\mathrm{def}}{=} k a_1^{k a_2^{\cdots^{k a_i}}}, \\
\mathrm{lt}_i^k(n) &\stackrel{\mathrm{def}}{=} \mathrm{tp}_k(\mathrm{ld}(1, n), \mathrm{ld}(2, n), \ldots, \mathrm{ld}(i, n), 1), \text{ and} \\
\mathrm{ltn}_i^k(n) &\stackrel{\mathrm{def}}{=} n \cdot \mathrm{lt}_i^k(n).
\end{aligned}$$

Here, ld stands for nested **l**ogarithmus **d**ualis, tp stands for **t**ower of **p**owers, lt stands for **l**ogarithmic **t**ower, and ltn stands for **l**ogarithmic **t**ower multiplied with **n**. Note that the order of the exponentiations in $\mathrm{tp}_k(a_1, \ldots, a_i)$ is from upper to lower, i.e., first you compute $a_{i-1}^{a_i}$, which is the exponent for a_{i-2} and so on. In particular,

$$\mathrm{tp}_k(a_1, a_2, \ldots, a_i) \neq k((a_1^{k a_2})^{\cdots})^{k a_i}.$$

We write tp for tp_1. Note, that we can replace all occurrences of lg in the definition of $\mathrm{lt}_i^k(n)$, by log, if $n \geq \mathrm{tp}(\underbrace{2, \ldots, 2}_{i})$. We used n in the above definitions instead of the usual

x or y, since we want to emphasise that these functions are meant to be upper bounds for computing-times and the normal variable we use for input-lengths is n.
We are interested in the classes $\mathrm{FDTIME}(\mathrm{ltn}_i^{\mathrm{O}(1)}(n))$ for all $i \geq 1$. In the next two lemmas and the following corollary, we show that all these classes are closed under substitution and are therefore clones.

Lemma 5.4. *For all $\varepsilon > 0$, and all $c, i \geq 1$ holds*

$$\mathrm{lt}_i^c(x) \leq_{\mathrm{ae}} x^\varepsilon.$$

Proof. If $x \geq 1$ then

$$\begin{aligned}
\mathrm{lt}_1^c(x) &= \mathrm{tp}_c(\mathrm{ld}(1,x),1) \\
&= c \cdot \mathrm{ld}(1,x)^{c\cdot 1} \\
&= c(\log x)^c \\
&= c2^{c \log\log x} \\
&\leq_{\mathrm{ae}} 2^{\varepsilon \log x} \\
&= x^\varepsilon
\end{aligned}$$

for all $c \geq 1$ and $\varepsilon > 0$. Now assume that for all $j \leq i$, for all $\delta > 0$, and for all $c \geq 1$ holds $\mathrm{lt}_j^c(x) \leq_{\mathrm{ae}} x^\delta$. Then for all $\varepsilon > 0$, all $c \geq 1$ and all $x \geq \mathrm{tp}(\underbrace{2,\ldots,2}_{i})$ we have

$$\begin{aligned}
\mathrm{lt}_{i+1}^c(x) &= \mathrm{tp}_c(\lg x, \mathrm{ld}(2,x), \ldots, \mathrm{ld}(i+1,x), 1) \\
&= \mathrm{tp}_c(y, \mathrm{ld}(2,2^y), \ldots, \mathrm{ld}(i+1,2^y), 1) && \text{, where } y = \log x \\
&= c \cdot \mathrm{tp}(y, \mathrm{lt}_i^c(y)) \\
&\leq_{\mathrm{ae}} c \cdot \mathrm{tp}(y, y^\delta) && \text{, for a } \delta < \frac{1}{2} \\
&= c \cdot \mathrm{tp}(\log x, (\log x)^\delta) \\
&= c \cdot 2^{\log\log x \cdot (\log x)^\delta} \\
&\leq_{\mathrm{ae}} 2^{(\log x)^{2\delta}} \\
&\leq_{\mathrm{ae}} 2^{\varepsilon \log x} \\
&= x^\varepsilon
\end{aligned}$$

❑

Lemma 5.5. *For all $i \geq 1, k \geq 0$, holds $\mathrm{ltn}_i^k \circ \mathrm{ltn}_i^k \leq_{\mathrm{ae}} \mathrm{ltn}_i^c$ for a constant $c \geq 0$.*

Proof. First, using Lemma 5.4, observe that for all $n > 0$ and $m \geq 1$ and holds

$$\mathrm{ld}(m, \mathrm{ltn}_i^k(n)) = \mathrm{ld}(m, n \cdot \mathrm{lt}_i^k(n)) \leq_{\mathrm{ae}} \mathrm{ld}(m, n^2) \leq_{\mathrm{ae}} 2\mathrm{ld}(m,n)$$

Using this, we obtain:

$$\begin{aligned}
\mathrm{ltn}_i^k(\mathrm{ltn}_i^k(n)) &= \mathrm{ltn}_i^k(n) \cdot \mathrm{lt}_i^k(\mathrm{ltn}_i^k(n)) \\
&= \mathrm{ltn}_i^k(n) \cdot \mathrm{tp}_k(\lg(n \cdot \mathrm{lt}_i^k(n)), \mathrm{ld}(2, \mathrm{ltn}_i^k(n)), \ldots, \mathrm{ld}(i, \mathrm{ltn}_i^k(n)), 1) \\
&\leq_{\mathrm{ae}} \mathrm{ltn}_i^k(n) \cdot \mathrm{tp}_k(2 \lg n, 2\mathrm{ld}(2, n), \ldots, 2\mathrm{ld}(i, n), 1) \\
&\leq_{\mathrm{ae}} n \cdot \mathrm{lt}_i^k(n) \cdot \mathrm{tp}_k((\lg n)^2, (\mathrm{ld}(2, n))^2, \ldots, (\mathrm{ld}(i, n))^2, 1) \\
&\leq n \cdot k \cdot \mathrm{lt}_i^{2k}(n) \cdot \mathrm{lt}_i^{2k}(n) \\
&\leq \mathrm{ltn}_i^{4k}(n)
\end{aligned}$$

❑

With Proposition 5.2 and Lemma 5.5 we can conclude:

Corollary 5.6. $\mathrm{FDTIME}(\mathrm{ltn}_i^{\mathrm{O}(1)}(n))$ *is a clone for all* $i \geq 1$.

Next, we show that this hierarchy is strict. For $f, g \in \mathcal{F}^1(\mathbb{N})$, let $f \prec_{\mathrm{io}} g$ if and only if

$$\liminf_{n \to \infty} \frac{f(n)+1}{g(n)+1} < \infty.$$

We need this definition in the following hierarchy theorem:

Theorem 5.7 ([HS66]). *Let* $f, g \in \mathcal{F}^1(\mathbb{N})$ *such that* $\mathrm{id} \leq_{\mathrm{ae}} g$, $f \prec_{\mathrm{io}} g$, $f \leq_{\mathrm{ae}} g \cdot \log g$, *and let* g *be time-constructible in time* $g \log g$. *Then* $\mathrm{FDTIME}(f) \subsetneq \mathrm{FDTIME}(g \log g)$.

A function g is time-constructible in time t, if and only if there is a deterministic Turing machine that outputs $1^{g(n)}$ on an input of length n in at most $t(n)$ steps. Let us now verify, that the functions that define the levels of our hierarchy meet the preconditions of the hierarchy theorem.

Lemma 5.8. *Let* $1 \leq i < j$ *and* $k \geq 1$. *Then the following holds:*

1. $\mathrm{id} \leq_{\mathrm{ae}} \mathrm{ltn}_i^k$.
2. $\mathrm{ltn}_i^k \prec_{\mathrm{io}} \mathrm{ltn}_j^k$.
3. $\mathrm{ltn}_i^k \leq_{\mathrm{ae}} \mathrm{ltn}_j^k$.
4. ltn_i^k *is time-constructible in time* $\mathrm{ltn}_i^k \cdot \log \mathrm{ltn}_i^k$.

Proof. The first point is obvious. The second point holds true since we even have

$$\lim_{n \to \infty} \frac{\mathrm{ltn}_i^k(n)+1}{\mathrm{ltn}_j^k(n)+1} = 0.$$

The third point is immediate from the definition of ltn_i^k. This leaves the fourth point to prove.

We construct a machine, that works as follows on an input x of length n. First, the machine computes $\mathrm{ld}(\ell, n)$ for all $1 \leq \ell \leq i+1$. This can be done in time $\mathrm{O}(n \log n)$. The function $\exp(x, y) \stackrel{\mathrm{def}}{=} x^y$ can be computed in time $\mathrm{O}(|x|^{\mathrm{O}(|y|)})$ and

$$\begin{aligned} |\exp(x,y)| &< |(2^{|x|})^{2^{|y|}}| \\ &= |2^{|x|2^{|y|}}| \\ &= |x|2^{|y|}+1 \\ &= 2^{\log|x|+|y|}+1 \\ &\leq_{\text{ae}} 2^{|y|\cdot\log|x|} \\ &= |x|^{|y|}. \end{aligned}$$

Therefore, $\mathrm{lt}_i^k(n) = \mathrm{tp}_k(\mathrm{ld}(1,n),\mathrm{ld}(2,n),\ldots,\mathrm{ld}(i,n),1)$ can be computed in time

$$\begin{aligned} \sum_{j=1}^{i} \mathrm{tp}_c(\mathrm{ld}(j+1,n),\mathrm{ld}(j+2,n),\ldots,\mathrm{ld}(i+1,n),1) \quad &\leq \\ n\cdot\mathrm{tp}_c(\mathrm{ld}(2,n),\mathrm{ld}(3,n),\ldots,\mathrm{ld}(i+1,n),1) \quad &\leq_{\text{ae}} \quad \mathrm{ltn}_i^k(n) \end{aligned}$$

for some constant c. Finally, we have to output $\mathrm{ltn}_i^k(n)$ symbols. To do that, we need $\ell\cdot n\cdot(\log n+\mathrm{lt}_i^k(n))$ steps, where ℓ is some constant (First, we write $\mathrm{lt}_i^k(n)$ symbols on an extra tape, then we copy this string n times to the output tape). Hence, we need no more than

$$\begin{aligned} O(n\log n)+\mathrm{ltn}_i^k(n)+\ell n(\log n+\mathrm{lt}_i^k(n)) &= O(n\log n)+\mathrm{ltn}_i^k(n)+\ell(n\log n+\mathrm{ltn}_i^k) \\ &= O(\mathrm{ltn}_i^k(n)) \\ &\leq_{\text{ae}} \mathrm{ltn}_i^k(n)\cdot\log\mathrm{ltn}_i^k(n) \end{aligned}$$

steps to write a string of $\mathrm{ltn}_i^k(n)$ symbols to the output tape. ❑

So we can apply the hierarchy theorem, and therefore this hierarchy of time-complexity clones is strict. We summarize our results as follows.

Theorem 5.9. *1. For all $i\geq 1$ the class $\mathrm{FDTIME}(\mathrm{ltn}_i^{\mathrm{O}(1)})$ is a clone.*
2. If $1\leq i<j$ then $\mathrm{FDTIME}(\mathrm{ltn}_i^{\mathrm{O}(1)})\subsetneq\mathrm{FDTIME}(\mathrm{ltn}_j^{\mathrm{O}(1)})$
3. For all i holds $\mathrm{FDTIME}(\mathrm{ltn}_i^{\mathrm{O}(1)})\subsetneq\mathrm{FP}$.
4. $\mathrm{LT}\stackrel{\text{def}}{=}\bigcup_{i\geq 1}\mathrm{FDTIME}(\mathrm{ltn}_i^{\mathrm{O}(1)})\subsetneq\mathrm{FP}$ is a clone.

5.3 Finite Bases

In this section, we show that FL, FP, and FPSPACE have finite bases.

Definition 5.10. *Let A be a set of functions.*

– *A pair of functions $(f,h)\in A^2$ where f is binary and h is unary is called a* universal pair of functions for A *if for all unary $g\in A$, there are $i,k\in\mathbb{N}$ such that for all $x\in\mathbb{N}$ holds $g(x)=f(i,h^k(x))$.*

– *A is* closed under pairing *if and only if there is a binary function* $\text{pair} \in A$ *such that for all* $n \geq 2$ *and all* n*-ary functions* $g \in A$ *there is a unary* $f \in A$ *such that*

$$g(x_1, \ldots, x_n) = f(\text{pair}(x_1, \text{pair}(x_2, \ldots \text{pair}(x_{n-1}, x_n))))$$

for all $x_1, \ldots, x_n \in \mathbb{N}$*. We write* $\text{pair}^n(x_1, \ldots, x_n)$ *for* $\text{pair}(x_1, \text{pair}(x_2, \ldots \text{pair}(x_{n-1}, x_n)))$ *and call* pair *a* pairing function *for* A*.*

Lemma 5.11. *If* A *is a clone that contains* c_0 *and* succ*, that has a universal pair of functions* (univ, h)*, and is closed under pairing, then it has a finite base.*

Proof. We show that $B \stackrel{\text{def}}{=} \{c_0, \text{pair}, \text{univ}, \text{succ}, h\}$ is a base of A, where pair is a pairing function for A and (univ, h) is a universal pair of functions of A. Obviously, $[B] \subseteq A$, since all functions from B are in A, and A is a clone.

Now let $g \in A$ be an arbitrary n-ary function. Since A is closed under pairing, there is a function g' such that $g(x_1, \ldots, x_n) = g'(\text{pair}^n(x_1, \ldots, x_n))$ for all $x_1, \ldots, x_n \in \mathbb{N}$. Since (univ, h) is a universal pair of functions for A, there are $i, k \in \mathbb{N}$ such that $g'(x) = \text{univ}(i, h^k(x))$ for all $x \in \mathbb{N}$. For all $n \in \mathbb{N}$ is $c_n \in [B]$, since $c_n(x) = \underbrace{\text{succ}(\ldots \text{succ}(}_{n} c_0(x)) \ldots)$. Hence $g(x_1, \ldots, x_n) = \text{univ}(c_i(x_1), h^k(\text{pair}^n(x_1, \ldots, x_n)))$ for all $x_1, \ldots, x_n \in \mathbb{N}$, and therefore $g \in [B]$. ❑

Corollary 5.12. *The classes* FP*,* FPSPACE*, and* FL *have finite bases.*

Proof. Obviously, all three classes contain c_0 and succ and are closed under pairing. We show that FP has a universal pair of functions. For that, let $M_0, M_1, \ldots$ be an enumeration of Turing machines such that M_i can be computed in time $O(|i|^2)$ on input i and every step of M_i can be computed in time $O(|i|^2)$. Let

$$\begin{aligned} \text{pad}(x) &\stackrel{\text{def}}{=} 1^{|x|^2-|x|-1}0x \\ \text{univ}(i,x) &\stackrel{\text{def}}{=} \begin{cases} M_i \text{ on input } x & \text{, if } M_i \text{ halts after } |i||x|^2 + |i| \text{ steps} \\ \text{pad}(x) & \text{otherwise.} \end{cases} \end{aligned}$$

Then $\text{pad}(x)$ can be computed in time $O(|x|^2)$ and univ can be computed in time $O(|i|^3|x|^2)$ and hence univ and pad are in FP. Let g be a unary function from FP. Then there is an i such that M_i computes g in time $O(n^k)$ for some constant $k \in \mathbb{N}$. Let

$$f(x) \stackrel{\text{def}}{=} \begin{cases} g(x') & \text{, if } x = \text{pad}^{\lceil \log k \rceil}(x') \\ 0 & \text{otherwise.} \end{cases}$$

We can compute f in time $O(|x|^2)$ and therefore there is a j such that M_j computes f and halts always after less than $|j||x|^2+|j|$ steps on all inputs x. Therefore $f(x) = \text{univ}(c_j(x), x)$ and $g(x) = \text{univ}(c_j(x), \text{pad}^{\lceil \log k \rceil}(x))$.

We can use analogous proofs to show that FPSPACE and FL have a universal pair of functions. ❑

Together with Proposition 2.1 this leads us to the following corollary.

Corollary 5.13. *The lattices* $\mathcal{L}$(FP)*,* $\mathcal{L}$(FPSPACE)*, and* $\mathcal{L}$(FL) *are dual-atomic.*

We remark, that FNC is not likely to have a finite base unless the NC-hierarchy collapses, due to Proposition 2.1.

Proposition 5.14. *If* FNC *has a universal pair of functions then the* NC*-hierarchy collapses.*

So, since $\mathcal{L}$(FP) is dual-atomic, it would be nice to know the dual-atoms of this lattice. In the next section, we search for these.

5.4 Dual-atoms for $\mathcal{L}$(FP)

We first show, that there are uncountably many dual-atoms for all complexity clones that contain all constant functions and can do something like a case-distinction. For this, we make use of the fact that for every set $A \subseteq \mathbb{N}$, the set $\mathrm{Pol}(A)$ is a clone (see Proposition 3.3). Furthermore, we know that for two clones A and B, the intersection $A \cap B$ is a clone, too.

Proposition 5.15. *Let* $\emptyset \neq A \subsetneq \mathbb{N}$*. For* $k \in \mathbb{N}$*, let* $f_k : \mathbb{N}^3 \to \mathbb{N}$ *be such that*

$$f_k(x,y,z) \stackrel{\text{def}}{=} \begin{cases} y & \text{, if } x = k \\ z & \text{otherwise.} \end{cases}$$

If $\mathcal{K}$ *is a clone of functions that contains all constant functions on* $\mathbb{N}$ *and* f_k *for all* $k \in \mathbb{N}$*, then* $B \stackrel{\text{def}}{=} \mathcal{K} \cap \mathrm{Pol}(A)$ *is a predecessor of* $\mathcal{K}$ *in the lattice* $\mathcal{L}(\mathcal{K})$*, i.e.,* $B \prec \mathcal{K}$*.*

Proof. For every $a \in A$, the constant functions c_a with $c_a(x) = a$ for all $x \in \mathbb{N}$ is contained in both, $\mathcal{K}$ and $\mathrm{Pol}(A)$. Therefore the clone B is not empty. Since $A \neq \emptyset$, there a constant function $c_b \in \mathcal{K}$ such that $c_b \not\sim A$, hence $B \subsetneq \mathcal{K}$.

Let $f \in \mathcal{K} - B$ be an n-ary function. Then there are $a_1, \ldots, a_n \in A$ such that $z \stackrel{\text{def}}{=} f(a_1, \ldots, a_n) \notin A$. Obviously, $c_{a_i} \in \mathrm{Pol}(A)$ for $1 \leq i \leq n$. Let $g \in \mathcal{K}$ be an m-ary function and $a \in A$. We define

$$s_z(x_1, \ldots, x_m, y) \stackrel{\text{def}}{=} \begin{cases} g(x_1, \ldots, x_m) & \text{, if } y = z \\ a & \text{otherwise} \end{cases}$$

Observe, that $s_z \in \mathcal{K} \cap \mathrm{Pol}(A)$. Furthermore,

$$g(x_1, \ldots, x_m) = s_z(x_1, \ldots, x_m, f(c_{a_1}(x_1), \ldots, c_{a_n}(x_1))).$$

Hence, $g \in [B \cup \{f\}]$. ❑

Corollary 5.16. *If $\mathcal{K}$ is a clone that meets the prerequisites of Proposition 5.15, then $\mathcal{L}(\mathcal{K})$ has uncountably many dual-atoms.*

Proof. We have to show that if $A, B \subsetneq \mathbb{N}$ and $A \neq B$, then $\mathcal{K} \cap \mathrm{Pol}(A) \neq \mathcal{K} \cap \mathrm{Pol}(B)$. This is the case, since $\mathcal{K}$ contains all constant functions: Without loss of generality, there is an $a \in A$ with $a \notin B$. Then $c_a \in \mathcal{K} \cap \mathrm{Pol}(A)$ but $c_a \notin \mathcal{K} \cap \mathrm{Pol}(B)$. ❑

This shows that classes like FP do have uncountably many dual-atoms and that these classes can not be defined as the set of functions preserving a subset of $\mathbb{N}$. We can generalize this statement.

Proposition 5.17. *Let $\mathcal{K} \in \{\mathrm{FL}, \mathrm{FP}, \mathrm{FPSPACE}, \mathrm{FNC}\} \cup \bigcup_{i \geq 1} \mathrm{FNC}^i$. Then $\mathcal{K} \neq \mathrm{Pol}(A)$ for every $A \subseteq \mathbb{N}^n$ where $n \geq 1$.*

Proof. Since $\mathrm{Pol}(\mathbb{N}^n) = \mathrm{Pol}(\emptyset)$ is the set of all functions on $\mathbb{N}$ for all $n \geq 1$, $\mathcal{K}$ can be none of them. Let therefore $A \subsetneq \mathbb{N}^n$ and non-empty, and suppose $\mathcal{K} = \mathrm{Pol}(A)$.

Then there is a $\beta \in \mathbb{N}^n - A$. We can assume that for all $i, j \in \{1, \ldots, n\}$ there is an $\alpha \in A$ such that $\alpha[i] \neq \alpha[j]$: If there are i, j such that for all $\alpha \in A$ holds $\alpha[i] = \alpha[j]$ then $\mathcal{K} = \mathrm{Pol}(A')$ where A' is derived from A by projecting out the i-th coordinate. Let $\alpha_1, \ldots, \alpha_n \in A$ such that $\alpha_1[1] \neq \alpha_1[2], \alpha_2[2] \neq \alpha_2[3], \ldots, \alpha_n[n] \neq \alpha_n[1]$. Then for all $i, j \in \{1, \ldots, n\}$ we have $(\alpha_1[i], \ldots, \alpha_n[i]) \neq (\alpha_1[j], \ldots, \alpha_n[j])$ if $i \neq j$.

Let

$$f(x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} \begin{cases} \beta[i] & \text{, if } (x_1, \ldots, x_n) = (\alpha_1[i], \ldots, \alpha_n[i]) \\ 0 & \text{otherwise.} \end{cases}$$

Obviously, $f \in \mathcal{K}$ and $f \not\sim A$ since $f(\alpha_1, \ldots, \alpha_n) = \beta$. ❑

Corollary 5.18. *Let $\mathcal{K} \in \{\mathrm{FL}, \mathrm{FP}, \mathrm{FPSPACE}, \mathrm{FNC}\} \cup \bigcup_{i > 1} \mathrm{FNC}^i$. If $\Gamma \subseteq \mathcal{R}(\mathbb{N})$ is finite, then $\mathcal{K} \neq \mathrm{Pol}(\Gamma)$.*

Proof. Let $\Gamma = \{R_1, \ldots, R_n\}$. If $\mathcal{K} = \mathrm{Pol}(\Gamma)$ then $\mathcal{K} = \mathrm{Pol}(R_1 \times \cdots \times R_n)$ which is a contradiction to Proposition 5.17. ❑

We have found a lot of dual-atoms for FP. However, it is not difficult to see, that these dual-atoms have essentially the same computational power as FP: Let $A \subsetneq \mathbb{N}$ be non-empty, and let $a \notin A$. Then for every n-ary function $f \in \mathrm{FP}$ there is an $(n+1)$-ary function $g \in \mathrm{Pol}(A) \cap \mathrm{FP}$ such that $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n, a)$ for all $x_1, \ldots, x_n \in \mathbb{N}$.

We want to find dual-atoms for FP that are not only syntactically weaker than FP. We will examine two approaches to accomplish this: For one, it is known that every function from FP can be computed by uniform families of B-circuits with polynomial size where B is a complete set of Boolean functions. What is the power of such circuits if B is not complete? In the second approach, we will have a look at deterministic Turing machines that have a running-time that is not quite polynomial but slightly less.

5.4.1 Dual-Atoms of FP and Polynomial-Sized Circuits

Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be sets of functions on $\mathbb{N}$. Let

$$\mathrm{FCPS}_B \stackrel{\mathrm{def}}{=} \mathrm{FSIZE\text{-}DEPTH}_B(O(n^{O(1)}), O(n^{O(1)}))$$

be the class of **f**unctions computable by **c**ircuits of **p**olynomial **s**ize with gates from B.

For every set of Boolean functions B, for which $[B] = \mathrm{BF}$, we have $\mathrm{FP} = \mathrm{FCPS}_B$, (e.g. $B = \{\textsc{And}, \textsc{Or}, \textsc{Not}\}$) [Lad75b]. Note that the actual base B we choose is not relevant. Let $f \in \mathrm{FCPS}_B$ be calculated by a family of circuits $(C_i)_{i\in\mathbb{N}}$ over B. Every gate in the circuits computes Boolean function from B, and since $[B'] = [B]$, each of these Boolean functions can be computed by a B'-circuit: Hence we find a family of circuits $(C'_i)_{i\in\mathbb{N}}$ over B' where we replace every gate in C_i by the corresponding circuit over B'. The circuits in the family $(C'_i)_{i\in\mathbb{N}}$ grow for a constant factor, at most, with respect to the circuits in the family $(C_i)_{i\in\mathbb{N}}$.

In our search for candidates for dual-atoms for FP, our first observation is, that we can rule out all classes FCPS_B where $[B]$ is not a dual-atom of $\mathcal{L}$(BF).

Proposition 5.19. *If $[B]$ is not a dual-atom in $\mathcal{L}$(BF) then FCPS_B cannot be a dual-atom in $\mathcal{L}$(FP).*

Proof. Let $A \in \{\mathrm{R}_0, \mathrm{R}_1, \mathrm{L}, \mathrm{M}, \mathrm{D}\}$ be a dual-atom in $\mathcal{L}$(BF) with $[B] \subset A$, let A' be a base of A, and let $f \in A - [B]$. Of course, all Boolean functions are in FP and no circuit over B (regardless of its size and depth) can compute f. Therefore, $f \in \mathrm{FP} - \mathrm{FCPS}_B$. If FCPS_B were a dual-atom in $\mathcal{L}$(FP) we would expect $[\mathrm{FCPS}_B \cup \{f\}] = \mathrm{FP}$. But since every $g \in \mathrm{FCPS}_B$ can be computed by a family of circuits over B, every $g' \in [\mathrm{FCPS}_B \cup \{f\}]$ can be computed by a family of circuits over $B \cup \{f\}$ and therefore $[\mathrm{FCPS}_B \cup \{f\}] \subseteq \mathrm{FCPS}_{A'} \subset \mathrm{FP}$. ❑

So, for FCPS_B to be a dual-atom in $\mathcal{L}$(FP) it is necessary that $[B]$ is a dual-atom in $\mathcal{L}$(BF). However, this condition is not sufficient: For clones of Boolean functions $[A]$ and $[B]$ where $[A] \subset [B] = \mathrm{BF}$, suppose there exists a function $f \in [A]$ that can be computed by relatively small circuits over B, but all circuits over A computing f are huge in comparison. Then $[\mathrm{FCPS}_A \cup \{f\}]$ could be a proper super-clone of FCPS_A which would still not contain, e.g., the Boolean function $\textsc{Nand} \in \mathrm{FP}$. We will see that this is the case with the monotonic Boolean functions. First, we need to formalize the notion of huge and small circuits with respect to different bases.

Definition 5.20. *Let B_1 and B_2 be finite sets of Boolean functions. We say B_2* is polynomially unnecessary for *B_1 if there is a polynomial p such that for every B_2-circuit C_2 that describes a Boolean function $f \in [B_1]$ there is a B_1-circuit C_1 describing f with* $\mathrm{size}(C_1) \leq p(\mathrm{size}(C_2))$. *Otherwise, we call B_2* polynomially necessary for *B_1.*

Lemma 5.21. *Let B_1, B_2 be sets of Boolean functions. If $[B_1] \in \{\mathrm{L}, \mathrm{D}, \mathrm{R}_1, \mathrm{R}_0\}$ then B_2 is polynomially unnecessary for B_1.*

Proof. If $[B_1] = \mathrm{L}$ then every B_1 circuit can be described by a circuit of linear size in the number of variables.

If $[B_1] = \mathrm{D}$, let $\mathrm{sd}(x,y,z) \stackrel{\text{def}}{=} x\overline{y} \wedge x\overline{z} \wedge \overline{yz}$, which is a base for D. Observe, that $\mathrm{sd}(1,x,y) = \textsc{Nand}(x,y)$. Remember, that the NAND-function is complete for BF. Now let f be an n-ary Boolean function from D. Let $f'(x_2,\ldots,x_n) \stackrel{\text{def}}{=} f(1,x_2,\ldots,x_n)$. Since

$$f'(x_2,\ldots,x_n) = f(\textsc{Nand}(x_n,\textsc{Nand}(x_n,x_n)),x_2,\ldots,x_n)$$

there is a $\{\textsc{Nand}\}$-circuit C' computing f' which has two more gates than a $\{\textsc{Nand}\}$-circuit computing f. We build an $\{\mathrm{sd}\}$-circuit C from C' by replacing every NAND-gate g as follows: If y and z are the inputs of g, replace it by an sd-gate with inputs x_1, y, and z. Here, x_1 is an additional input gate and we connect every g with this gate. Let f_C be the function computed by C. Then $f_C(1,x_2,\ldots,x_n) = f'(x_2,\ldots,x_n) = f(1,x_2,\ldots,x_n)$. On the other hand, since f_C is self-dual, we have $f_C(0,x_2,\ldots,x_n) = \overline{f_C(1,\overline{x_2},\ldots,\overline{x_n})} = \overline{f'(\overline{x_2},\ldots,\overline{x_n})} = \overline{f(1,\overline{x_2},\ldots,\overline{x_n})} = f(0,x_2,\ldots,x_n)$. Hence $f_C = f$.

Let $B_1 = \mathrm{R}_0$. Obviously $\{\vee,\wedge,\oplus\} \subseteq \mathrm{R}_0$. Let $f \in \mathrm{R}_0$ be described by a circuit C over $\{\wedge,\neg\}$ with inputs $x_1,\ldots,x_n$. We build a new circuit C' by replacing every occurrence of a $\neg$-gate in C with an $\oplus$-gate with the following inputs: The first input is the input of the original $\neg$-gate and the second one is $\bigvee_{i=1}^{n} x_i$. Since C' describes an R_0 function, $f_{C'}(0,\ldots,0) = f(0,\ldots,0) = 0$. For every other input, the new $\oplus$-gates behave like $\neg$-gates on their first input, since their second input evaluates to 1. The proof for $B_1 = \mathrm{R}_1$ is due to the duality of R_1 and R_0 and can be done by switching 1 and 0, $\wedge$ and $\vee$, and $\oplus$ and $\leftrightarrow$. ❑

In the following proofs, we need two special functions.

– For $k \in \mathbb{N}$, $x = x_1 \cdots x_n \in \{0,1\}^n$, and $i = \min\{k,n\}$, let $\mathrm{id}_k(x) \stackrel{\text{def}}{=} x_i$.
– For $x,y \in \{0,1\}^*$, let $\mathrm{conc}(x,y) \stackrel{\text{def}}{=} xy$ be the function that concatenates x and y.

Note that for all sets of Boolean functions B, and all $k > 0$, id_k and conc are in FCPS_B.

Theorem 5.22. *Let $B \subseteq \mathrm{BF}$. If $[B] = \mathrm{D}$ then FCPS_B is a dual-atom of* FP.

Proof. Let $[B] = \mathrm{D}$. Obviously $\mathrm{FCPS}_B \subset \mathrm{FP}$. We have to show its precompleteness. For that, let $f \in \mathrm{FP} - \mathrm{FCPS}_B$. Then there is a uniform family of circuits $(C_n^f)_{n\in\mathbb{N}}$ over $\{\vee,\wedge,\neg\}$ that computes f. Since $f \in \mathrm{FCPS}_B$ all C_k^f have polynomial size with respect to k. Therefore, there is a polynomial p such that for $i \in \{1,\ldots,p(k)\}$ Boolean functions f_i^k are computed at the i-th output gate of circuit C_k^f. Since $\{\vee,\wedge,\neg\}$ is polynomially unnecessary for $\{\mathrm{sd}\}$, cf. Lemma 5.21, and since $f \notin \mathrm{FCPS}_B$, there must be a $k \in \mathbb{N}$ and an $\ell \in \{1,\ldots,p(k)\}$ such that $f_\ell^k \notin [\mathrm{sd}]$. That means, that there are $a_1,\ldots,a_k \in \{0,1\}$ such that $f_\ell^k(a_1,\ldots,a_k) = f_\ell^k(\overline{a_1},\ldots,\overline{a_k})$. For $a \in \{0,1\}$, let $a^1 \stackrel{\text{def}}{=} a$ and $a^0 \stackrel{\text{def}}{=} 1-a$. Observe, that the function $h(x_1\ldots x_n) \stackrel{\text{def}}{=} x_1^{a_1}\cdots x_1^{a_k} \in \{0,1\}^k$ is in FCPS_B, since $\mathrm{N} \subset \mathrm{D}$. In fact, h can be calculated by a family of small circuits over any base that can express negation. Then for all $x \in \mathbb{N}$ holds $|h(x)| = k$ and $f_\ell^k(h(x)) = c$ for some $c \in \{0,1\}$.

Therefore there is a $c \in \{0,1\}$ such that the constant function $c(x) = c = \mathrm{id}_\ell(f(h(x)))$ is in $[\mathrm{FCPS}_B \cup \{f\}]$.

Assume that $c = 1$. Now let $g \in \mathrm{FP}$ be computed by a uniform family of circuits $(D_n^g)_{n\in\mathbb{N}}$ over $\{\textsc{Nand}\}$ such that each circuit has polynomial size with respect to n. Regard the family of circuits $(E_n)_{n\in\mathbb{N}}$ where E_1 computes c and E_{n+1} is derived from D_n^g by adding an additional input gate x_0 and replacing every Nand-gate that has inputs x and y by an sd-gate with inputs x_0, x, and y. Obviously, $(E_n)_{n\in\mathbb{N}}$ is uniform. Then for the function g' that is computed by $(E_n)_{n\in\mathbb{N}}$ and all $x \in \{0,1\}^+$ holds $g'(1x) = g(x)$. Hence $g(x) = g'(\mathrm{conc}(c(x), x))$ is in $[\mathrm{FCPS}_B \cup \{f\}]$.

If $c = 0$ the proof is analogous with a family of circuits over $\{\textsc{Nor}\}$. ❑

Theorem 5.23. *Let $B \subseteq \mathrm{BF}$. If $[B] \in \{\mathrm{R}_0, \mathrm{R}_1\}$ then FCPS_B is a precomplete sub-clone of* FP.

Proof. We prove the claim for $[B] = \mathrm{R}_0$. Obviously $\mathrm{FCPS}_B \subset \mathrm{FP}$. We have to show its precompleteness. For that, let $f \in \mathrm{FP} - \mathrm{FCPS}_B$. Then there is a uniform family of circuits $(C_n^f)_{n\in\mathbb{N}}$ over $\{\vee, \wedge, \neg\}$ that computes f. Also, there is a polynomial p, such that for $i \in \{1, \ldots, p(k)\}$ Boolean functions f_i^k are computed by the i-th output gate of circuit C_k^f. Since $\{\vee, \wedge, \neg\}$ is polynomially unnecessary for $\{\vee, \wedge, \oplus\}$, cf. Lemma 5.21, there must be a $k \in \mathbb{N}$ and an $\ell \in \{1, \ldots, p(k)\}$ such that $f_\ell^k \notin [\vee, \wedge, \oplus]$ and $f_\ell^k(0, \ldots, 0) = 1$. Since the constant unary Boolean function c_0 is in R_0, the function c_0^k that maps all words from $\{0,1\}^+$ to 0^k is in FCPS_B. So the constant function c_1 with $c_1(x) = 1$, for all $x \in \mathbb{N}$, is in $[\mathrm{FCPS}_B \cup \{f\}]$, since $c_1(x) = \mathrm{id}_\ell(f(c_0^k(x)))$.

Now let $g \in \mathrm{FP}$. Then there is a uniform family of circuits $(G_n)_{n\in\mathbb{N}}$ over $\{\wedge, \neg\}$ that computes g. Let $g' \in [\mathrm{FCPS}_B \cup \{f\}]$ be computed by the uniform family of circuits $(E_n)_{n\in\mathbb{N}}$ that is derived from $(G_n)_{n\in\mathbb{N}}$ as follows. Let E_1 map the input constantly to 1. Construct E_{n+1} from G_n by introducing a new input gate x_0 and replacing every $\neg$ gate with input y by an $\oplus$ gate with inputs x_0 and y. Obviously, this family of circuits is uniform, and for all $x \in \{0,1\}^+$, $g'(1x) = g(x)$. Therefore $g(x) = g'(\mathrm{conc}(c_1(x), x))$ is in $[\mathrm{FCPS}_B \cup \{f\}]$.

The proof for $[B] = \mathrm{R}_1$ is as above, but $\wedge$ is switched with $\vee$, $\oplus$ is switched with $\leftrightarrow$, and 0 is switched with 1. ❑

Theorem 5.24. *Let $B \subseteq \mathrm{BF}$. If $[B] = \mathrm{M}$, the clone FCPS_B is not precomplete for* FP.

Proof. Razborov [Raz85] proved a super-polynomial lower bound for the size of monotonic circuits computing the perfect matching function. Since this function is in FP, it can be computed by a family of circuits of polynomial size over $\{\wedge, \vee, \neg\}$. That means that there is a function f in $\mathrm{FP} - \mathrm{FCPS}_B$ that can be expressed by a family of monotonic circuits. Therefore $[\mathrm{FCPS}_B \cup \{f\}]$ contains just functions computable by monotonic circuits, but not, for example, the function that flips just the first bit of the input, which is a non-monotonic function that is clearly in FP. ❑

There remains just the one case were $[B] = \mathrm{L}$. In this case FCPS_B is no dual-atom in $\mathcal{L}$(FP), too: Per definition, every linear Boolean function can be described by a very short

propositional formula, and therefore every $f \in \mathrm{FCPS}_B$ can be described by circuits of very small depth. Hence we can use Smolensky's theorem [Smo87] to show that FCPS_B is not a dual-atom in $\mathcal{L}(\mathrm{FP})$. We need a few definitions for that.

Definition 5.25. *Let $y \in \mathbb{N}$ be such that $\mathrm{bin}(y) = a_1 \cdots a_m \in \{0,1\}^m$, $n > 0$ and $x_1, \ldots, x_n \in \{0,1\}$. Let $p \in \mathbb{N}$. We define*

- $\mathrm{mod}_p^n(x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} 1$ *if and only if* $\Sigma_{i=1}^n x_i \equiv 0 (\mathrm{mod}\ p)$
- $\mathrm{mod}_p(y) \stackrel{\mathrm{def}}{=} \mathrm{mod}_p^m(a_1, \ldots, a_m)$.
- $\vee^n(x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} 1$ *if and only if* $\Sigma_{i=1}^n x_i > 0$
- $\wedge^n(x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} 1$ *if and only if* $\Sigma_{i=1}^n x_i = n$
- $\mathrm{MOD}_p \stackrel{\mathrm{def}}{=} \{\neg\} \cup \bigcup_{n \in \mathbb{N}} \{\mathrm{mod}_p^n, \vee^n, \wedge^n\}$

Theorem 5.26 (Smolensky). *Let p be prime and $r > 1$ be relatively prime to p. Then* $\mathrm{mod}_r \notin \mathrm{FSIZE\text{-}DEPTH}_{\mathrm{MOD}_p}(O(n^{O(1)}), O(1))$.

Corollary 5.27. *Let $B \subseteq \mathrm{BF}$. If $[B] = \mathrm{L}$ then FCPS_B is not precomplete for* FP.

Proof. Since every function from L can be computed by a circuit of the form $c_0 \bigoplus_{i=1}^n c_i x_i$, every function in FCPS_B can be computed by a circuit of linear size over MOD_2. However, there are functions in $\mathrm{FSIZE\text{-}DEPTH}_{\mathrm{MOD}_2}(O(1), O(1))$ that are not in $\mathrm{FCPS}(B)$ as, for example, the Boolean function Or. Because of Theorem 5.26, and since $\mathrm{mod}_r \in \mathrm{FP}$ for all $r \in \mathbb{N}$, we obtain

$$\begin{aligned} \mathrm{FCPS}_B &\subsetneq \mathrm{FSIZE\text{-}DEPTH}_{\mathrm{MOD}_2}(O(1), O(1)) \\ &\subseteq \mathrm{FSIZE\text{-}DEPTH}_{\mathrm{MOD}_2}(n^{O(1)}, O(1)) \subsetneq \mathrm{FP}. \end{aligned}$$

We can show that $\mathrm{FSIZE\text{-}DEPTH}_{\mathrm{MOD}_2}(O(1), O(1))$ is a clone with a similar argumentation as in Theorem 5.1, where we prove that FNC^i is a clone for all $i \geq 1$. Therefore there is a clone between FCPS and FP, which means that FCPS is not precomplete for FP. ❑

Let us summarize the results of this subsection.

Corollary 5.28. *Let B be a set of Boolean functions. Then FCPS_B is a dual-atom in $\mathcal{L}(\mathrm{FP})$ if and only if $[B] \in \{\mathrm{R}_0, \mathrm{R}_1, \mathrm{D}\}$.*

By restricting the power of the gates of polynomial-sized circuits, we found three dual-atoms in $\mathcal{L}(\mathrm{FP})$. These classes are weaker than FP itself, however, they are still very near to the full power of FP: So, for every function f from FP there is a function $f' \in \mathrm{FCPS}_{\{\mathrm{And},\mathrm{Xor}\}}$ such that $f'(x) = f(x)$ for all $x \neq 0$ ($[\mathrm{And}, \mathrm{Xor}] = \mathrm{R}_0$). We can bypass this insufficiency by choosing a different encoding for the natural numbers where each number is mapped to a binary string that contains at least one occurrence of 1. Also, for every $f \in \mathrm{FP}$ there is an $f' \in \mathrm{FCPS}_{\{x\overline{y} \vee x\overline{z} \vee \overline{yz}\}}$ such that for all $x \in \{0,1\}^*$ holds $f(x) = f'(1x)$ ($[x\overline{y} \vee x\overline{z} \vee \overline{yz}] = \mathrm{D}$). Again a minimal re-encoding of the natural numbers fixes the insufficiency of the class.

5.4.2 Time-Complexity Classes as Dual-Atoms of FP

We want to find dual-atoms of FP that are not only syntactically weaker than FP. FP is the class of functions that can be computed by a Turing machine in polynomial time, but can we find a more restrictive time bound for Turing machines, such that the functions computed by these machines form a dual-atom of FP? We will see in this subsection that this is not the case.

In particular, we show that there is no $\mathcal{K}$ such that [FDTIME($\mathcal{K}$)] is precomplete for FP. The technique we use is similar to the one used to show that there exist infinitely many Turing degrees between P and NP if P $\neq$ NP [Lad75a, Sch82]. We define classes of functions that on large areas of $\mathbb{N}$ produce very short, sub-polynomial, outputs and on other areas produce outputs of polynomial size. We find such classes that are in fact clones and show that every [FDTIME($\mathcal{K}$)] is either contained in one of these or already contains FP.

For a set $B \subseteq \mathcal{F}^1(\mathbb{N})$ of unary functions, let

$$[B]_{+\text{fict}} \stackrel{\text{def}}{=} \{f : \text{there are } n \geq 1, i \in \{1,\ldots,n\}, \text{ and } f' \in B \text{ such that } f \in \mathcal{F}(\mathbb{N}) \text{ and } f(x_1,\ldots,x_n) = f'(x_i) \text{ for all } x_1,\ldots,x_n \in \mathbb{N}\ \}.$$

That means that $[B]_{+\text{fict}}$ contains all functions f that have at most one non-fictive variable and on this variable f behaves like a function from B. Clearly, $B \subset [B]_{+\text{fict}}$ and if B is closed under substitution, $[B]_{+\text{fict}}$ is closed under superposition. We use this construction in order to not have to deal with all the operations of superposition.

We define the function $\text{pl}(x,y) \stackrel{\text{def}}{=} 2^{(\log x)^y}$; pl stands for **p**ower of **l**ength.

Proposition 5.29. *For all $x, y > 0$ holds* $\text{pl}(\text{pl}(x,y),z) = \text{pl}(x,yz)$.

Proof. Observe the following equation:

$$\begin{aligned} \text{pl}(\text{pl}(x,y),z) &= \text{pl}(2^{(\log x)^y}, z) \\ &= 2^{(\log 2^{(\log x)^y})^z} \\ &= 2^{((\log x)^y)^z} \\ &= 2^{(\log x)^{yz}} \\ &= \text{pl}(x,yz) \end{aligned}$$

❑

In particular, this means that $\text{pl}(\text{pl}(x,y), \frac{1}{y}) = x$.

Definition 5.30. *For a function $g : \mathbb{N} \to \mathbb{N}$, $k \in \mathbb{N}$, and $\ell \geq k$, let*

$$\begin{aligned} \text{lgb}_k^g(\ell) &\stackrel{\text{def}}{=} \text{pl}(g(\ell), 1/2^{\ell-k+1}) \\ \text{rgb}_k^g(\ell) &\stackrel{\text{def}}{=} \text{tp}(2, g(\ell), 2, \ell - k + 1) - 1 \end{aligned}$$

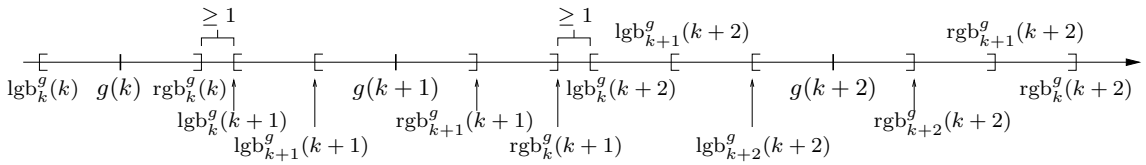


Fig. 5.1: A k-gap-defining function. For every $\ell \geq k$, there are several nested intervals (the gaps) around $g(\ell)$. The number of intervals depends on the size of $\ell - k$. If $k' \geq k$, then g is a k'-gap-defining function. The distance between the right border of an interval around $g(\ell)$ and the left border of an interval around $g(\ell+1)$ has to be at least 1.

Then g is a k-gap-defining function *if for all $\ell \geq k$ holds*

$$\mathrm{rgb}^g_k(\ell) + 1 < \mathrm{lgb}^g_k(\ell+1).$$

Lemma 5.31. *If g is a k-gap-defining function, then g is a k'-gap-defining function for all $k' \geq k$.*

Proof. It suffices to show that g is a $(k+1)$-gap-defining function. For this, let $\ell \geq k+1$. Then

$$\mathrm{rgb}^g_{k+1}(\ell) + 1 < \mathrm{rgb}^g_k(\ell) + 1 < \mathrm{lgb}^g_k(\ell+1) < \mathrm{lgb}^g_{k+1}(\ell+1),$$

and hence g is a $k+1$-gap-defining function. ❑

The intervals $[\mathrm{lgb}^g_k(\ell), \mathrm{rgb}^g_k(\ell)]$ are the gaps that give gap-defining functions their name. These intervals are gaps in the computational power of the functions f, that we define in Definition 5.32. If an input x is in such a gap, then the output of f has to be very short whereas for every other input y, the length of $f(y)$ is not limited.

Definition 5.32. *We say a function f* has (g,k)-gaps *if g is a k-gap-defining function and for all ε with $0 < \varepsilon < 1$ and all $\ell \geq k$ we have*

$$x \in [\mathrm{lgb}^g_k(\ell), \mathrm{rgb}^g_k(\ell)] \Rightarrow |f(x)| \leq |x|^{1+\varepsilon}.$$

For a k-gap-defining function g, let

$$\mathrm{GAPP}_g \stackrel{\mathrm{def}}{=} [\{f : f \in \mathrm{FP} \text{ and there is a } k' \geq k \text{ such that } f \text{ has } (g,k')\text{-gaps}\}]_{+\mathrm{fict}}$$

We give an example of a gap-defining function in the next lemma.

Lemma 5.33. *The function g with $g(0) \stackrel{\mathrm{def}}{=} 1$ and*

$$g(x) \stackrel{\mathrm{def}}{=} \mathrm{pl}(2 \cdot \mathrm{tp}(2, g(x-1), 2, x-1), 2^x),$$

for $x > 0$, is a 1-gap-defining function.

Proof. For all $\ell \geq 1$ holds:

$$\begin{aligned} \mathrm{rgb}_1^g(\ell)+1 &= \mathrm{tp}(2,g(\ell),2,\ell) \\ &< 2\cdot \mathrm{tp}(2,g(\ell),2,\ell) \\ &= \mathrm{pl}(\mathrm{pl}(2\cdot \mathrm{tp}(2,g(\ell),2,\ell),2^{\ell+1}),1/2^{\ell+1}) \\ &= \mathrm{pl}(g(\ell+1),1/2^{\ell+1}) \\ &= \mathrm{lgb}_1^g(\ell+1) \end{aligned}$$

❑

Lemma 5.34. *Let g be a k-gap-defining function, where $k \geq 1$*

1. *For all $\ell \geq k$ and all $0<\varepsilon<1$ holds $|\mathrm{rgb}_k^g(\ell)|^{1+\varepsilon} < |\mathrm{rgb}_{k-1}^g(\ell)|$.*
2. *If f has (g,k)-gaps, then it has (g,k')-gaps for all $k' \geq k$.*

Proof. 1. Let $\ell \geq k$ and $\varepsilon \in (0,1)$. Observe for $a \stackrel{\text{def}}{=} g(\ell)$ and $b \stackrel{\text{def}}{=} 1+\varepsilon$:

$$\begin{aligned} |\mathrm{rgb}_k^g(\ell)|^b &= |\mathrm{tp}(2,a,2,\ell-k+1)-1|^b \\ &= \mathrm{tp}(a,2,\ell-k+1)^b \\ &= \mathrm{tp}(a,b\cdot 2^{\ell-k+1}) \\ &< \mathrm{tp}(a,2\cdot 2^{\ell-k+1}) \\ &= \mathrm{tp}(a,2,\ell-k+2) \\ &= |\mathrm{rgb}_{k-1}^g(\ell)| \end{aligned}$$

2. This holds, since g is a k'-gap-defining function and $\mathrm{rgb}_{k'}^g(\ell) \leq \mathrm{rgb}_k^g(\ell)$ and $\mathrm{lgb}_{k'}^g(\ell) \geq \mathrm{lgb}_k^g(\ell)$, for all $\ell \geq k'$.

❑

The next lemma shows that our classes of functions having gaps, are closed under substitution. Figure 5.2 depictures the idea of the proof.

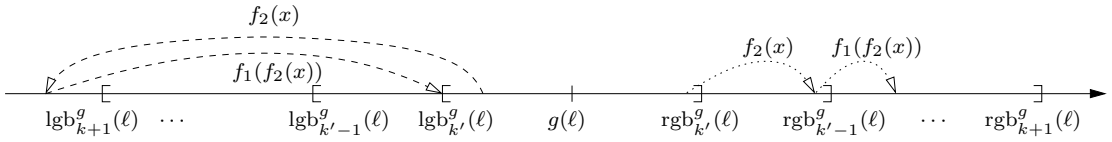


Fig. 5.2: If g is a k-gap-defining function, $f_1, f_2 \in$ FP have (g,k)-gaps, and $k' \geq k$ is large enough, we have the following situation for $f_1 \circ f_2$ and $x \in [\mathrm{lgb}_{k'}^g(\ell), \mathrm{rgb}_{k'}^g(\ell)]$: Either $f_2(x)$ is even smaller than $\mathrm{lgb}_{k+1}^g(\ell)$. Then $f_2(x)$ is more than polynomially smaller than x, which means that $f_1(f_2(x)) \leq x$, since $f_1 \in$ FP. The second case is that $f_2(x)$ falls into a gap of f_1 and therefore $f_1(f_2(x))$ can grow just slightly.

Lemma 5.35. *Let g be a k-gap-defining function. Then* GAPP_g *is closed under substitution.*

Proof. Let $f_1 \in \mathrm{GAPP}_g$ have (g, k_1)-gaps and let $f_2 \in \mathrm{GAPP}_g$ have (g, k_2)-gaps. Without loss of generality, we can assume that $k_1 = k_2 \stackrel{\text{def}}{=} k$, because of Lemma 5.34.2. Let f_1 be such that for all x holds $|f_1(x)| \leq |x|^c$. Choose a k' such that

$$\begin{aligned} k' &\geq k+1, \\ 2^{k'-k-1} &\geq c+1, \text{ and} \\ \log(\mathrm{lgb}^g_{k+1}(k'))^{c+1} &\geq (\log(\mathrm{lgb}^g_{k+1}(k'))+2)^c. \end{aligned}$$

We show that $f_1 \circ f_2$ has (g, k')-gaps. For that, let $\ell \geq k'$ and let $x \in [\mathrm{lgb}^g_{k'}(\ell), \mathrm{rgb}^g_{k'}(\ell)]$.

Case $f_2(x) < x$:
If $f_2(x) \geq \mathrm{lgb}^g_{k+1}(\ell)$, then $|f_2(x)| \leq |x|^{1+\varepsilon}$ for all $0 < \varepsilon < 1$, since f_2 has $(g, k+1)$-gaps. Because of Lemma 5.34.1 we have

$$|f_2(x)|^{1+\varepsilon} < |\mathrm{rgb}^g_{k'}(\ell)|^{1+\varepsilon} < |\mathrm{rgb}^g_k(\ell)|.$$

Since f_1 has (g, k)-gaps,

$$|f_1(f_2(x))| \leq |f_2(x)|^{1+\varepsilon} \leq (|x|^{1+\varepsilon})^{1+\varepsilon} = |x|^{1+2\varepsilon+\varepsilon^2} \tag{5.1}$$

for all $0 < \varepsilon < 1$.
Now let $f_2(x) < \mathrm{lgb}^g_{k+1}(\ell)$ and hence $|f_2(x)|^c \leq |\mathrm{lgb}^g_{k+1}(\ell)|^c$. Then

$$\begin{aligned} f_1(f_2(x)) &< 2^{(|\mathrm{lgb}^g_{k+1}(\ell)|+1)^c} \\ &= 2^{(\lfloor \log(\mathrm{lgb}^g_{k+1}(\ell))\rfloor+2)^c} \\ &\leq 2^{\log(\mathrm{lgb}^g_{k+1}(\ell))^{c+1}} \\ &= \mathrm{pl}(\mathrm{lgb}^g_{k+1}(\ell), c+1) \\ &= \mathrm{pl}(\mathrm{pl}(g(\ell), 1/2^{\ell-k}), c+1) \\ &= \mathrm{pl}(g(\ell), (c+1)/2^{\ell-k}) \\ &\leq \mathrm{pl}(g(\ell), 1/2^{\ell-k'+1}) \\ &= \mathrm{lgb}^g_{k'}(\ell) \\ &\leq x \end{aligned}$$

Case $f_2(x) \geq x$: Then, since f_2 has (g, k')-gaps, $|f_2(x)| \leq |x|^{1+\varepsilon}$ for all $0 < \varepsilon < 1$. Because of Lemma 5.34.1 we have $f_2(x) < \mathrm{lgb}^g_{k'-1}(\ell)$ and since f_1 has $k'-1$-gaps, we can conclude as in Equation 5.1. ❑

Corollary 5.36. *For all k-gap-defining functions g, the class* GAPP_g *is a proper sub-clone of* FP.

Proof. Since GAPP_g is closed under superposition and $\mathrm{GAPP}_g \subseteq \mathrm{FP}$, it is a sub-clone of FP. Since the function $s(x) \stackrel{\text{def}}{=} 1^{|x|^2} \in \mathrm{FP}$ has no (g, k)-gaps, for any gap-defining function g, the inclusion is proper. ❑

We have seen in Corollary 5.12 how to construct a finite base for FP. There are two main ingredients in the proof of the corollary. One is a universal function. In our proof this function had a quadratic running time, but this was in order to keep the proof simple and rather arbitrary. We could have chosen $n^{1+\varepsilon}$ for every $\varepsilon > 0$ as upper bound, with aan appropriate enumeration of Turing machines.

The second main ingredient is the padding function, which has the following important property: For every input x of length n and every $k \in \mathbb{N}$ we can blow up x to a size of n^k with a constant number of applications of the padding function. Again, for the sake of simplicity, we chose a quadratic function. However, it suffices to have a padding function, that inflates every word of length n to a length of $n^{1+\varepsilon}$ for some $\varepsilon > 0$.

The remaining functions of the finite base for FP are a constant, the successor function and a pairing function. All these functions are in FDTIME($O(n^{1+\varepsilon})$) for all $\varepsilon > 0$. Hence the following proposition.

Proposition 5.37. *For every $\varepsilon > 0$ holds* [FDTIME($O(n^{1+\varepsilon})$)] = FP.

Therefore, if there is a dual-atom in $\mathcal{L}$(FP) of the form [FDTIME($\mathcal{K}$)] for some class of functions $\mathcal{K}$, then every function in $\mathcal{K}$ grows more slowly than $n^{1+\varepsilon}$ for every $\varepsilon > 0$. It is obvious, that all these classes are contained in the class SUBP $\stackrel{\text{def}}{=} \bigcap_{\varepsilon>0}$ FDTIME($O(n^{1+\varepsilon})$). SUBP stands for **subp**olynomial time.

Lemma 5.38. SUBP *is a sub-clone of* FP *and it is no dual-atom in the clone lattice below* FP.

Proof. Obviously, SUBP $\subseteq$ FP. We show that it is a clone. For that let $f_1, f_2 \in$ SUBP. Then for all $\varepsilon > 0$, there are constants c, k_1, k_2 such that for all x, we can compute $f_1(f_2(x))$ in time less than $k_1(k_2|x|^{1+\varepsilon})^{1+\varepsilon} = c|x|^{1+2\varepsilon+\varepsilon^2}$ steps.

To show that SUBP is not a dual-atom, we show that SUBP $\subsetneq$ GAPP$_g$ for a k-gap-defining function g. Since obviously SUBP $\subseteq$ GAPP$_g$ for all gap-functions g, we have to show that the inclusion is proper.

Let g be the 1-gap-defining function from Lemma 5.33. Let $s(x) \stackrel{\text{def}}{=} \text{tp}(2, |x|^2)$. Let

$$f(x) \stackrel{\text{def}}{=} \begin{cases} s(x) & \text{, if } x = \text{rgb}_1^g(y) + 1 \text{ for some } y \\ 0 & \text{otherwise.} \end{cases}$$

Since there are infinitely many x such that $|f(x)| > |x|^2$, the function f is not in SUBP. We show that $f \in$ GAPP$_g$. Obviously, $s \in$ FP. So, in order to show that $f \in$ FP, we have to observe the following: We can decide whether there is a y such that $x = \text{rgb}_1^g(y) + 1$ for all $x \in \mathbb{N}$ in polynomial time. We can do this by sequentially computing $\text{rgb}_1^g(0), \text{rgb}_1^g(1), \ldots$, until we find a suitable y, or the result is growing larger than x. Since rgb_1^g grows faster than exponentially, we have to compute less than a linear number of such values.

Hence $f \in$ FP and it remains to show, that there is a k such that f has (g, k)-gaps. This is obvious from the definition of f, since $f(x) = 0$ for all x but those for which there is a y with $x = \text{rgb}_1^g(y) + 1$. It is clear that these x are not in an interval $[\text{lgb}_1^g(\ell), \text{rgb}_1^g(\ell)]$ for all $\ell \geq 1$. So f has $(g, 1)$-gaps and therefore $f \in$ GAPP$_g$. ❑

Theorem 5.39. *For all classes $\mathcal{K}$ of functions,* $[\mathrm{FDTIME}(\mathcal{K})]$ *is not a dual-atom in the lattice of clones below* FP.

Proof. Suppose there is an $\varepsilon > 0$ such that for a function $f \in \mathcal{K}$ holds $f(n) \geq n^{1+\varepsilon}$ for all but finite n. Then $\mathrm{FP} \subseteq [\mathrm{FDTIME}(\mathcal{K})]$ because of Proposition 5.37.

On the other hand, if for all $\varepsilon > 0$ and every $f \in \mathcal{K}$ we have $f(n) < n^{1+\varepsilon}$, then $[\mathrm{FDTIME}(\mathcal{K})] \subseteq \mathrm{SUBP} \subset \mathrm{GAPP}_g \subset \mathrm{FP}$ where g is the 1-gap-defining function from Lemma 5.33. ❑

We remark that the GAPP_g clones are not precomplete for FP themselves, since for a gap-defining function g we can always construct a gap-defining function g' such that g' shares every second gap with g. Then g' is still a gap-defining function, but a less restrictive one, and therefore $\mathrm{GAPP}_g \subset \mathrm{GAPP}_{g'} \subset \mathrm{FP}$.

Although we talked about time-complexity classes $\mathrm{FDTIME}(\mathcal{K})$ and their closures in this subsection, our main argumentation only used the fact, that the length of the output of such functions is bounded in the length of their input. So, the same argumentation would hold true for classes of functions where the computation time is bounded polynomially, but the length of the output is bounded by a sub-polynomial function.

In summary, all dual-atoms that we found for $\mathcal{L}(\mathrm{FP})$ are very close to FP: We can "simulate" every function from FP with a function from one of these dual-atoms if we use a proper encoding of $\mathbb{N}$. We have seen, that there is rich structure of clones between every class $[\mathrm{FDTIME}(\mathcal{K})]$ and FP, if the former is a proper sub-clone of the latter. All this hints, that the structure of clones below FP is very "fine-grained". We saw that the width of $\mathcal{L}(\mathrm{FP})$ is uncountable since there are uncountably many dual-atoms of FP. Since the size of $\mathcal{L}(\mathrm{FP})$ is uncountable, most clones in this lattice have no finite base. The depth, however, is countable, since the number of functions in FP is countable, but infinite, as we saw in subsection 5.2.

In the next chapter, we examine clones with a fixed, small base that contains only one binary function from FP. We will see, that a question concerning a closure property of such a clone characterizes important complexity classes.

6. The Generation Problem

Many important questions could be solved if we could decide for every set of functions B, and every function f, whether or not $f \in [B]$. For example, if we let B be a (finite) base of FL, and c_A be the characteristic function of a P-complete problem A, then P = L if and only if $c_A \in [B]$. However, such problems are undecidable in general. Therefore we study a more special version of this problem. For one, we want to examine the problem just for $B = \{g\}$, where g is a binary function on $\mathbb{N}$. Binary functions are of special interest, since many interesting algebras like monoids and groups are based on this kind of operation. Secondly, we just ask whether there is a function f in $[B]$ such that

$$\begin{aligned} f(a_{1,1}, \ldots, a_{1,n}) &= z_1 \\ f(a_{2,1}, \ldots, a_{2,n}) &= z_2 \\ &\vdots \\ f(a_{m,1}, \ldots, a_{m,n}) &= z_m \end{aligned}$$

where $a_{i,j}, z_i \in \mathbb{N}$ for $1 \leq i \leq m$ and $1 \leq j \leq n$. With other words, we define a function f on a finite number of inputs (its *carrier*) and ask whether there is a function in $[g]$ that has the same input-output behavior as f on these carriers. This is much easier than asking whether an arbitrary function f, perhaps specified by a program for a Turing-machine, is in $[g]$: In this setting we would define f on every possible carrier rather than just finite many ones.

If the number of carriers is 1, an alternative way of putting this problem, is to ask whether z can be *generated* from $a_1, \ldots, a_n$ by repeated application of g. If $m > 1$, it is the question, whether all z_i can be generated from $a_{i,1}, \ldots, a_{i,n}$ where in the generation process we apply g in the same way for all i. This generation problem is a generalization of the following ones:

- Does b belong to the closure of $\{a_1, \ldots, a_n\}$ under pairwise addition? This is equivalent to a modification of the sum-of-subset problem where factors other than 0 and 1 are allowed. It can be shown that this is NP-complete [vEB79].
- Does the empty clause belong to the closure of the clauses $\{\phi_1, \ldots, \phi_n\}$ under the rule of the resolution proof system. This problem is coNP-complete.
- Does a given element of a monoid belong to the sub-monoid that is generated by a given set?

The complexity of special generation problems has been investigated earlier, especially for groups. Generation problems for matrix groups [Bab85, BS84], for finite groups, where the group operation is given by a multiplication table [BKLM01], and for permutation groups [BLS87, FHL84, Sim70] have been examined.

Let us make our generation problems precise. Let g be a computable binary operation on Σ^*, i.e., $g : \Sigma^* \times \Sigma^* \to \Sigma^*$. For fixed g we define the generation problem.

Problem:	Generation problem for a set of carriers, GEN(g)
Input:	$m, a_{1,1}, \ldots, a_{1,n}, \ldots, a_{m,1}, \ldots, a_{m,n}, z_1, \ldots, z_m \in \Sigma^*$.
Question:	Is there an $f \in [g]$ such that $f(a_{i,1}, \ldots, a_{i,n}) = z_i$ for all $1 \leq i \leq m$?

An easier version of the generation problem is the following where we define only one carrier of the function:

Problem:	Generation problem for a single carrier, $\mathrm{GEN_s}(g)$
Input:	$a_1, \ldots, a_n, z \in \Sigma^*$.
Question:	Is there an $f \in [g]$ such that $f(a_1, \ldots, a_n) = z$? In other words: Can z be generated out of $a_1, \ldots, a_n$ by repeated application of g? This is equivalent to asking whether $z \in [a_1, \ldots, a_n]_g$.

For convenience we write operations like addition in infix form. Obviously, $\mathrm{GEN}_s(g) \leq_m^{\log}$ GEN(g) for all binary functions g.

The process of generating elements by iterated application of a binary operation can be visualized by a *generation tree*. For example, $z \in [a_1, \ldots, a_n]_g$ if and only if there is a binary tree of the following form: All leaves of the tree are labeled with a_i where $1 \leq i \leq n$. All inner nodes v of the tree are labeled with $g(x_1, x_2)$, where x_1 is the label of the left successor of v and x_2 is the label of the right successor of v. The root is labeled with z.

We introduce some notation we need in order to work with trees. For a binary tree T, let Leaf(T) be the set of leaves, Root(T) be the root and Node(T) be the set of nodes of T (including the leaves and the root). Let $\mathrm{path}(T) \stackrel{\mathrm{def}}{=} \{w : w \text{ is a path of } T\} \subseteq \{l, r\}^*$. Every $v \in \mathrm{path}(T)$ that does not lead to a leaf node is called *initial path* of T. In contrast, every path in path(T) that is not an initial path is a *full path*. Let ipath(T) be the set of initial paths of T and fpath(T) be the set of full paths in T. For $q \in \mathrm{path}(T)$, let $l(q)$ and $r(q)$ be the number of left turns and right turns, respectively, in q. For a node x of T with path v, let $l(x) \stackrel{\mathrm{def}}{=} l(v)$ (respectively, $r(x) \stackrel{\mathrm{def}}{=} r(v)$).

Definition 6.1. *Let $B \subseteq \Sigma^*$. If f is a binary operation, then a tuple (T, α) where T is a binary tree and $\alpha : \mathrm{Node}(T) \to \Sigma^*$ is a function that labels the nodes of T is called* f-generation tree from B for z *if the following holds:*

- *If $v \in \mathrm{Leaf}(T)$ then $\alpha(v) \in B$.*
- *If $v \notin \mathrm{Leaf}(T)$ and v has left successor v_1 and right successor v_2, then $\alpha(v) = f(\alpha(v_1), \alpha(v_2))$.*
- $\alpha(\mathrm{Root}(T)) = z$.

If α is clear from the context, we call $\alpha(v)$ the *value* of v or the *label* of v. So, T determines the function from $[f]$ that is computed by the generation tree. The values of all inner nodes, including the root, are determined by T and the labels of the leaves.

If f is binary function, the generation process of every function in $[f]$ can be separated into two steps: One step, where the substitution structure, the generation tree, is defined. In the second step no substitution occurs, but variables are permuted and identified.

Proposition 6.2. *Let f be a binary function. The following statements are equivalent:*

1. $(m, a_{1,1}, \ldots, a_{1,n}, \ldots, a_{m,1}, \ldots, a_{m,n}, z_1, \ldots, z_m) \in \mathrm{GEN}(f)$
2. *There is a binary tree T and labeling functions $\alpha_1, \ldots, \alpha_m$ such that*
   a) *for all $i \in \{1, \ldots m\}$ is (T, α_i) is an f-generation tree from $\{a_{i,1}, \ldots, a_{i,n}\}$ for z_i, and*
   b) *for all $v \in \mathrm{Leaf}(T)$ there is a $k \in \{1, \ldots, n\}$ such that for all $i \in \{1, \ldots m\}$ we have $\alpha_i(v) = a_{i,k}$.*

Proof. Let (T, α_i) be as in 2. Then we can build a function $f' \in [f]$ with arity $|\mathrm{Leaf}(T)|$, using the operators SUB and RF on f such that the structure of T is reproduced in the substitution structure. Then the operators TF, RF, and LVF can be used to build an n-ary function f'' that maps all m input tuples to the respective outputs. Due to 2b, exactly the same sequence of these operators can be applied for all i.

The other direction can be shown by an easy induction over the superposition-operators. ❑

6.1 Generation Problems for General Operations

Since we are mostly interested in complexity issues, we restrict ourselves to computable operations. All of the corresponding generation problems are recursively enumerable and we show that there are polynomial-time computable operations whose generation problems are undecidable. There remain undecidable problems even if we furtherly restrict the operation's resources like time and space. The reason is that even with restricted resources it is possible to let a generation problem simulate grammatical derivation trees of arbitrary formal languages. We achieve decidability when we demand the operation to be length-monotonic. Hence we study the complexity of various restrictions of length-monotonic operations.

Theorem 6.3. $\mathrm{GEN}(\circ)$ *is recursively enumerable for every computable operation* $\circ : \Sigma^* \times \Sigma^* \to \Sigma^*$.

Proof. Consider the following algorithm working on $\circ$-formulae, i.e., formulae built up from words in Σ^* using the operation $\circ$. Given a formula F that uses just the variables $x_1, \ldots, x_n$ and the connector $\circ$, the algorithm evaluates F on inputs $(a_{1,1}, \ldots, a_{1,n}), \ldots, (a_{m,1}, \ldots, a_{m,n}) \in (\Sigma^*)^n$ and outputs $m, a_{1,1}, \ldots, a_{1,n}, \ldots, a_{m,1}, \ldots, a_{m,n}, z_1, \ldots, z_m$ where z_i is the value of $F(a_{i,1}, \ldots, a_{i,n})$. Obviously, the algorithm just enumerates $\mathrm{GEN}(\circ)$. ❑

In the next theorem, we observe that polynomial-time computable operations are still too difficult for a complexity-oriented examination of generation problems. For example, with such an operation we can simulate single steps of arbitrary Turing machines.

Theorem 6.4. *There is an associative, commutative, polynomial-time computable operation* $\circ : \Sigma^* \times \Sigma^* \to \Sigma^*$ *such that* $\mathrm{GEN}_s(\circ)$ *is m-complete for recursively enumerable sets.*

Proof. Let $\varphi : \Sigma^* \to \Sigma^*$ be a function that is recursive such that $D_\varphi \stackrel{\text{def}}{=} \{x : \varphi(x) \text{ is defined}\}$ is the halting problem, and let M be a machine that computes φ. We define $\circ$ as follows: For $n, m_1, m_2 \geq 0$ let

$$0^{n+1}1^{m_1} \circ 0^{n+1}1^{m_2} \stackrel{\text{def}}{=} \begin{cases} 0^{n+1}1^{m_1+m_2}, & \text{if } M \text{ on } n \text{ still runs after } m_1+m_2 \text{ steps} \\ 1, & \text{otherwise,} \end{cases}$$

and for all other $x, y \in \Sigma^*$ let $x \circ y \stackrel{\text{def}}{=} 1$.

Observe, that $\circ$ is commutative and $\circ \in \mathrm{FP}$. For associativity let $x, y, z \in \Sigma^*$. In case that there are $n, m_1, m_2, m_3 \geq 0$ such that $x = 0^{n+1}1^{m_1}$, $y = 0^{n+1}1^{m_2}$, $z = 0^{n+1}1^{m_3}$ and M on n does not stop within $m_1+m_2+m_3$ we obtain $x \circ (y \circ z) = (x \circ y) \circ z = 0^{n+1}1^{m_1+m_2+m_3}$. In all other cases we obtain $x \circ (y \circ z) = (x \circ y) \circ z = 1$.

Now, if M on n stops within m steps, then $[0^{n+1}1^1]_\circ = \{0^{n+1}1^1, 0^{n+1}1^2, \ldots, 0^{n+1}1^{m-1}, 1\}$. If M on n does not stop, then $[0^{n+1}1^1]_\circ = \{0^{n+1}1^1, 0^{n+1}1^2, \ldots\}$. Hence,

$$n \in D_\varphi \Leftrightarrow M \text{ on } n \text{ stops } \Leftrightarrow 1 \in [0^{n+1}1^1]_\circ \Leftrightarrow (0^{n+1}1, 1) \in \mathrm{GEN}_s(\circ).$$

❑

6.1.1 Length-Monotonic Polynomial-Time Operations

We have seen that in order to get decidable generation problems we have to restrict the class of operations. Therefore, we demand that in the generation tree of some z, the lengths of all intermediate results are bounded by $|z|$. This is equivalent to saying that we restrict to operations $\circ$ that satisfy $|x \circ y| \geq \max(|x|, |y|)$. Call such operations *length-monotonic*. If $|x \circ y| = \max(|x|, |y|)$, then the operation is called *minimal length-monotonic*. Generation trees of such operations can be exhaustively searched by an alternating polynomial-space machine.

Theorem 6.5. $\mathrm{GEN}(\circ) \in \mathrm{EXPTIME}$ *for every length-monotonic, polynomial-space computable operation* $\circ : \Sigma^* \times \Sigma^* \to \Sigma^*$.

Proof. Let $\circ$ be a length-monotonic, polynomial-space computable operation. $\mathrm{GEN}(\circ)$ can be decided by the following alternating algorithm that uses at most polynomial space:

```
function GEN(m,x[1,1],...,x[1,n],x[m,1],...,x[m,n],z[1],...,z[m])
  repeat
```

```
    if there is a j such that z[i]=x[i,j] for all 1 ≤ i ≤ m then accept;
    if |z[i]| = 0 for an i ∈ {1,...,m} then reject;
    existentially choose z[i,1], z[i,2] such that
      (z[i,1]∘z[i,2]) = z[i] for all 1 ≤ i ≤ m;
    universally choose j from {1,2}
    let z[i]=z[i,j] for all 1 ≤ i ≤ m
  forever
```

Since $\circ$ is computable in polynomial space it is obvious that the above algorithm is an alternating polynomial-space algorithm. Chandra, Kozen, and Stockmeyer [CKS81] proved that these can be simulated in deterministic exponential time. ❑

This exponential-time upper bound for length-monotonic, polynomial-space computable operations is tight, even for polynomial-time computable operations and for the generation problem with a single carrier. To see this we start with a technical lemma which simplifies the argumentation. It shows that for certain sets A, we can translate operations $* : A \times A \to A$ to operations $\circ : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that the complexity of the generation problem with single carriers and other properties are preserved. This is done by an appropriate encoding of elements from A.

Lemma 6.6. *Let $A_1, \ldots, A_{k+l}$ be finite sets, $A \stackrel{\text{def}}{=} A_1^* \times \cdots \times A_k^* \times A_{k+1} \times \cdots \times A_{k+l}$, and let $* : A \times A \to A$ be a polynomial-time computable operation. Then there exists a polynomial-time computable operation $\circ : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that:*

1. $\mathrm{GEN_s}(*) \leq_{\mathrm{m}}^{\mathrm{p}} \mathrm{GEN_s}(\circ)$.
2. *If $*$ is commutative then $\circ$ is commutative.*
3. *If $*$ is associative then $\circ$ is associative.*
4. *If $*$ is minimal length-monotonic then $\circ$ is minimal length-monotonic.*

Proof. Let $m \geq 2$ be such that $|A_i| \leq 2^m$ for $i = 1, 2, \ldots, k+l$. Let $h_i : A_i^* \to (\Sigma^m)^*$ be a continuation of a block encoding with block length m for $i = 1, 2, \ldots, k+l$. Let $d : \Sigma^+ \to \Sigma^+$ be a continuation of the homomorphism defined by $d(0) \stackrel{\text{def}}{=} 00$ and $d(1) \stackrel{\text{def}}{=} 11$ on all binary words. Let $\mathrm{code} : A \to \Sigma^*$ be an encoding given by

$$\mathrm{code}(x_1, x_2, \ldots x_{k+l}) \stackrel{\text{def}}{=} d(h_1(x_1))01d(h_2(x_2))01 \cdots 01d(h_{k+l}(x_{k+l})).$$

Note that $|\mathrm{code}(u)| = 2m|u| + 2(k+l-1)$ and that code is a log-space function. For $w_1, w_2 \in \Sigma^*$, $\circ$ can be defined as

$$w_1 \circ w_2 \stackrel{\text{def}}{=} \begin{cases} \mathrm{code}(u_1 * u_2) & \text{if } w_1 = \mathrm{code}(u_1) \text{ and } w_2 = \mathrm{code}(u_2) \\ 0^{\max(|w_1|,|w_2|)} & \text{otherwise.} \end{cases}$$

Certainly, since $*$ is computable in polynomial time, so is $\circ$. Obviously, if $*$ is commutative then so is $\circ$, and if $*$ is associative then so is $\circ$. Now let $*$ be minimal length-monotonic. If $w_1 = \mathrm{code}(u_1)$, $w_2 = \mathrm{code}(u_2)$, and $u_1 * u_2 = v$ then we conclude:

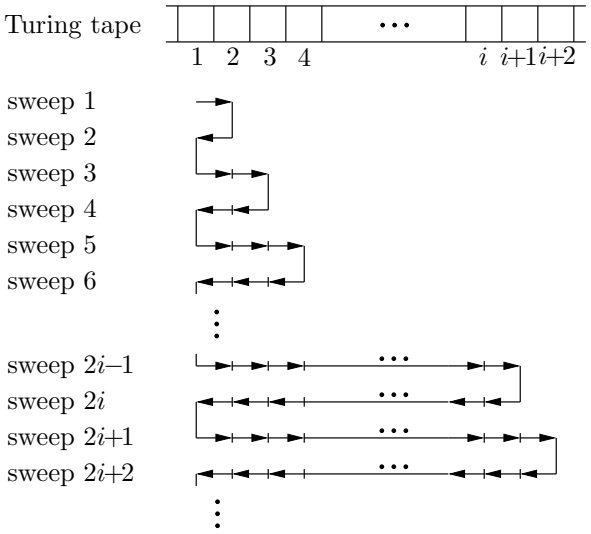
$$\begin{aligned} |w_1 \circ w_2| &= |\mathrm{code}(u_1 * u_2)| = |\mathrm{code}(v)| = 2m|v| + 2(k+l-1) \\ &= 2m \cdot \max(|u_1|, |u_2|) + 2(k+l-1) \\ &= \max(2m|u_1| + 2(k+l-1), 2m|u_2| + 2(k+l-1)) \\ &= \max(|\mathrm{code}(u_1)|, |\mathrm{code}(u_2)|) = \max(|w_1|, |w_2|). \end{aligned}$$

Otherwise, $|w_1 \circ w_2| = |0^{\max(|w_1|,|w_2|)}| = \max(|w_1|, |w_2|)$. Hence, $\circ$ is minimal length-monotonic.

Finally, by definition of $\circ$, $v \in [u_1, \ldots, u_m]_*$ if and only if $\mathrm{code}(v) \in [\mathrm{code}(u_1), \ldots, \mathrm{code}(u_m)]_\circ$. This completes the proof. ❑

Theorem 6.7. *There is a commutative, minimal length-monotonic, polynomial-time computable operation* $\circ : \Sigma^* \times \Sigma^* \to \Sigma^*$ *such that* $\mathrm{GEN_s}(\circ)$ *is* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-complete for* EXPTIME.

Proof. We follow an idea of Cook [Coo71] to simulate deterministic exponential-time computations. Without loss of generality, a deterministic exponential-time one-tape Turing machine M deciding a set $A \subseteq \Sigma^*$ can be normalized in such a way that on input $x = a_1 a_2 \cdots a_n$ it makes $2^{p(|x|)}$ *sweeps* where p is a suitable polynomial. For $0 \leq 2i < 2^{p(|x|)}$, the $(2i+1)$-st sweep is a right move from tape cell 1 (with the first symbol of x) to tape cell $i+2$ within $i+1$ steps, and the $(2i+2)$-nd sweep is a left move from tape cell $i+2$ to tape cell 1 within $i+1$ steps. Each of the turning points belongs to two sweeps.



Furthermore, let M have the tape alphabet Δ, the set of states S, the initial state s_0, and the accepting state s_1. In the case of acceptance the tape of M is empty. If M is in state s and reads a, then the next state is $\sigma(s,a)$, and the symbol printed is $\lambda(s,a)$.

We say that the quintuple (x,i,j,s,a) is *correct* if during the i-th sweep on input x the machine M prints the symbol a in tape cell j and leaves that cell with state s. One can compute a correct (x,i,j,s,a) by knowing only two other correct quintuples, namely the correct $(x,i-1,j,s',a')$ and the correct (x,i,k,s'',a'') where $k \in \{j-1, j+1\}$. The idea of our operation is as follows: multiply $(x,i-1,j,s',a')$ with (x,i,k,s'',a'') and obtain

(x, i, j, s, a). In an accepting computation of M on x (and only in this case) one generates finally the correct $(x, 2^{p(|x|)}, 2, s_1, \square)$.

To make this precise, let $a_j \stackrel{\text{def}}{=} \square$ for all $j > n$. Furthermore we assume that, in a quintuple (x, i, j, s, a) where $i, j \in \{0, 1, \ldots, 2^{p(|x|)}\}$, the numbers i and j are given in binary representation of length exactly $p(|x|) + 1$. Now define the operation $*$ as follows.
Right sweep, for $1 \leq 2i < 2^{p(|x|)}$ and $j = 1, 2, \ldots, i$:

$$(x, 2i, j+1, s, a) * (x, 2i+1, j, s', b) \stackrel{\text{def}}{=} (x, 2i+1, j+1, \sigma(s', a), \lambda(s', a))$$

Left sweep, for $1 \leq 2i + 1 < 2^{p(|x|)}$ and $j = 1, 2, \ldots, i+1$:

$$(x, 2i+1, j, s, a) * (x, 2i+2, j+1, s', b) \stackrel{\text{def}}{=} (x, 2i+2, j, \sigma(s', a), \lambda(s', a))$$

New tape cell right, for $1 \leq 2i + 1 < 2^{p(|x|)}$:

$$(x, 2i+1, i+1, s, a) * (x, 0, 0, s_0, \square) \stackrel{\text{def}}{=} (x, 2i+1, i+2, \sigma(s, a_{i+2}), \lambda(s, a_{i+2}))$$

Turning point left, for $1 \leq 2i < 2^{p(|x|)}$:

$$(x, 2i, 1, s, a) * (x, 0, 0, s_0, \square) \stackrel{\text{def}}{=} (x, 2i+1, 1, s, a)$$

Turning point right, for $1 \leq 2i + 1 < 2^{p(|x|)}$:

$$(x, 2i+1, i+2, s, a) * (x, 0, 0, s_0, \square) \stackrel{\text{def}}{=} (x, 2i+2, i+2, s, a)$$

If $u * v$ is defined in this way then $v * u$ is defined in the same way. For remaining products not yet defined, we define

$$(x, u, v, s, a) * (x', u', v', s', a') \stackrel{\text{def}}{=} (0^{\max(|x|+|u|+|v|, |x'|+|u'|+|v'|)}, \varepsilon, \varepsilon, s_0, \square).$$

Obviously, $*$ is polynomial-time computable, minimal length-monotonic, and commutative. Starting with $(x, 1, 1, \sigma(s_0, a_1), \lambda(s_0, a_1))$ and $(x, 0, 0, s_0, \square)$ exactly the correct quintuples of the form $(x, \ldots)$ together with $(0^{|x|+2p(|x|)+2}, \varepsilon, \varepsilon, s_0, \square)$ and $(x, 0, 0, s_0, \square)$ can be generated. Hence, M accepts x if and only if

$$((x, 1, 1, \sigma(s_0, a_1), \lambda(s_0, a_1)), (x, 0, 0, s_0, \square), (x, 2^{p(|x|)}, 2, s_1, \square)) \in \mathrm{GEN}_{\mathrm{s}}(*),$$

consequently $A \leq^{\mathrm{p}}_{\mathrm{m}} \mathrm{GEN}_{\mathrm{s}}(*)$. By Lemma 6.6, we obtain a polynomial-time computable, minimal length-monotonic and commutative operation $\circ : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that $A \leq^{\mathrm{p}}_{\mathrm{m}} \mathrm{GEN}_{\mathrm{s}}(\circ)$. ❑

6.1.2 Finiteness of Generated Sets: An Excursion

Although this chapter is about the complexity of the generation problem, we want to examine in this subsection an interesting question that is strongly related to the generation problem. Namely, we examine, how difficult it is to determine whether or not the sets generated by a function f from a basic set is finite. It turns out, that this problem is very difficult. It is undecidable, even for length-monotonic functions. Let us first give a formal definition of the finiteness problems for functions $f \in \mathcal{F}^2(\mathbb{N})$.

Problem:	Finiteness Problem FIN(f)
Input:	A finite $B \subset \mathbb{N}$.
Question:	Is $[B]_f$ finite?

Theorem 6.8. *For all computable f is* $\mathrm{FIN}(f)$ *recursively enumerable.*

Proof. Obviously, the set $\{(A,B) : B \subseteq A = [A]_f \text{ and } A \text{ is finite}\}$ is decidable. So $\mathrm{FIN}(f)$ is the projection of a decidable set and hence recursively enumerable. ❑

In Theorem 6.4, we show that there is an associative, commutative, polynomial-time computable function f, such that $\mathrm{GEN_s}(f)$ is undecidable. In the proof, a 1 is generated if and only if the generated set is finite. Hence, the result carries over to $\mathrm{FIN}(f)$.

Theorem 6.9. *There is an associative, commutative, polynomial-time computable operation $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that* $\mathrm{FIN}(f)$ *is m-complete for recursively enumerable sets.*

However, for length-monotonic, polynomial-space computable functions f, we show in Theorem 6.5 that $\mathrm{GEN}(f)$ is decidable in EXPTIME. This is not true for $\mathrm{FIN}(f)$; we even find $f \in \mathrm{FP}$ such that $\mathrm{FIN}(f)$ is undecidable.

Theorem 6.10. *There is a length-monotonic, associative, and commutative function $f \in$* FP *such that* $\mathrm{FIN}(f)$ *is m-complete for recursively enumerable sets.*

Proof. Let $a, b, c \in \{0,1\}^2$ such that they differ pairwise, let φ be a recursive function where D_φ is the halting problem, and let M compute φ. In the case that M halts on some input n, let t_n be the number of steps M makes until it halts. Let $L_n \stackrel{\text{def}}{=} \{a^n b^k : k \in \mathbb{N}\}$. For all $n \in \mathbb{N}$ we define the functions $f_n : L_n \times L_n \to L_n$ as follows:

$$f_n(a^n b^{m_1}, a^n b^{m_2}) \stackrel{\text{def}}{=} \begin{cases} a^n b^{2t_n} & \text{, if } M \text{ halts on } n \text{ and } (m_1 \geq t_n \text{ or } m_2 \geq t_n) \\ & \quad \text{and } m_1, m_2 \leq 2t_n \\ a^n b^{m_1} & \text{, if } M \text{ halts on } n \text{ and } m_1 > 2t_n \text{ and } m_2 \leq 2t_n \\ a^n b^{m_2} & \text{, if } M \text{ halts on } n \text{ and } m_2 > 2t_n \text{ and } m_1 \leq 2t_n \\ a^n b^{m_1+m_2} & \text{otherwise} \end{cases}$$

Observe, that $f_n \in \mathrm{FP}$, that f_n is length-monotonic and commutative for all $n \in \mathbb{N}$. We will now show that f_n is associative for all $n \in \mathbb{N}$: If M does not halt on n, then obviously $f_n(f_n(a^n b^{m_1}, a^n b^{m_2}), a^n b^{m_3}) = a^n b^{m_1+m_2+m_3} = f_n(a^n b^{m_1}, f_n(a^n b^{m_2}, a^n b^{m_3}))$, so let there be an t_n such that M halts on input n after t_n steps.
Case 1: $m_1, m_2, m_3 \leq 2t_n$
Then either $f_n(f_n(a^n b^{m_1}, a^n b^{m_2}), a^n b^{m_3}) = a^n b^{m_1+m_2+m_3} = f_n(a^n b^{m_1}, f_n(a^n b^{m_2}, a^n b^{m_3}))$ or $f_n(f_n(a^n b^{m_1}, a^n b^{m_2}), a^n b^{m_3}) = a^n b^{2t_n} = f_n(a^n b^{m_1}, f_n(a^n b^{m_2}, a^n b^{m_3}))$.
Case 2: $m_1 > 2t_n$, $m_2, m_3 \leq 2t_n$
Then $f_n(f_n(a^n b^{m_1}, a^n b^{m_2}), a^n b^{m_3}) = f_n(a^n b^{m_1}, a^n b^{m_3}) = a^n b^{m_1}$. On the other side $f_n(a^n b^{m_1}, a^n b^{m_2+m_3}) = a^n b^{m_1} = f_n(a^n b^{m_1}, a^n b^{2t_n})$. Note that the choice of m_1 to be greater than $2t_n$ is arbitrary, since f_n is commutative.
Case 3: $m_1, m_2 > 2t_n$, $m_3 \leq 2t_n$
In this case $f_n(f_n(a^n b^{m_1}, a^n b^{m_2}), a^n b^{m_3}) = a^n b^{m_1+m_2} = f_n(a^n b^{m_1}, f_n(a^n b^{m_2}, a^n b^{m_3}))$.
Case 4: $m_1, m_2, m_3 > 2t_n$
In this case $f_n(f_n(a^n b^{m_1}, a^n b^{m_2}), a^n b^{m_3}) = a^n b^{m_1+m_2+m_3} = f_n(a^n b^{m_1}, f_n(a^n b^{m_2}, a^n b^{m_3}))$.

This finishes the proof of the associativity of f_n. Note, that $n \in D_\varphi$ if and only if $[a^n b]_{f_n}$ is finite. In the rest of the proof, we will define a length-monotonic, associative, commutative function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ that is a expansion of all f_n.

For given $k, n_1, \ldots n_k, m_1, \ldots, m_k \in \mathbb{N}$, $i \in \{1, \ldots, k\}$, let

$$g(n_i) \stackrel{\text{def}}{=} \begin{cases} n_i & \text{, if there is no } j \in \{1, \ldots, i-1\} \text{ with } n_i = n_j \\ n_i' & \text{, otherwise, where } n_i' \stackrel{\text{def}}{=} |\{(j,o) : 1 \le j < o \le i \wedge n_j = n_o\}| \\ & \quad + \max\{n_1, \ldots, n_k\} \end{cases}$$

Observe, that $g(n_i) \neq g(n_j)$ for all $i, j \in \{1, \ldots, k\}$ and let $g'(a^{n_1} b^{m_1} c \ldots c a^{n_k} b^{m_k}) \stackrel{\text{def}}{=} a^{g(n_1)} b^{m_1} c \ldots c a^{g(n_k)} b^{m_k}$. If there are no $i, j \in \{1, \ldots, k\}$ with $n_i = n_j$, let $h(a^{n_1} b^{m_1} c \ldots c a^{n_k} b^{m_k}) \stackrel{\text{def}}{=} a^{n_{i_1}} b^{m_{i_1}} c \ldots c a^{n_{i_k}} b^{m_{i_k}}$ such that $\{n_{i_1}, \ldots, n_{i_k}\} = \{n_1, \ldots, n_k\}$ and for all $r, s \in \{1, \ldots, k\}$ holds $r < s \to n_{i_r} < n_{i_s}$. For $w \in \Sigma^*$, let

$$d(w) \stackrel{\text{def}}{=} \begin{cases} w & \text{, if } w = a^{n_1} b^{m_1} c \ldots c a^{n_k} b^{m_k} \text{ for some} \\ & \quad k, n_1, \ldots, n_k, m_1, \ldots, m_k \in \mathbb{N} \\ a^0 b^{|w|} & \text{otherwise} \end{cases}$$

and $\text{clean}(w) \stackrel{\text{def}}{=} h(g'(d(w)))$. Observe, that $\text{clean}(w) = a^{n_1} b^{m_1} c \ldots c a^{n_k} b^{m_k}$ such that $k, n_1, \ldots, n_k, m_1, \ldots, m_k \in \mathbb{N}$ and $n_1 < n_2 < \cdots < n_k$. Now, let $v \stackrel{\text{def}}{=} a^{n_1} b^{m_1} c \ldots c a^{n_k} b^{m_k}$ and $w \stackrel{\text{def}}{=} a^{o_1} b^{p_1} c \ldots c a^{o_l} b^{p_l}$ such that $l, k, n_1, \ldots, n_k, m_1, \ldots, m_k, o_1, \ldots, o_l, p_1, \ldots, p_l \in \mathbb{N}$ and $n_1 < n_2 < \cdots < n_k$ and $o_1 < o_2 < \cdots < o_l$. Let $\{w_1, \ldots, w_r\} = \{a^{o_j} b^{p_j} : 1 \le j \le l \text{ and there is no } i \in \{1, \ldots, k\} \text{ with } n_i = o_j\}$. If $r > 0$, let $f'(v, w) \stackrel{\text{def}}{=} v_1 c \ldots c v_k c w_1 c \ldots c w_r$, where

$$v_i \stackrel{\text{def}}{=} \begin{cases} f_{n_i}(a^{n_i} b^{m_i}, a^{o_j} b^{p_j}) & \text{, if there is a } j \in \{1, \ldots, l\} \text{ with } n_i = o_j \\ a^{n_i} b^{m_i} & \text{otherwise} \end{cases}$$

and $f'(v, w) \stackrel{\text{def}}{=} v_1 c \ldots c v_k$, otherwise. Finally, for arbitrary $v, w \in \Sigma^*$, let $f(v, w) \stackrel{\text{def}}{=} h(f'(\text{clean}(v), \text{clean}(w)))$. Observe, that $f \in \text{FP}$ and that it is length-monotonic, commutative, and associative and that for all $n \in \mathbb{N}$ we have $[\{a^n b\}]_f = [\{a^n b\}]_{f_n}$. ❑

6.1.3 Length-Monotonic Associative Polynomial-Time Operations

We have seen in Theorem 6.7 that, in general, commutativity does not lower the complexity of the generation problem for length-monotonic, polynomial-time computable operations. In this subsection we show that associativity does. If we want to generate a number z from a basic set by repeated application of an associative operation $\circ$, we do not need to know the exact structure of a $\circ$-generation tree for z: Associativity makes all generation trees with the same sequence of leaves equivalent with respect to the generated element. As we have seen in Proposition 6.2, this effect carries over to the functions in $[\circ]$, which means the the sequence of substitutions we use to build a function from $[\circ]$ is not relevant. We show that

PSPACE is upper bound for all generation problems with associative, polynomial-space computable operations and that there are even associative, polynomial-time computable operations such that their generation problem is a lower bound for PSPACE.

Theorem 6.11. $\mathrm{GEN}(\circ) \in \mathrm{PSPACE}$ *if* $\circ : \Sigma^* \times \Sigma^* \to \Sigma^*$ *is length-monotonic, associative, and polynomial-space computable.*

Proof. The following algorithm decides $\mathrm{GEN}(\circ)$ in polynomial space:

```
function GEN(m,x[1,1],...,x[1,n],...,x[m,1],...,x[m,n],z[1],...,z[m]);
  choose an i ∈ {1,...,n} nondeterministically;
  y[j] := x[j,i] for all 1 ≤ j ≤ m;
  while (y[j] ≠ z[j] for a j ∈ {1,...,m}) and
        (|y[j]| ≤ |z[j]| for all j ∈ {1,...,m}) do begin
    choose an i ∈ {1,...,n} nondeterministically;
    y[j] := y[j] ∘ x[j,i] for all 1 ≤ j ≤ m
  end;
  if (y[j] = z[j] for all 1 ≤ j ≤ m) then accept else reject
```

❑

The polynomial-space bound is tight even for a polynomial-time operation $\circ$ and the generation problem for a single carrier.

Theorem 6.12. *There is a minimal length-monotonic and associative polynomial-time computable operation* $\circ : \Sigma^* \times \Sigma^* \to \Sigma^*$ *such that* $\mathrm{GEN_s}(\circ)$ *is* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-complete for* PSPACE.

Proof. At the beginning we remark that much of the complexity of the following construction stems from the possible associativity of the operation. Let $L \subseteq \Sigma^*$ be a set that is $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for PSPACE such that $\varepsilon \notin L$. By Lemma 6.6, it suffices to prove existence of a finite alphabet Δ and a minimal length-monotonic and associative polynomial-time computable operation $* : (\Sigma^* \times \Delta^*) \times (\Sigma^* \times \Delta^*) \to (\Sigma^* \times \Delta^*)$ such that $L \leq^{\mathrm{p}}_{\mathrm{m}} \mathrm{GEN_s}(*)$.

Since $L \in \mathrm{PSPACE}$, it follows [CF91] that there exists a polynomial-time computable function $f : \Sigma^* \times \mathbb{N} \to \mathrm{A}_5$ and a polynomial p such that for all $x \in \Sigma^*$,

$$x \in L \leftrightarrow f(x,0) \cdot f(x,1) \cdot \cdots \cdot f(x, 2^{p(|x|)} - 2) = a_0 \tag{6.1}$$

where $(\mathrm{A}_5, \cdot)$ is the group of even permutations on five elements with identity permutation a_0. For $x \in \Sigma^*$, let $K_x \stackrel{\mathrm{def}}{=} p(|x|)$ and $M_x \stackrel{\mathrm{def}}{=} 4K_x + 3$ (x will always be clear from the context).

We consider the set

$$\{(x, \mathrm{bin}_{K_x}(i)\, a\, \mathrm{bin}_{K_x}(j)) : 0 \leq i < j < 2^{K_x}, a \in \mathrm{A}_5\} \subseteq \{x\} \times (\Sigma^{K_x} \cdot \mathrm{A}_5 \cdot \Sigma^{K_x})$$

with a multiplication $*$ whose essential idea is given by the following equation:

$$(x, \mathrm{bin}_{K_x}(i)\, a\, \mathrm{bin}_{K_x}(j)) * (x, \mathrm{bin}_{K_x}(j{+}1)\, b\, \mathrm{bin}_{K_x}(m)) = (x, \mathrm{bin}_{K_x}(i)\, a \cdot f(x,j) \cdot b\, \mathrm{bin}_{K_x}(m)).$$

However, we need $*$ to be defined in a more general way; the exact definition follows. From equation (6.1) we obtain

$$x \in L \leftrightarrow (x, 0^{K_x} a_0 1^{K_x}) \in [(x, \mathrm{bin}_{K_x}(i) f(x,i) \mathrm{bin}_{K_x}(i+1)) : 0 \le i < 2^{K_x} - 1]_*.$$

Since $\{(x, \mathrm{bin}_{K_x}(i) f(x,i) \mathrm{bin}_{K_x}(i+1)) : i < 2^{K_x} - 1\}$ has exponentially many elements (in the length of x), this cannot be used as reduction function for $L \le^{\mathrm{p}}_{\mathrm{m}} \mathrm{GEN}_{\mathrm{s}}(*)$. So we have to generate this set from a few basic pairs. For this we modify $*$ as follows. We use a new *separation symbol* $\#$ and, to achieve minimal length-monotonicity, a new *padding symbol* 2. For $u \in \{0,1,\#\}^*$, let $\mathrm{p}_x(u) \stackrel{\mathrm{def}}{=} u 2^{M_x - |u|}$ and for $w \in \{0,1,2,\#\}^*$, let $\overline{w} \in \{0,1,\#\}^*$ be the word w without symbols 2. Define the following sets of words:

- $A_x \stackrel{\mathrm{def}}{=} \{w \in \Sigma^* : |w| \le K_x\}$
- $B_x \stackrel{\mathrm{def}}{=} A_x \# A_x$
- $C_x \stackrel{\mathrm{def}}{=} A_x \# \Sigma^{K_x} \# A_x$
- $D_x \stackrel{\mathrm{def}}{=} \{u \,\#\, \mathrm{bin}_{K_x}(i_1)\, c_1\, \mathrm{bin}_{K_x}(i_2)\, c_2 \cdots c_{s-1}\, \mathrm{bin}_{K_x}(i_s) \,\#\, u' :$
  $s \ge 2,\ u, u' \in A_x,\ 0 \le i_1 < \cdots < i_s < 2^{K_x},$
  $c_1, c_2, \ldots, c_{s-1} \in \mathrm{A}_5 \cup \{\#\}$, and
  $(c_j = \# \Rightarrow i_j + 1 = i_{j+1})$ for $j = 1, \ldots, s-1\}$
- $G_x \stackrel{\mathrm{def}}{=} A_x \cup B_x \cup C_x \cup D_x$

Let $\Delta \stackrel{\mathrm{def}}{=} \{0,1,2,\#\} \cup \mathrm{A}_5$ and define $g_x : (\{0,1,\#\} \cup \mathrm{A}_5)^* \to (\{0,1,\#\} \cup \mathrm{A}_5)^*$ as follows

1. $g_x(v) \stackrel{\mathrm{def}}{=} v$ if $v \in A_x \cup B_x \cup C_x$.
2. If $v = u\#\mathrm{bin}_{K_x}(i_1) c_1 \mathrm{bin}_{K_x}(i_2) c_2 \ldots c_{s-1} \mathrm{bin}_{K_x}(i_s) \# u' \in D_x$ then

   $$g_x(v) \stackrel{\mathrm{def}}{=} u\#\mathrm{bin}_{K_x}(i_1) a \mathrm{bin}_{K_x}(i_s) \# u',$$

   where $a \stackrel{\mathrm{def}}{=} b_1 \cdot b_2 \cdot \ldots \cdot b_{s-1}$ such that $b_j = c_j$ if $c_j \in \mathrm{A}_5$ and $b_j = f(x, i_j)$ otherwise.
3. $g_x(v) \stackrel{\mathrm{def}}{=} \#\#\#$ in all other cases, i.e., if $v \notin G_x$.

Finally, define $*$ on $\Sigma^* \times \Delta^*$ by

$$(x,v) * (y,w) \stackrel{\mathrm{def}}{=} \begin{cases} (\varepsilon, 2^{\max\{|x|+|v|, |y|+|w|\}}) & \text{, if } x \ne y \text{ or } x = \varepsilon \text{ or } y = \varepsilon \text{ or one of} \\ & \overline{v}, \overline{w} \text{ is not in } (G_x \cup \{\#\#\#\}) \cap \Delta^{M_x} \\ (x, \mathrm{p}_x(g_x(\overline{vw}))) & \text{, otherwise} \end{cases}$$

Observe, that $*$ is minimal length-monotonic, which is basically ensured by the padding function $\mathrm{p}_x(\cdot)$.

We show the associativity for $*$. For that, let first $r \stackrel{\mathrm{def}}{=} (x, r'), s \stackrel{\mathrm{def}}{=} (y, s'), t \stackrel{\mathrm{def}}{=} (z, t') \in \Sigma^* \times \Delta^*$ such that $|\{x, y, z\}| > 1$. Then $r*(s*t) = (\varepsilon, 2^{\max\{|x|+|r'|, |y|+|s'|, |z|+|t'|\}}) = (r*s)*t$. We obtain the same result, if $x = y = z$ and one of $\overline{r'}, \overline{s'}, \overline{t'}$ is not from $(G_x \cup \{\#\#\#\}) \cap \Delta^{M_x}$. The remaining cases are such that $x = y = z$ and $r', s', t' \in (G_x \cup \{\#\#\#\}) \cap \Delta^{M_x}$. Here it suffices to show

$$r * (s * t) = (x, \mathrm{p}_x(g_x(\overline{r's't'}))). \tag{6.2}$$

If one of $\overline{r'}, \overline{s'}, \overline{t'}$ is equal to ###, this is obvious. The same holds in the case where $\overline{s't'} \in A_x \cup B_x \cup C_x$. If $\overline{s't'} = ###$, then $r*(s*t) = (x, \mathrm{p}_x(g_x(\overline{r's't'}))) = (x, \mathrm{p}_x(###)) \stackrel{\text{def}}{=} \alpha$ and we are done.

If $\overline{s't'} = u\#\mathrm{bin}_{K_x}(i_1)c_1\mathrm{bin}_{K_x}(i_2)c_2 \cdots c_{s-1}\mathrm{bin}_{K_x}(i_s)\#u' \in D_x$. Then $s*t = (x, d)$ where $d = \mathrm{p}_x(u\#\mathrm{bin}_{K_x}(i_1)a\mathrm{bin}_{K_x}(i_s)\#u')$ as in the second case of the definition of g_x. Assume $\overline{r'} = v \in A_x$ or $\overline{r'} = w\#v \in B_x \cup C_x \cup D_x$. If $|vu| > K$ then $r*(s*t) = \alpha = \mathrm{p}_x(g_x(\overline{r's't'}))$. If $|vu| < K$ and $\overline{r'} \notin A_x$ then again $r*(s*t) = \alpha = \mathrm{p}_x(g_x(\overline{r's't'}))$. If $|vu| \leq K_x$ and $\overline{r'} \in A_x$ we have $r * (s * t) = (x, \mathrm{p}_x(vu\#\mathrm{bin}_{K_x}(i_1)a\mathrm{bin}_{K_x}(i_s)\#u')) = (x, \mathrm{p}_x(g_x(\overline{r's't'})))$. The remaining cases are where $\overline{r'} = w\#v \in B_x \cup C_x \cup D_x$ and $|vu| = K_x$. Since $u\#\mathrm{bin}_{K_x}(i_1)$ is a prefix of both $\overline{s't'}$ and $g_x(\overline{s't'})$, we have $\overline{r'}g_x(\overline{s't'}) \in D_x$ if and only if $\overline{r's't'} \in D_x$. If $\overline{r's't'} \notin D_x$, then $r * (s * t) = \alpha = \mathrm{p}_x(g(\overline{r's't'}))$, so let $\overline{r's't'} \in D_x$. In this case the equivalence (6.2) can be easily seen for all cases of $\overline{r'}$.

The remaining case is where $\overline{s't'} \notin G_x \cup \{###\}$; we show that $\overline{r's't'} \notin G_x \cup \{###\}$. Obviously, $\overline{r's't'} \neq ###$. Suppose that $\overline{r's't'} \in G_x$. If $\overline{r's't'} \in A_x$, then $\overline{s't'} \in A_x$. If $\overline{r's't'} \in B_x$, then $\overline{s't'} \in A_x \cup B_x$. If $\overline{r's't'} \in C_x$, then $\overline{s't'} \in A_x \cup B_x \cup C_x$. Therefore $\overline{r's't'} = u\#\mathrm{bin}_{K_x}(i_1)c_1\mathrm{bin}_{K_x}(i_2)c_2 \cdots c_{s-1}\mathrm{bin}_{K_x}(i_s)\#u' \in D_x$. Since $\overline{s't'} \notin G_x$ and $\overline{r'} \in G_x$, there is a k such that $\overline{r'} = u\#\mathrm{bin}_{K_x}(i_1)c_1\mathrm{bin}_{K_x}(i_2)c_2 \cdots c_{k-1}w$, $\overline{s't'} = w'c_k\mathrm{bin}_{K_x}(i_{k+1}) \cdots c_{s-1}\mathrm{bin}_{K_x}(i_s)\#u'$ where $ww' = \mathrm{bin}_{K_x}(i_k)$, $c_{k-1} = \#$, and $c_k \in \mathrm{A}_5$. Hence either $\overline{s'}$ or $\overline{t'}$ is not in G_x. So if $\overline{s't'} \notin G_x$, then $r * (s * t) = \alpha = (x, \mathrm{p}_x(g_x(\overline{r's't'})))$. This finishes the proof of associativity for $*$.

Observe that $(x, \mathrm{p}_x(u\#\mathrm{bin}_{K_x}(i)a\mathrm{bin}_{K_x}(j)\#v))$ is in $[(x, \mathrm{p}_x(0)), (x, \mathrm{p}_x(1)), (x, \mathrm{p}_x(\#))]_*$ if and only if $i < j$ and $f(x,i) \cdot f(x,i+1) \cdot \cdots \cdot f(x,j-1) = a$. Consequently, we obtain

$$x \in L \leftrightarrow (x, \mathrm{p}_x(\#0^K a_0 1^K \#)) \in [(x, \mathrm{p}_x(0)), (x, \mathrm{p}_x(1)), (x, \mathrm{p}_x(\#))]_*.$$

❑

Now let us additionally assume $\circ$ to be commutative. Again, the associativity enables us to ignore the $\circ$-generation tree and instead search for a word over $\{1, \ldots, n\}$. Together with commutativity, we just have to guess exponents $k_1, \ldots, k_n$ and test whether $a_{i,1}^{k_1} \circ \cdots \circ a_{i,n}^{k_n} = z$ for all $1 \leq i \leq m$. If the operation is computable in polynomial-time, then the exponentiations are computable in polynomial-time, too (by squaring and multiplying), which yields the following theorem.

Theorem 6.13. $\mathrm{GEN}(\circ) \in \mathrm{NP}$ *for all length-monotonic, associative, and commutative polynomial-time computable operations* $\circ : \Sigma^* \times \Sigma^* \to \Sigma^*$.

Again, this upper bound is tight, i.e., there exist associative, commutative, and length-monotonic polynomial-time computable operations whose generation problems are NP-complete. Even the usual addition on natural numbers has this property.

Theorem 6.14. $\mathrm{GEN_s}(+)$ *is* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-complete for* NP*, where* $+$ *is the addition on* $\mathbb{N}$.

Proof. It is known that $\mathrm{GEN_s}(+)$ is NP-complete for the addition on integers [vEB79]. This proof exclusively uses natural numbers. ❑

6.2 Generation Problems for Polynomials

The previous section gave an overview over the complexity of generation problems for polynomial-time computable operations. Now we want to have a look at the more restricted class of generation problems whose operations are polynomials. The Davis-Putnam-Robinson-Matiyasevich theorem [Mat70] states that every recursively enumerable set is range of a polynomial with integer coefficients. Based on this we find polynomials with multiple variables, where even the generation problem for single carriers is undecidable.

The idea is the following: Take a polynomial q with undecidable positive range and replace every variable x by $x_1^2 + x_2^2 + x_3^2 + x_4^2$. Take another polynomial r that is capable to generate all negative numbers and negative numbers only. Build a new polynomial out of q and r with an additional variable x_0 such that for $x_0 = 0$ the value of q is calculated, and for $x_0 \neq 0$ the value of r is calculated. In this way it is possible to generate all negative numbers which in turn allow the generation of the positive range of q. Let us make this precise.

Theorem 6.15. *There is a polynomial p with integer coefficients such that the problem $\mathrm{GEN_s}(p)$ is undecidable.*

Proof. Let $M \subseteq \mathbb{N}$ be enumerable but undecidable, such that $0, 1 \in M$, and let $q : \mathbb{Z}^n \to \mathbb{Z}$ such that $q(\mathbb{N}^n) \cap \mathbb{N} = M$. Without loss of generality we can assume, that $q(0, \ldots, 0) \geq 0$. Now, let

$$\begin{aligned}
\alpha &\stackrel{\text{def}}{=} (x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}, \ldots, x_{n,1}, x_{n,2}, x_{n,3}, x_{n,4}),\\
\beta &\stackrel{\text{def}}{=} (\sum_{i=1}^{4} x_{1,i}^2, \ldots, \sum_{i=1}^{4} x_{n,i}^2),\\
r(x_1, x_2, x_3, x_4) &\stackrel{\text{def}}{=} -1 \cdot \sum_{i=1}^{4} x_i^2 - 1,\\
a &\stackrel{\text{def}}{=} q(0, \ldots, 0)^2,\\
s(\alpha) &\stackrel{\text{def}}{=} \sum_{i=1}^{n} \sum_{j=1}^{4} x_{i,j}^2, \text{ and}\\
p'(\alpha) &\stackrel{\text{def}}{=} q(\beta)^2 - a + s(\alpha)a + 1.
\end{aligned}$$

Observe that the following holds for all $a_{1,1}, \ldots, a_{n,4} \in \mathbb{Z}$:

$$\begin{array}{rcl} [\{0,1\}]_r & = & -\mathbb{N} \cup \{1\}, \\ r(a_{1,1},\ldots,a_{1,4}) & < & 0, \\ s(a_{1,1},\ldots,a_{n,4}) & = & 0 \text{ if and only if } a_{1,1} = \cdots = a_{n,4} = 0, \\ p'(0,\ldots,0) & = & 1, \\ p'(a_{1,1},\ldots,a_{n,4}) & > & 0, \text{ and} \\ \text{if } s(a_{1,1},\ldots,a_{n,4}) > 0 & \text{then} & |p'(a_{1,1},\ldots,a_{n,4})| > |q(\beta)|. \end{array}$$

Now define for $\gamma \stackrel{\text{def}}{=} (x_0, y_1, \ldots, y_4, x_{1,1}, \ldots, x_{n,4})$

$$p(\gamma) \stackrel{\text{def}}{=} x_0^2 \cdot r(y_1,\ldots,y_4) \cdot p'(\alpha) + (1 - x_0^2) \cdot q(\beta)$$

and observe that the following holds:

$$p(\gamma) = \begin{cases} r(y_1,\ldots,y_4) & \text{, if } x_{1,1} = \cdots = x_{n,4} = 0, |x_0| = 1 \\ q(\beta) & \text{, if } x_0 = 0 \\ \leq 0 & \text{, otherwise} \end{cases}$$

Therefore we have $[\{0,1\}]_p \cap -\mathbb{N} = -\mathbb{N}$ and $[\{0,1\}]_p \cap \mathbb{N} = q(\mathbb{N}^n) \cap \mathbb{N} = M$. Therefore a positive number is in M if and only if it is in $[\{0,1\}]_p$. ❑

To obtain this undecidability result, the polynomials must have negative coefficients and they usually contain a rather large number of variables. Therefore, we concentrate on bivariate polynomials with positive coefficients. These are always length-monotonic and hence, the corresponding generation problem is decidable (see Theorem 6.5). We show that many of them are even in NP and, in the case of GEN_s, all of them belong to NTIME-SPACE($2^{\log^2 n}, n \log n$). In the case of GEN, all of them, except GEN($x + q(y)$), where q is a non-linear, univariate polynomial, belong to that class. So far we have no evidence against the conjecture that all these generation problems belong to NP (see also the discussion in Section 6.3). However, we cannot prove this.

This section has two main results: First, we show that if p is not of the form $q(x) + ky$ where q is non-linear and $k \geq 2$, then the corresponding generation problem belongs to NP. Second, we prove NP-completeness for polynomials of the form $x^a y^b c$ where $a, b, c \geq 1$.

6.2.1 The Main Case

Let us start our investigation with univariate polynomials p, i.e., $p(x, y) = q(x)$ for a suitable polynomial q.

Theorem 6.16. *If p is a univariate polynomial, then* GEN(p) *is in* P.

Proof. Every function in $[p]$ is essentially unary which means that they all have at most one relevant variable, since p is unary. So, if we get m carrier $a_{i,1}, \ldots, a_{i,n}, z_i$ $(1 \leq i \leq m)$ as input, either $z_i = a_{i,k}$ for a k and all $i \in \{1,\ldots,m\}$ or there is a k such that $z_i \in [a_{i,k}]_p$ for all $i \in \{1,\ldots,m\}$.

Hence, we go through all $j \in \{1, \ldots, n\}$ and test whether there is a k such that $p^k(a_{i,j}) = z_i$ for all $i \in \{1, \ldots, m\}$, which can be done in polynomial time: If $p(x, y) = q(x) = c$, then we have $[a_{i,j}]_p = \{a_{i,j}, c\}$. If $p(x, y) = q(x) = x + c$, then $[a_{i,j}]_p = \{a_{i,j} + kc : k \geq 0\}$. In all other cases we have $q(x) \geq 2x$ or $q(x) \geq x^2$. It follows that $[a_{i,j}]_p \subseteq \{p^k(a_{i,j}) : k = 0, 1, \ldots, |\mathrm{bin}(z_i)| + 1\}$. ❑

A univariate polynomial $p(x)$ is *linear*, if there are $a, c \in \mathbb{N}$ such that $p(x) = ax + c$.

Lemma 6.17. *Let p be a bivariate polynomial that is not of the form $p(x, y) = kx + q(y)$ or $p(x, y) = q(x) + ky$ where q is non-linear and $k \geq 2$. Then p must have one of the following properties:*

1. *$p(x, y) = x + q(y)$ or $p(x, y) = q(x) + y$ for some univariate polynomial q,*
2. *$p(x, y) = ax + by + c$ for some $a, b, c \in \mathbb{N}$ such that $a, b \geq 2$, or*
3. *$p(x, y) \geq x \cdot y$ for all x, y.*

Proof. Assume that the polynomial p has non of the properties *1.*, *2.*, and *3.*. Since p does not fulfill *3.* there are univariate polynomials q and r such that $p(x, y) = q(x) + r(y)$. Since $x^2 + y^2 \geq x \cdot y$ at least one of the polynomials q and r is linear. Consequently there exist a univariate polynomial q and an $k \geq 0$ such that $p(x, y) = kx + q(y)$ or $p(x, y) = q(x) + ky$. Since p does not fulfill *2.*, the polynomial q is not linear. Since p does not fulfill *1.*, we obtain $k \geq 2$. ❑

Lemma 6.18. *If $p(x, y) = x + q(y)$ for some univariate polynomial q, then $\mathrm{GEN_s}(p) \in \mathrm{NP}$.*

Proof. It is sufficient to prove:

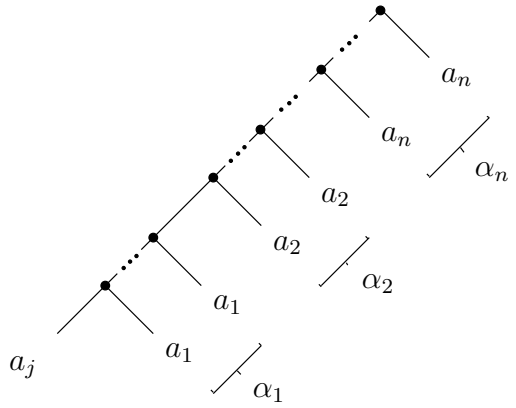
$$[a_1, \ldots, a_r]_p = \{a_j + \Sigma_{i=1}^r \alpha_i \cdot q(a_i) : j \in \{1, \ldots, r\} \text{ and } \alpha_1, \ldots, \alpha_r \in \mathbb{N}\}.$$

The inclusion from right to left is obvious. For the other direction, we observe that $\{a_1, \ldots, a_r\}$ is included in the right hand side (which is obvious) and that the right hand side is closed under p. For the latter let $\alpha_i, \beta_i \in \mathbb{N}$, let $s \stackrel{\text{def}}{=} \Sigma_{i=1}^r(\alpha_i \cdot q(a_i))$, and let $t \stackrel{\text{def}}{=} \Sigma_{i=1}^r(\beta_i \cdot q(a_i))$, for $1 \leq i \leq r$, and $j, k \in \{1, \ldots, r\}$. Then for some $c \geq 0$,

$$\begin{aligned} p(a_j + s, a_k + t) &= a_j + s + q(a_k + t) \\ &= a_j + s + q(a_k) + ct \\ &= a_j + \Sigma_{i=1}^r((\alpha_i + c\beta_i) \cdot q(a_i)) + q(a_k). \end{aligned} \tag{6.3}$$

To see equality (6.3), observe that by binomial theorem, for all $a, b \geq 0$, $q(a+b) = q(a) + cb$ for some $c \in \mathbb{N}$. ❑

So, we can check whether $z \in [a_1, \ldots, a_n]_p$ due to the fact, that for every p-generation tree (T, β) for z there is another generation tree (T', β') of the following form:



We can decide the generation problem for single carriers by guessing j and the α_i-values. However, this "uniform" tree T' describes another function than T, in general. Since the the α_i-values depend on $\{a_1, \ldots, a_n\}$, they may be different for different carriers. This is why we cannot use this method to decide GEN(p): For that, we need to find one generation tree that works for all given carriers. Such generation trees can have an exponential size. Nevertheless, we can decide GEN(p) with an alternating Turing-machine that uses $O(n)$ space, with an algorithm that is analogous to the one of the proof of Theorem 6.5. We will see later, that there is a polynomial q such that $\mathrm{GEN_s}(x + q(y))$ is NP-hard. Hence there remains a large gap between a known upper bound and a known lower bound of GEN($x + q(y)$); we have to leave the exact complexity of this problem as an open question.

Lemma 6.19. *If $p(x, y) = ax + by + c$ for $a, b, c \in \mathbb{N}$ and $a, b \geq 2$, then* GEN(p) $\in$ NP.

Proof. We show, that there is an encoding of polynomial size for p-generation trees that can be used to check in polynomial time whether an element can be generated from a given base via p.

Let T be a p-generation tree for z. Without loss of generality we can assume that value 0 occurs only in the leaves of this tree T. Since $a, b \geq 2$, the depth of T is bounded by $|\mathrm{bin}(z)| + 1$.

Let T be an arbitrary binary tree whose leaves have values from $\{a_1, \ldots, a_n\}$. For a full path q in T, choose $i(q) \in \{1, \ldots, n\}$ such that the leaf of q has value $a_{i(q)}$. We obtain that $z \in [a_1, \ldots, a_n]_p$ if and only if there exists a binary tree T whose leaves have values from $\{a_1, \ldots, a_n\}$ such that

$$z = \sum_{q \,\in\, \mathrm{fpath}(T)} a_{i(q)} \cdot a^{l(q)} \cdot b^{r(q)} + \sum_{q \,\in\, \mathrm{ipath}(T)} c \cdot a^{l(q)} \cdot b^{r(q)}.$$

For a binary tree T of depth bounded by d and for $i, j \in \{0, \ldots, d\}$ we define the characteristics

$$\begin{aligned} s^T_{i,j} &\stackrel{\mathrm{def}}{=} \#\{q : q \in \mathrm{ipath}(T), l(q) = i \text{ and } r(q) = j\} \text{ and} \\ r^T_{i,j} &\stackrel{\mathrm{def}}{=} \#\{q : q \in \mathrm{fpath}(T), l(q) = i \text{ and } r(q) = j\}. \end{aligned}$$

Note that the $r^T_{i,j}$ can be computed from the $s^T_{i,j}$ by

$$\left.\begin{array}{l} \bullet\ r^T_{0,0} = 1 - s^T_{0,0}, \\ \bullet\ r^T_{0,j+1} = s^T_{0,j} - s^T_{0,j+1} \text{ for } j \in \{0,\ldots,d\}, \\ \bullet\ r^T_{i+1,0} = s^T_{i,0} - s^T_{i+1,0} \text{ for } i \in \{0,\ldots,d\}, \text{ and} \\ \bullet\ r^T_{i+1,j+1} = s^T_{i,j+1} + s^T_{i+1,j} - s^T_{i+1,j+1} \text{ for } i,j \in \{0,\ldots,d\}. \end{array}\right\} \tag{6.4}$$

Using these characteristics we obtain that $z \in [a_1,\ldots,a_n]_p$ if and only if there exist a binary tree T of depth $d \leq |\mathrm{bin}(z)| + 1$ and a set of natural numbers $\{r_{i,j,k} : i,j \in \{0,\ldots,d\}, k \in \{1,\ldots,n\}\}$ such that $\Sigma^n_{k=1} r_{i,j,k} = r^T_{i,j}$ and

$$z = \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=1}^{n} r_{i,j,k}\cdot a_k \cdot a^i \cdot b^j + \sum_{i=0}^{d}\sum_{j=0}^{d} s^t_{i,j}\cdot c \cdot a^i \cdot b^j.$$

Observe that the characteristics $s^T_{i,j}$ have the following properties.

$$\left.\begin{array}{l} \bullet\ s^T_{0,0} \leq 1, \\ \bullet\ s^T_{0,j+1} \leq s^T_{0,j} \text{ for } j \in \{0,\ldots,d-1\}, \\ \bullet\ s^T_{i+1,0} \leq s^T_{i,0} \text{ for } i \in \{0,\ldots,d-1\}, \\ \bullet\ s^T_{i+1,j+1} \leq s^T_{i+1,j} + s^T_{i,j+1}, \text{ for } i,j \in \{0,\ldots,d-1\}, \text{ and} \\ \bullet\ s^T_{i,d} = s^T_{d,j} = 0 \text{ for } i,j \in \{0,\ldots,d\}. \end{array}\right\} \tag{6.5}$$

On the other hand, we can prove the following.

Claim 6.20. Consider an arbitrary set M of natural numbers $s_{i,j}$ where $i,j \in \{0,\ldots,d\}$. If these $s_{i,j}$ fulfill (6.5), then there exists a binary tree T such that $s^T_{i,j} = s_{i,j}$ for $i,j \in \{0,\ldots,d\}$.

Proof: We proof the claim by induction on $w(M) \stackrel{\text{def}}{=} \Sigma^d_{i=0}\Sigma^d_{j=0} s_{i,j}$.
If $w(M) = 0$, then the tree with only one node fulfills the statement.
If $w(M) > 0$, then we have $s_{0,0} > 0$. Since $s_{i,d} = s_{d,j} = 0$ for $i,j \in \{0,\ldots,d\}$ there exists a pair $(i,j) \in \{0,\ldots,d\}^2$ such that $s_{i,j} > 0$ and $s_{i+1,j} = s_{i,j+1} = 0$. Let (i_0,j_0) be such a pair. Define $M' \stackrel{\text{def}}{=} \{s'_{i,j} : i,j \in \{0,\ldots,d\}\}$ such that $s'_{i_0,j_0} \stackrel{\text{def}}{=} s_{i_0,j_0} - 1$ and $s'_{i,j} = s_{i,j}$ for all other $(i,j) \in \{0,\ldots,d\}^2$. Obviously, M' fulfills (6.5) and $w(M') = w(M) - 1$. By the induction hypothesis, there exists a binary tree T' such that $s^{T'}_{i,j} = s'_{i,j}$ for $i,j \in \{0,\ldots,d\}$. To know that there exists a full path q in T' such that $l(q) = i_0$ and $r(q) = j_0$ we have to prove $r^{T'}_{i_0,j_0} > 0$. We do this by considering four cases.

If $i_0 = j_0 = 0$ then $s^{T'}_{0,0} = s'_{0,0} < s_{0,0} \leq 1$ and hence $s^{T'}_{0,0} = 0$.
If $i_0 = 0$ and $j_0 > 0$ then $s^{T'}_{0,j_0} = s'_{0,j_0} < s_{0,j_0} \leq s_{0,j_0-1} = s'_{0,j_0-1} = s^{T'}_{0,j_0-1}$.
If $i_0 > 0$ and $j_0 = 0$ then $s^{T'}_{i_0,0} = s'_{i_0,0} < s_{i_0,0} \leq s_{i_0-1,0} = s'_{i_0-1,0} = s^{T'}_{i_0-1,0}$.

If $i_0 > 0$ and $j_0 > 0$ then $s^{T'}_{i_0,j_0} = s'_{i_0,j_0} < s_{i_0,j_0} \leq s_{i_0-1,j_0} + s_{i_0,j_0-1} = s'_{i_0-1,j_0} + s'_{i_0,j_0-1} = s^{T'}_{i_0-1,j_0} + s^{T'}_{i_0,j_0-1}$.

Now choose a full path q in T' such that $l(q) = i_0$ and $r(q) = j_0$ and attach two successors to it. For the binary tree T defined in such a way, we have $s^T_{i_0,j_0} = s^{T'}_{i_0,j_0} + 1 = s'_{i_0,j_0} + 1 = s_{i_0,j_0}$ and $s^T_{i,j} = s^{T'}_{i,j} = s'_{i,j} = s_{i,j}$ for all other $(i,j) \in \{0,\ldots,d\}^2$. This completes the proof of the claim. □

Together with Proposition 6.2, we obtain that $(m, a_{1,1}, \ldots, a_{1,n}, \ldots, a_{m,1}, \ldots, a_{m,n}, z_1, \ldots, z_m) \in \mathrm{GEN}(p)$ if and only if for $d \stackrel{\mathrm{def}}{=} |\mathrm{bin}(z_1)| + 1$ and $i, j \in \{0,\ldots,d\}$ there exist natural numbers $s_{i,j}$ and there exists a set of natural numbers $\{r_{i,j,k} : i,j \in \{0,\ldots,d\}, k \in \{1,\ldots,n\}\}$ such that

1. the $s_{i,j}$ fulfill (6.5),
2. $\Sigma^n_{k=1} r_{i,j,k} = r_{i,j}$ for $i,j \in \{0,\ldots,d\}$ (where the $r_{i,j}$ are computed from the $s_{i,j}$ as in (6.4), and
3. $z_\ell = \Sigma^d_{i=0}\Sigma^d_{j=0}(\Sigma^n_{k=1} r_{i,j,k} \cdot a_{\ell,k}) \cdot a^i \cdot b^j + \Sigma^d_{i=0}\Sigma^d_{j=0} s_{i,j} \cdot c \cdot a^i \cdot b^j$ for all $\ell \in \{1,\ldots,m\}$.

The properties 1. – 3. can be checked deterministically in polynomial time, hence $\mathrm{GEN}(p) \in \mathrm{NP}$. ❑

The next two lemmas show that $\mathrm{GEN}(p) \in \mathrm{NP}$, if $p(x,y) \geq x \cdot y$ for all $x, y \in \mathbb{N}$. Although Lemma 6.21 follows from Lemma 6.23, we prefer to first explicitly prove the former one, as we can use the construction, that we introduce there, in the proof of the latter one.

Lemma 6.21. *If the polynomial p fulfills $p(x,y) \geq x \cdot y$ for all x, y, then $\mathrm{GEN_s}(p) \in \mathrm{NP}$.*

Proof. Let $A \subseteq \mathbb{N}$ be finite. Let

$$A' \stackrel{\mathrm{def}}{=} \begin{cases} A & \text{, if } c \notin A \\ A \cup \{p(0,0)\} & \text{otherwise.} \end{cases}$$

Obviously we have $[A]_p = [A']_p$ and for every $z \in [A']_p$ there is generation tree where no node has two children with value 0. If for every $x \in \mathbb{N}$ (resp., $y \in \mathbb{N}$),

$$p(x,0) \geq 2x \text{ (resp., } p(0,y) \geq 2y) \text{ or } p(x,0) \geq x^2 \text{ (resp., } p(0,y) \geq y^2) \text{ or}$$
$$p(x,1) \geq 2x \text{ (resp., } p(1,y) \geq 2y) \text{ or } p(x,1) \geq x^2 \text{ (resp., } p(1,y) \geq y^2),$$

then there is a p-generation tree for z from A' such that there are at most $|z|$ nodes with left (resp., right) child that has a value ≥ 2 $(*)$. Let D be a p-generation tree from A' for z. We can assume that there are at most $|z|$ leaves v in D that have a value greater than 1 and there can at most be $|z|$ nodes having two children with values greater than 1. Furthermore, we can assume that there are at most $|z|$ nodes v in D such that both children of v are leaves with value from $\{0,1\}$ since the value of theses nodes would be

greater or equal to 2 (if the value of such a node were 0 or 1, the node would not be necessary). That means that if D has more than polynomially many nodes, then nearly every node (except polynomially many ones)

$$\left.\begin{array}{l}\bullet \text{ has one child with value } \leq 1 \text{ and another one that is no leaf, or}\\ \bullet \text{ is a leaf with value } \leq 1, \text{ and its parent's other child is no leaf.}\end{array}\right\} \tag{6.6}$$

We consider four cases:

1. Let there be $x_1, \ldots, x_8 \in \mathbb{N}$ such that

$$\begin{array}{l}(p(x_1,0) \not\geq 2x_1 \text{ and } p(x_2,0) \not\geq x_2^2 \text{ and } p(x_3,1) \not\geq 2x_3 \text{ and } p(x_4,1) \not\geq x_4^2) \text{ and}\\ (p(0,x_5) \not\geq 2x_5 \text{ and } p(0,x_6) \not\geq x_6^2 \text{ and } p(1,x_7) \not\geq 2x_7 \text{ and } p(1,x_8) \not\geq x_8^2).\end{array}$$

   Then $p(x,y) = xy + d$, where $d \in \mathbb{N}$. Note, that $p(0,y) = p(x,0) = d$. Since $d \in A'$ if $0 \in A'$, we can assume, that every generation tree has no node with value 0. Observe, that $q(x) \stackrel{\text{def}}{=} x + d = p(1,x) = p(x,1)$ for all $x \in \mathbb{N}$. Note that $q^k(x) = x + kd$, so k applications of q can be guessed in one step. Using property (6.6), we can guess a polynomially sized generation tree, where each node either represents a normal generation step or $k \leq z$ steps of the above form.
2. Let there be $x_1, \ldots, x_4 \in \mathbb{N}$ such that for all $x \in \mathbb{N}$ we have

$$\begin{array}{l}(p(x,0) \geq 2x \quad \text{or} \quad p(x,0) \geq x^2 \quad \text{or} \quad p(x,1) \geq 2x \quad \text{or} \quad p(x,1) \geq x^2) \text{ and}\\ (p(0,x_1) \not\geq 2x_1 \text{ and } p(0,x_2) \not\geq x_2^2 \text{ and } p(1,x_3) \not\geq 2x_3 \text{ and } p(1,x_4) \not\geq x_4^2).\end{array}$$

   Then $p(x,y) = x^k y + \Sigma_{i=1}^n b_i x^i + d$ where $k \geq 1$, $n, b_i, d \in \mathbb{N}$ $(1 \leq i \leq n)$. Because of $(*)$ there can only be polynomially many nodes in D with a left child that has a value greater than 1. So, if there are more than polynomially many nodes in D, then all of them except polynomially many ones have a left child with value ≤ 1 and a right child that is not a leaf. Observe, that $p(0,y) = d$. So, if $0 \in A'$ then $d \in A'$, too, and we can assume that there are no nodes that have a left child with value 0. Note that $r(y) \stackrel{\text{def}}{=} p(1,y) = y + \Sigma_{i=1}^n b_i + d$ and $r^k(y) = y + k(\Sigma_{i=1}^n b_i + d)$. Therefore we can guess a polynomial-sized generation tree for z where each node is either a normal generation step or $k \leq z$ subsumed steps of the form $p(1,y)$.
3. Let there be $x_1, \ldots, x_4 \in \mathbb{N}$ such that for all $x \in \mathbb{N}$ we have

$$\begin{array}{l}(p(x_1,0) \not\geq 2x_1 \text{ and } p(x_2,0) \not\geq x_2^2 \text{ and } p(x_3,1) \not\geq 2x_3 \text{ and } p(x_4,1) \not\geq x^2) \text{ and}\\ (p(0,x) \geq 2x \quad \text{or} \quad p(0,x) \geq x^2 \quad \text{or} \quad p(1,x) \geq 2x \quad \text{or} \quad p(1,x) \geq x^2).\end{array}$$

   Here a symmetrical argumentation holds.
4. Let for all $x \in \mathbb{N}$ hold

$$(p(x,0) \geq 2x \text{ or } p(x,0) \geq x^2 \text{ or } p(x,1) \geq 2x \text{ or } p(x,1) \geq x^2) \text{ and}$$
$$(p(0,x) \geq 2x \text{ or } p(0,x) \geq x^2 \text{ or } p(1,x) \geq 2x \text{ or } p(1,x) \geq x^2).$$

By $(*)$ there is a polynomial sized p-generation tree from A' for b that can be guessed and checked in P.

❑

Corollary 6.22. *If p is a bivariate polynomial that is not of the form $p(x,y) = kx + q(y)$ or $p(x,y) = q(x) + ky$ where q is non-linear and $k \geq 2$, then* $\mathrm{GEN_s}(p) \in \mathrm{NP}$.

With polynomials p such that $p(x,y) \geq xy$, there is a main problem: p is not monotonic at every point, since $p(x,0) = p(0,x) = d$ for some constant d, regardless of the size of x. In the last lemma, we coped with this problem as follows: We eliminated nearly all occurrences of 0 from the generation tree and replaced them with the value of $p(0,0)$.

However, if we have to find a generation tree for multiple carriers, we cannot simply eliminate occurrences of 0, since a leaf that has the value 0 for one carrier does not have to have the same value for another one. The idea to solve this problem is the following. We guess an encoding for every carrier as in the previous lemma and mark every leaf v that is a replacement for an inner node as described above. Then we compare all these trees and test whether they all share the same "core", i.e., whether they have the same structure between the root and marked leaves.

Lemma 6.23. *If the polynomial p fulfills $p(x,y) \geq x \cdot y$ for all x, y, then* $\mathrm{GEN}(p) \in \mathrm{NP}$.

Proof. We have seen in the proof of Lemma 6.21 that we can always guess encodings of p-generation trees of polynomial size. We showed, that such a generation tree has superpolynomial size only in chains of nodes v that have children w_1, w_2 such that w_1 is a leaf (the *leaf-nodes of the chain*) with value 1 and w_2 is not a leaf, i.e., w_2 is a node of the same form as v, again. Hence, we can guess generation-trees $(D_1, \zeta_1), \ldots, (D_m, \zeta_m)$ of polynomial size, and, additionally, functions $f_1, \ldots, f_m$ such that f_i maps some nodes of D_i to $\{1, \ldots, \min\{z_1, \ldots, z_m\}\}$. We do this with those nodes that represent a chain of superpolynomial size. These nodes then have to have just one child and they stand for a chain of length $f_i(v)$. What remains to do is to show, that it is easy to test whether these generation trees are "compatible", i.e., that we can merge them to one generation tree for all z_i.

For this, consider an instance $(m, a_{1,1}, \ldots, a_{1,n}, \ldots, a_{m,1}, \ldots, a_{m,n}, z_1, \ldots, z_m)$ of the generation problem. An *index-tree* is a triple (D, ζ, f) such that the following holds:

- D is a tree where every node has fan-out ≤ 2. All parents of leaves have fan-out 2. If p is a polynomial as in Lemma 6.21, case 4, then D has no nodes of fan-out 1.
- For every leaf in D with path $w \in \{l,r\}^*$ there exists exactly one $u \in \{l,r\}^*$ such that $w = uv$ for some $v \in \{l,r\}^*$, u describes a node with fan-out 2, and $\zeta(u) \in \{0, \ldots, n\}$. If $u \neq w$, then $\zeta(u) = 0$. For all other $w' \in \{l,r\}^*$ holds $\zeta(w') = \perp$.
- For all inner nodes v of D that have fan-out 1, is $f(v) \in \{1, \ldots, \min\{z_1, \ldots, z_m\}\}$.

(D, ζ, f) is called *a full index-tree*, if for all leaves v in D that have path w, we have $\zeta(w) \neq \perp$. The full index-trees $T \stackrel{\text{def}}{=} (D, \zeta, f)$ and $T' \stackrel{\text{def}}{=} (D', \zeta', f')$ *match* if

1. For every leaf v in D that is characterized by the word $w \in \{l, r\}^*$, either
   – there is a leaf v' in D' that is characterized by w and $\zeta(w) = \zeta'(w)$ or
   – $\zeta(w) = 0$ and there is a node v' in D' such that w is an initial path in D' characterizing v' (in this case we say T *prunes* w in T'),
   and vice versa.
2. If $w \in \{l, r\}^*$ is a path describing inner nodes v from D and v' from D', then $f(v) = f'(v')$.

We can test in P whether or not two full index-trees T and T' match. Let

$$q(x) \stackrel{\text{def}}{=} \begin{cases} p(x, 1) & \text{, if } p \text{ is as in Lemma 6.21, cases 1 or 2} \\ p(1, x) & \text{, if } p \text{ is as in Lemma 6.21, case 3} \end{cases}$$

For $i \in \{1, \ldots, m\}$, an index-tree $T = (D, \zeta, f)$, and a node v from D that is characterized by $w \in \{l, r\}^*$ we define

$$\alpha_i^T(v) \stackrel{\text{def}}{=} \begin{cases} a_{i,\zeta(w)} & \text{, if } v \text{ is a leaf in } D \text{ and } \zeta(w) \notin \{0, \perp\} \\ b_{0,0} & \text{, if } \zeta(w) = 0 \text{ and } 0 \in \{a_{i,1}, \ldots, a_{i,n}\} \\ q^k(\alpha_i^T(v_1)) & \text{, if } f(v) = k \text{ and } v_1 \text{ is the child of } v \\ p(\alpha_i^T(v_1), \alpha_i^T(v_2)) & \text{, if } v \text{ is a node in } D \text{ with left child } v_1 \\ & \quad \text{and right child } v_2 \end{cases}$$

Observe, that $\alpha_i^T(\text{Root}(D))$ is undefined, only if there is a node with path w in D such that $\zeta(w) = 0$ and $0 \notin \{a_{i,1}, \ldots, a_{i,n}\}$. We say, the index-tree T *generates* z_i if $\alpha_i^T(\text{Root}(D))$ is defined and $\alpha_i^T(\text{Root}(D)) = z_i$.

Let $T_1 \stackrel{\text{def}}{=} (D_1, \zeta_1, f_1), \ldots, T_m \stackrel{\text{def}}{=} (D_m, \zeta_m, f_m)$ be full index-trees that generate $z_1, \ldots, z_m$ respectively. We say $T_1, , \ldots, T_m$ are *functionally equivalent*, if they match pairwise and the following holds:

(i) Let $i, j \in \{1, \ldots, m\}$. For every path w such that w characterizes a node v with children v_1, v_2 in D_j and T_i prunes w in T_j holds $\{\zeta_j(v_1), \zeta_j(v_2)\} \cap \{k : a_{i,k} = 0\} \neq \emptyset$ and vice versa.

(ii) Let $A \subseteq \{1, \ldots, m\}$. If there is a path w such that there are nodes v_j in D_j that are characterized by w such that $\zeta_j(v_j) = 0$ for all $j \in A$, then there are $s, t \in \{1, \ldots, n\}$ such that $a_{j,s} = 0$ or $a_{j,t} = 0$ for all $j \in A$ (we explicitly allow $s = t$).

We can test, whether $T_1, \ldots, T_m$ are functionally equivalent in P. Hence, we can finish the proof with the following claim.

Claim 6.24. $C \stackrel{\text{def}}{=} (m, a_{1,1}, \ldots, a_{1,n}, \ldots, a_{m,1}, \ldots, a_{m,n}, z_1, \ldots, z_m) \in \text{GEN}(p)$ if and only if there are index-trees, $T_1 \stackrel{\text{def}}{=} (D_1, \zeta_1, f_1), \ldots, T_m \stackrel{\text{def}}{=} (D_m, \zeta_m, f_m)$ which have size less or equal than $\sum_{i=1}^m |z_i|$ such that for all $i \in \{1, \ldots, m\}$ holds T_i generates z_i and $T_1, \ldots, T_m$ are functionally equivalent.

Proof: First, let $C \in \mathrm{GEN}(p)$. Then there is a binary tree D and α_i such that (D, α_i) is a p-generation tree for the i-th carrier, as described in Proposition 6.2 for all $i \in \{1, \ldots, m\}$. As shown in Lemma 6.21, we can collapse the tree to linear size, by subsuming the long chains to one node. We build the index-tree T_i by removing the subtrees of nodes v such that one of the following conditions holds:

– v has two children with value 0.
– p is as in Lemma 6.21, case 2, and v has a left child with value 0.
– p is as in Lemma 6.21, case 3, and v has a right child with value 0.

We set $\zeta_i(v) \stackrel{\text{def}}{=} 0$, and for all other leaves w, we set $\zeta_i(w) \stackrel{\text{def}}{=} k$, if $\alpha_i(w) = a_{i,k}$. Observe that T_i generates z_i and that the thus constructed index-trees are functionally equivalent.

On the other hand, let $T_1, \ldots, T_m$ be as claimed. We can build a p-generation tree for all carriers. For that, we first merge T_1 and T_2 into a new index-tree $T' \stackrel{\text{def}}{=} (D', \zeta', f')$ as follows: We start with T_1. If T_1 prunes w in T_2 then we replace the node v with path w in D_1 with the whole subtree under the node with path w of D_2. For all leaves u of such a subtree, we set $\zeta'(r) \stackrel{\text{def}}{=} \zeta_2(r)$ where r is the path characterizing u. For all other paths q to a leaf, we set $\zeta'(q) \stackrel{\text{def}}{=} \zeta_1(q)$. For nodes v of fan out 1 of such a subtree, we set $f'(v) \stackrel{\text{def}}{=} f_2(v)$, for all other nodes v' of fan-out 1, we set $f'(v') \stackrel{\text{def}}{=} f_1(v')$. Observe, that T' generates both, z_1 and z_2. We merge T' with T_3, and so on, until all index-trees are merged into a final index-tree $\widehat{T} \stackrel{\text{def}}{=} (\widehat{D}, \widehat{\zeta}, \widehat{f})$. We build an index-tree $T \stackrel{\text{def}}{=} (D, \zeta, f)$ from $\widehat{T}$ as follows: If v is the path of a leaf in $\widehat{D}$ such that $\zeta_i(v) = 0$ for all i in a set $A \subseteq \{1, \ldots, m\}$, then there have to exist s, t as in (ii). We add two children with paths vl, vr to the leaf and set $\zeta(vl) \stackrel{\text{def}}{=} s$ and $\zeta(vr) \stackrel{\text{def}}{=} t$. Then for all $i \in \{1, \ldots, m\}$, the tuple (D, α_i^T) is a p-generation tree from $a_{i,1}, \ldots, a_{i,n}$ for z_i. □ ❑

Corollary 6.25. *If p is a bivariate polynomial that is not of the form $p(x, y) = kx + q(y)$ or $p(x, y) = q(x) + ky$ where q is non-linear and $k \geq 1$, then* $\mathrm{GEN}(p) \in \mathrm{NP}$.

6.2.2 $\mathrm{GEN_s}(x^a y^b c)$ is NP-complete

By Corollary 6.25, generation problems for polynomials of the form $\Sigma_{a,b} c_{a,b} x^a y^b$ where $a, b \geq 1$ belong to NP. In this subsection, we concentrate on polynomials that consist of only one term of that sum. For this special case we can show that even $\mathrm{GEN_s}(x^a y^b c)$ is NP-complete if $c \geq 1$. For $a = 1$ or $b = 1$ this is easy to prove with a reduction from the following problem:

Definition 6.26.

$$\text{1-IN-3-SAT} \stackrel{\text{def}}{=} \{H : H \textit{ is a } 3\text{-CNF } \textit{formula having an assignment that satisfies exactly one literal in each clause}\}$$

Theorem 6.27 ([Sch78]). 1-IN-3-SAT *is* NP*-complete.*

Proposition 6.28. *Let $p(x,y) = x^a \cdot y \cdot c$ where $a, c \geq 1$ are constants. Then* $\mathrm{GEN_s}(p)$ *is* $\leq^{\mathrm{p}}_{\mathrm{m}}$*-complete for* NP.

Proof. We reduce 1-IN-3-SAT to $\mathrm{GEN_s}(p)$. Let H be a 3-CNF formula with clauses $C_1, \ldots, C_m$ and variables $x_1, \ldots, x_n$. Let $p_1, p_2, \ldots$ be the prime numbers larger than c. Define for $2 \leq i \leq n$:

$$\begin{aligned}
a_1 &\stackrel{\text{def}}{=} (p_{m+1} \prod_{x_1 \in C_j} p_j)^a, \\
b_1 &\stackrel{\text{def}}{=} (p_{m+1} \prod_{\overline{x_1} \in C_j} p_j)^a, \\
a_i &\stackrel{\text{def}}{=} p_{m+i} \prod_{x_i \in C_j} p_j, \\
b_i &\stackrel{\text{def}}{=} p_{m+i} \prod_{\overline{x_i} \in C_j} p_j, \\
z &\stackrel{\text{def}}{=} c^{n-1} \prod_{i=1}^{m+n} p_i^a, \text{ and} \\
g(H) &\stackrel{\text{def}}{=} (a_1, \ldots, a_n, b_1, \ldots, b_n, z).
\end{aligned}$$

Note that g is polynomial-time computable.

Assume $H \in$ 1-IN-3-SAT. Then there is an assignment $I : \{x_1, \ldots, x_n\} \to \{0,1\}$ that satisfies exactly one literal in each clause. Therefore, we obtain $\prod_{i=1}^{m+n} p_i^a \cdot c^{n-1}$ by a linear generation tree that has leaf-values $c_n, \ldots, c_1$ where $c_i = a_i$ if $I(x_i) = 1$ and $c_i = b_i$ otherwise. Hence $g(H) \in \mathrm{GEN_s}(p)$.

Assume that $g(H) \in \mathrm{GEN_s}(p)$, hence $z \in [a_1, \ldots, a_n, b_1, \ldots, b_n]_p$. Every prime p_i occurs exactly a times in the factorization of z. Therefore, either a_i or b_i (and not both) has to be a leaf-value in the generation tree. If $a > 1$ then additionally the generation tree has to be linear and the rightmost leaf has value a_1 or b_1. If we can build a generation tree for z that contains each prime for a variable and each prime for a clause exactly a times it is possible to find an assignment that satisfies exactly one literal in each clause. Hence, the assignment I such that $I(x_i) = 1$ if and only if a_i is a leaf-value in the generation tree satisfies H in the sense of 1-IN-3-SAT. Therefore $H \in$ 1-IN-3-SAT. ❑

Now let us consider $\mathrm{GEN_s}(x^a y^b c)$ for $a, b > 1$. In general, the main problem in proving hardness for generation problems seems to be that various different trees can generate the same number. In our proofs we force the generation trees to have a specific shape such that the generation is possible only in a predefined way.

Consider an $x^a y^b c$-generation tree. Clearly, the generated number is a product that consists of various multiplicities of c and base elements. As a tool to control these multiplicities we introduce (a,b)-*weighted trees* where we mark each node as follows. If ℓ is the number of left turns on the way from the root to a node, and r is the number of respective

right turns, then we mark the node with a^ℓ and b^r. By controlling the marks of the leaves, we can force an $x^a y^b c$-generation tree into the shape of a complete (a,b)-weighted tree.

Definition 6.29. *Let D be a binary tree and let $a, b > 1$. $T = (D, g)$ is called* (a,b)-weighted tree *if g is a function that maps the nodes of D to natural numbers such that:*

If $v = \mathrm{Root}(D)$, then $g(v) = 1$.
If $v \in \mathrm{Node}(D)$ has a left and a right successor v_l and v_r, then $g(v_l) = a \cdot g(v)$ and $g(v_r) = b \cdot g(v)$.

T is called balanced *if* $\max\{g(v) : v \in \mathrm{Leaf}(D)\} \leq \max\{a,b\} \cdot \min\{g(v) : v \in \mathrm{Leaf}(D)\}$.
T is called complete *if* $\max\{g(v) : v \in \mathrm{Leaf}(D)\} < \max\{a,b\} \cdot \min\{g(v) : v \in \mathrm{Leaf}(D)\}$.

We immediately obtain the following connection to $\mathrm{GEN_s}(x^a y^b c)$.

Proposition 6.30. *Let $a, b > 1$. If $T = (D, g)$ is an (a,b)-weighted tree, and (D, α) is an $x^a y^b c$-generation tree for z, then*

$$z = \alpha(\mathrm{Root}(D)) = \prod_{v \in \mathrm{Leaf}(D)} \alpha(v)^{g(v)} \cdot \prod_{v \in \mathrm{Node}(D) - \mathrm{Leaf}(D)} c^{g(v)}.$$

We remark that it is possible to define the notion of (a,b)-weighted trees for $a = 1$ and $b = 1$. However, if $a = 1$ and $b = 1$, then complete trees do not exist in contrast to $a, b > 1$ where complete trees always exist. We show some important properties of (a,b)-weighted trees.

Proposition 6.31. *Let $a, b > 1$. For every $n \geq 1$ there exists a balanced (a,b)-weighted tree that has n leaves.*

Proof. For $n = 1$ take the tree that consists only of the root.
For arbitrary $n > 1$, let $T = (D, g)$ be a balanced (a,b)-weighted tree with $n - 1$ leaves. Let $v_0 \in \mathrm{Leaf}(D)$ have minimal weight, i.e., $g(v_0) = \min\{g(v) : v \in \mathrm{Leaf}(D)\}$. Define the tree D' by adding in D children v_l and v_r to v_0, and let $g'(v) \stackrel{\mathrm{def}}{=} g(v)$ for all $v \in \mathrm{Node}(D)$, $g'(v_l) \stackrel{\mathrm{def}}{=} a \cdot g(v_0)$, and $g'(v_r) \stackrel{\mathrm{def}}{=} b \cdot g(v_0)$. This defines an (a,b)-weighted tree $T' \stackrel{\mathrm{def}}{=} (D', g')$ with

$$\begin{aligned}
\max\{v : v \in \mathrm{Leaf}(D')\} &= \max\{\max\{g(v) : v \in \mathrm{Leaf}(D)\}, \max\{g'(v_l), g'(v_r)\}\} \\
&= \max\{\max\{g(v) : v \in \mathrm{Leaf}(D)\}, \max\{a,b\} \cdot g(v_0)\} \\
&= \max\{\max\{g(v) : v \in \mathrm{Leaf}(D)\}, \\
&\qquad \max\{a,b\} \cdot \min\{g(v) : v \in \mathrm{Leaf}(D)\}\} \\
&= \max\{a,b\} \cdot \min\{g(v) : v \in \mathrm{Leaf}(D)\} \\
&\leq \max\{a,b\} \cdot \min\{g(v) : v \in \mathrm{Leaf}(D')\}. \qquad (6.7)
\end{aligned}$$

Hence T' is balanced. ❑

Now we show that for each $n \geq 1$ there exists a complete (a, b)-weighted tree with nearly n leaves. Note that such a tree is polynomial-time constructible.

Proposition 6.32. *Let $a, b > 1$. For every $n \geq 1$ there exists a complete (a, b)-weighted tree with at least n and at most $2n - 1$ leaves.*

Proof. Proposition 6.31 gives a balanced (a, b)-weighted tree T with n leaves. If all leaves have minimal weight, then T is complete. Otherwise, there are k leaves of minimal weight m ($1 \leq k \leq n-1$). Since T is balanced, the maximal weight is less or equal to $m \cdot \max\{a, b\}$. If we add two successors to each of these leaves, then the minimal weight increases, while the maximum weight remains less or equal to $m \cdot \max\{a, b\}$. So in inequality (6.7), $\leq$ changes to $<$ and the resulting tree T' is complete. T' has $n - k + 2k = n + k$ leaves where $n \leq n + k \leq 2n - 1$. ❑

Now we show that if the generation tree is not the desired complete tree, then at least one leaf-value is taken to a power that is too large.

Proposition 6.33. *Let $a, b > 1$. Let $T = (D, g)$ be a complete (a, b)-weighted tree with n leaves. If $T' = (D', g')$ is an (a, b)-weighted tree with more than n leaves, then there exists a $v \in \mathrm{Leaf}(D')$ such that*

$$g'(v) > \max\{g(u) : u \in \mathrm{Leaf}(D)\}.$$

Proof. Without loss of generality we can assume $a \geq b$. Fix a shortest way in terms of deleting and adding leaves that transforms D to D'. We have to change at least one leaf $v_0 \in \mathrm{Leaf}(D)$ to an inner node of D'. Let v_l and v_r be the children of v_0. We obtain

$$\begin{aligned} g'(v_l) &= a \cdot g(v_0) \\ &\geq \max\{a, b\} \cdot \min\{g(u) : u \in \mathrm{Leaf}(D)\} \\ &> \max\{g(u) : u \in \mathrm{Leaf}(D)\}. \end{aligned}$$

Hence, every $v \in D'$ that for which v_l is an ancestor fulfills

$$g'(v) \geq g'(v_l) > \max\{g(u) : u \in \mathrm{Leaf}(D)\}.$$

❑

Next we show that balanced (a, b)-weighted trees have a height which is bounded logarithmically in the number of leaves.

Proposition 6.34. *Let $a \geq b > 1$. Let $T = (D, g)$ be a balanced (a, b)-weighted tree with n leaves. If d denotes the maximal depth of a leaf of D, then*

$$d \leq \log_b(a) \cdot (1 + \log_2 n)$$

Proof. Let $m \stackrel{\text{def}}{=} \min\{g(v) : v \in \text{Leaf}(D)\}$. Hence, D contains a complete binary tree of depth $\geq \log_a m$ and therefore $\log_2 n \geq \log_a m$. T is balanced, so $b^d \leq am$ which is equivalent to $d \leq \log_b am$. Therefore,

$$d \leq \log_b am = \log_b a \cdot \log_a am \leq \log_b a \cdot (1 + \log_2 n).$$

❑

Theorem 6.35. *For $a, b, c \geq 1$ and $p(x, y) \stackrel{\text{def}}{=} x^a y^b c$, $\text{GEN}_{\text{s}}(p)$ is $\leq^{\text{p}}_{\text{m}}$-complete for* NP.

Proof. By Proposition 6.28, we can assume $a, b > 1$. Containment in NP follows from Corollary 6.22. We reduce 1-IN-3-SAT to $\text{GEN}_{\text{s}}(p)$. Let H be a 3-CNF formula with clauses $C_1, \ldots, C_m$ and variables $x_1, \ldots, x_n$. Let $p_1, p_2, \ldots$ be the prime numbers larger than c. Define for $1 \leq i \leq n$,

$$\begin{aligned} a_i &\stackrel{\text{def}}{=} p_{m+i} \prod_{x_i \in C_j} p_j, \text{ and} \\ b_i &\stackrel{\text{def}}{=} p_{m+i} \prod_{\overline{x_i} \in C_j} p_j. \end{aligned}$$

Let $T = (D, g)$ be a complete (a, b)-weighted tree with k leaves where $n \leq k \leq 2n - 1$ and $\text{Leaf}(D) = \{v_1, \ldots, v_k\}$ (such a tree exists by Proposition 6.32). Furthermore, let d be the maximal depth of a leaf of D. Define $a_i \stackrel{\text{def}}{=} p_{m+i}$ for $i = n + 1, \ldots, k$,

$$\begin{aligned} B &\stackrel{\text{def}}{=} \left\{ a'_i \stackrel{\text{def}}{=} a_i^{a^d b^d / g(v_i)} : 1 \leq i \leq k \right\} \cup \left\{ b'_i \stackrel{\text{def}}{=} b_i^{a^d b^d / g(v_i)} : 1 \leq i \leq n \right\}, \text{ and} \\ z &\stackrel{\text{def}}{=} \prod_{i=1}^{m+k} p_i^{a^d b^d} \cdot \prod_{v \in \text{Node}(D) - \text{Leaf}(D)} c^{g(v)}. \end{aligned}$$

Proposition 6.34 shows that (B, z) is polynomial-time computable.

If $H \in$ 1-IN-3-SAT, then there is an assignment $I_H : \{x_1, \ldots, x_n\} \to \{0, 1\}$ that satisfies exactly one literal in each clause. We obtain

$$\prod_{I_H(x_i)=1} a_i \cdot \prod_{I_H(x_i)=0} b_i \cdot \prod_{i=n+1}^{k} p_{m+i} = \prod_{i=1}^{m+k} p_i.$$

We consider D as a p-generation tree with values

$$I_D(v_i) \stackrel{\text{def}}{=} \begin{cases} a'_i & \text{, if } i = 1, \ldots, n \text{ and } I_H(x_i) = 1 \\ b'_i & \text{, if } i = 1, \ldots, n \text{ and } I_H(x_i) = 0 \\ a'_i & \text{, if } i = n + 1, \ldots, k. \end{cases}$$

By Property 6.30, $I_D(\text{Root}(D))$, the value of the root, can be evaluated as follows.

$$
\begin{aligned}
I_D(\mathrm{Root}(D)) &= \prod_{v\in \mathrm{Leaf}(D)} I_D(v)^{g(v)} \cdot \prod_{v\in \mathrm{Node}(D)-\mathrm{Leaf}(D)} c^{g(v)} \\
&= \prod_{I_H(x_i)=1} (a_i')^{g(v_i)} \cdot \prod_{I_H(x_i)=0} (b_i')^{g(v_i)} \cdot \prod_{i=n+1}^{k} (a_i')^{g(v_i)} \cdot \prod_{v\in \mathrm{Node}(D)-\mathrm{Leaf}(D)} c^{g(v)} \\
&= \prod_{I_H(x_i)=1} a_i^{a^d b^d} \cdot \prod_{I_H(x_i)=0} b_i^{a^d b^d} \cdot \prod_{i=n+1}^{k} a_i^{a^d b^d} \cdot \prod_{v\in \mathrm{Node}(D)-\mathrm{Leaf}(D)} c^{g(v)} \\
&= \left(\prod_{I_H(x_i)=1} a_i \cdot \prod_{I_H(x_i)=0} b_i \cdot \prod_{i=n+1}^{k} p_{m+i} \right)^{a^d b^d} \cdot \prod_{v\in \mathrm{Node}(D)-\mathrm{Leaf}(D)} c^{g(v)} \\
&= \prod_{i=1}^{m+k} p_i^{a^d b^d} \cdot \prod_{v\in \mathrm{Node}(D)-\mathrm{Leaf}(D)} c^{g(v)} = z.
\end{aligned}
$$

Hence $(B, z) \in \mathrm{GEN_s}(p)$.

Assume $(B, z) \in \mathrm{GEN_s}(p)$. Then there exists an (a, b)-weighted tree $T' = (D', g')$ and a function $\alpha : \mathrm{Node}(D') \to \mathbb{N}$, such that (D', α) is a p-generation tree from B for z. Each element of B has exactly one prime factor from $p_{m+1}, \ldots, p_{m+k}$. Since z has all these prime factors at least once, D' must have at least k leaves. Assume D' has more than k leaves. By Proposition 6.33, there exists a $v \in \mathrm{Leaf}(D')$ such that $g'(v) > \max\{g(u) : u \in \mathrm{Leaf}(D)\}$. $\alpha(v)$ has exactly one prime factor from $p_{m+1}, \ldots, p_{m+k}$; say p_{m+i} with exponent $a^d b^d / g(v_i)$. Hence

$$p_{m+i}^{(a^d b^d / g(v_i)) \cdot g'(v)}$$

is a factor of $\alpha(\mathrm{Root}(D'))$. From $(a^d b^d / g(v_i)) \cdot g'(v) > a^d b^d$ follows that $\alpha(\mathrm{Root}(D')) \neq z$. So D' has exactly k leaves. Each prime $p_{m+1}, \ldots, p_{m+k}$ must appear as a factor in a value of some leaf. Therefore, besides the a_j' with $n+1 \leq j \leq k$, either a_i' or b_i' is a value of a leaf (but not both) for $i = 1, \ldots, n$. Define $I_H : \{x_1, \ldots, x_n\} \to \{0, 1\}$ such that $I_H(x_i) = 1 \overset{\text{def}}{\Leftrightarrow} a_i'$ is a leaf-value of D'. Observe that I_H shows $H \in$ 1-IN-3-SAT. ❑

6.3 The Generation Problem GEN($x^c + ky$)

So far, we do not have upper bounds for generation problems with polynomials $p(x, y) = q(x) + ky$, where q is non-linear and $k \geq 1$. The obvious algorithm guesses and verifies generation trees. We already mentioned that such trees can have exponential size if $k = 1$, although for the generation problem with single carriers, we could work around guessing such a tree. What is their size if $k \geq 2$? To answer this, observe that the trees take a special shape: When we go from the root to the leaves in y-direction, then in each step, the length of the value decreases by one bit. When we go in x-direction, then in each step, the length is bisected. It follows that the size of such trees grows faster than any polynomial, but not as fast as $2^{\log^2 n}$. Therefore, $\mathrm{GEN}(p) \in \mathrm{NTIME}(2^{\log^2 n})$. We do not have to guess complete

generation trees. If a subtree generates some values $b_1, \ldots, b_m$, then it suffices to store these values instead of the whole subtree. We need to store values every time we go in x-direction. So we need space $O(n \log n)$.

Proposition 6.36. $\mathrm{GEN}(p) \in \mathrm{NTIME\text{-}SPACE}(2^{\log^2 n}, n \log n)$ *if* $p(x,y) = q(x) + ky$ *where* $k \geq 2$ *and* q *is a non-linear polynomial.*

Because of the special form of a generation tree for such polynomials, the generation problem can be solved by special alternating machines: Some z can be generated via p from A if and only if there exist $z_1, \ldots, z_n \leq z$ such that $n \leq |z|$, $z = z_1$, $z_n \in A$, and for all $1 \leq i < n$, $z_i = p(y_i, z_{i+1})$ where y_i can be generated via p from A and $|y_i| \leq \frac{1}{2}|z_i|$. An alternating machine can check this predicate in polynomial time with a logarithmic number of alternations. Furthermore, in existential parts the machine guesses polynomially many bits. In contrast, in universal parts it guesses logarithmically many bits. This works for one carrier as well as for a finite number of carriers.

This discussion shows that $\mathrm{GEN}(p)$ can be solved with quite restricted resources. However, we do not know whether $\mathrm{GEN}(p)$ belongs to NP. Standard diagonalizations show that there exist oracles A and B such that $\mathrm{BPP}^A \not\subseteq \mathrm{NTIME}(2^{\log^2 n})^A$ and $\mathrm{coNP}^B \not\subseteq \mathrm{NTIME}(2^{\log^2 n})^B$. Therefore, we should not expect $\mathrm{GEN}(p)$ to be hard for any class that contains BPP or coNP. This rules out many reasonable classes above NP to be reducible to $\mathrm{GEN}(p)$. We consider this as a hint that $\mathrm{GEN}(p)$ could be contained in NP, but we do not have a proof for this. We leave this as an open question.

Nevertheless, in this section we prove lower bounds. The main result, Theorem 6.51, shows that if $p(x,y) = x^c + ky$ where $c, k \geq 1$, then $\mathrm{GEN_s}(p)$ is $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard for NP. The proof is difficult for two reasons which we want to explain for $p(x,y) = x^2 + 2y$.

1. We have to encode NP-computations into generation problems. For this, we need to construct an instance (B, z) of $\mathrm{GEN_s}(p)$ that represents information about a given NP-computation. The elements of B must be chosen in a way so that squaring will not destroy this information. This is difficult, since squaring a number heavily changes its (binary) representation.
2. We construct (B, z) such that if z can be generated, then x must be chosen always from B (and is not a generated number). So the generation tree is linear which makes it easier to control because every value from B has to be taken to the power of c exactly once. On the other hand, the intermediate result is multiplied by 2 in every step, i.e., the number generated so far is shifted to the left. We have to cope with this shifting.

With regard to item 2, our construction makes sure that the size of the linear generation tree is bounded. So the number of shifts is bounded. For B we choose numbers that are much longer than this bound such that each number is provided with a unique stamp. The stamps make sure that there is at most one possible tree that generates z. In particular, this fixes the sequence of numbers from B that are chosen for x. This keeps the shifting under control.

The problem in item 1 is more complicated and also more interesting. It comes down to prove NP-hardness of the following extended sum-of-subset problem.

$$\mathrm{SOS}_2 \stackrel{\text{def}}{=} \{(w_1, \ldots, w_n, z) : \exists I \subseteq \{1, \ldots, n\}(\Sigma_{i \in I} w_i^2 = z)\}$$

(In the proof we use a promise problem related to SOS_2, but for simplicity we argue with SOS_2 in this sketch.) First we reduce 1-IN-3-SAT to SOS and obtain an SOS instance $w = (w_1, \ldots, w_{2n}, z)$. The reduction is such that either $w \notin \mathrm{SOS}$ or there is a selection of exactly n weights which sum up to z. We choose a base b larger than $2n$ and $2\Sigma_i w_i^2$. So in the system to base b, z and all w_i^2 fit into one digit. For each w_i, define the following 6-digit numbers in the system to base b.

$$\begin{aligned} a_i &\stackrel{\text{def}}{=} \langle 11000 w_i \rangle_b \\ r_i &\stackrel{\text{def}}{=} \langle 10001 w_i \rangle_b \end{aligned}$$

The set of all a_i and all r_i build the weights for the SOS_2 instance we want to construct. The intention is to use the weight a_i whenever w_i is used in the sum that yields z, and to use r_i whenever w_i is not used. The squares of a_i and r_i look as follows with respect to base b.

$$\begin{aligned} a_i^2 &\stackrel{\text{def}}{=} \langle\ 1\ \ 2\ 1\ 0\ 0\ 2w_i\ \ 2w_i\ \ 0\ 0\ \ 0\ \ w_i^2 \rangle_b \\ r_i^2 &\stackrel{\text{def}}{=} \langle\ 1\ \ 0\ 0\ 0\ 2\ 2w_i\ \ 0\ \ 0\ 1\ 2w_i\ \ w_i^2 \rangle_b \end{aligned}$$

Note that a_i^2 and r_i^2 have the same first digit, the same last digit, and the same digit at the middle position. At all other positions, either a_i^2 or r_i^2 has digit 0. In the sum for SOS_2, for every i, either a_i or r_i is used. Therefore, in system b, the last digit of this sum becomes predictable: It must be $\Sigma_i w_i^2$. This is the most important point in our argumentation. Also, we choose exactly n weights a_i and n weights r_i. With $s_1 \stackrel{\text{def}}{=} \Sigma_i w_i$, $s_2 \stackrel{\text{def}}{=} \Sigma_i w_i^2$, and $\overline{z} \stackrel{\text{def}}{=} s_1 - z$ we can easily describe the destination number for the SOS_2 instance.

$$z' \stackrel{\text{def}}{=} \langle 2n\ \ 2n\ \ n\ \ 0\ \ 2n\ \ 2s_1\ \ \ 2z\ \ \ 0\ \ n\ \ 2\overline{z}\ \ \ \ s_2 \rangle_b$$

We obtain the instance $(a_1, r_1, \ldots, a_{2n}, r_{2n}, z')$ which belongs to SOS_2 if and only if $(w_1, \ldots, w_{2n}, z) \in \mathrm{SOS}$. This shows NP-hardness for SOS_2 and solves the difficulty mentioned in item 1.

We inductively use this technique to show that for all $c \geq 1$, the following extended sum-of-subset problem is NP-complete.

$$\mathrm{SOS}_c \stackrel{\text{def}}{=} \{(w_1, \ldots, w_n, z) : \exists I \subseteq \{1, \ldots, n\}(\Sigma_{i \in I} w_i^c = z)\}.$$

We need SOS_c as an auxiliary problem for generation problems. However, we feel that this new NP-completeness result is interesting in its own right.

6.3.1 Notations

We work with pairs (A, B) of disjoint languages (where for example $A \in \mathrm{NP}$ and $B \in \mathrm{coNP}$). Say that pair (A, B) *reduces to pair* (C, D), $((A, B) \leq_{\mathrm{m}}^{\mathrm{pp}} (C, D))$, if there exist a polynomial-time computable function f such that for all x,

$$\begin{aligned} x \in A &\Rightarrow f(x) \in C, \\ x \in B &\Rightarrow f(x) \in D. \end{aligned}$$

We will write $(A, \overline{A}) \leq_{\mathrm{m}}^{\mathrm{pp}} (C, D)$ short for $A \leq_{\mathrm{m}}^{\mathrm{pp}} (C, D)$ and $(A, B) \leq_{\mathrm{m}}^{\mathrm{pp}} C$ short for $(A, B) \leq_{\mathrm{m}}^{\mathrm{pp}} (C, \overline{C})$.

In the proofs below we have to construct natural numbers that contain information about NP computations. In addition, these numbers have to contain this information in a way such that exponentiation will not destroy it. For this we need to consider numbers with respect to several bases b. Therefore, we introduce the following notations. For $b \geq 2$ define $A_b = \{0, \ldots, b-1\}$ to be the alphabet that contains b digits. As abbreviation we write A instead of A_2. For digits $a_0, \ldots, a_{n-1} \in A_b$, let $\langle a_{n-1} \cdots a_0 \rangle_b \stackrel{\mathrm{def}}{=} \Sigma_{i=0}^{n-1} a_i b^i$. This means that $\langle a_{n-1} \cdots a_0 \rangle_b$ is the number that is represented by $a_{n-1} \cdots a_0$ with respect to base b.

We will consider vectors of weights $W = (w_1, \ldots, w_{2n})$ such that certain selections of these weights sum up to given destination numbers $z_1, \ldots, z_c$. We group W into pairs (w_1, w_2), (w_3, w_4), and so on. Each pair has a unique stamp u in its binary representation such that the destination number z_c shows the same stamp, but all other pairs have 0's at this position. This allows us to argue that if we want to reach z_c, then from each pair we have to use at least one weight. Moreover, in view of generation problems, we need the stamps still working if the weights are multiplied by small numbers. Therefore, additionally we demand that the stamp u is embedded in s digits 0. We make this precise:

Definition 6.37. *Let $W = (w_1, \ldots, w_{2n})$ and $Z = (z_1, \ldots, z_c)$ where $n, c \geq 1$. Define $\overline{z}_c \stackrel{\mathrm{def}}{=} (\Sigma_{w \in W} w^c) - z_c$. Let $s \geq 1$. We call (W, Z) s-*distinguishable *if all $\mathrm{bin}(w_i^c)$ have the same length l where $l \equiv 1 (\mathrm{mod}\, c)$, and if for every $0 \leq j < n$ there exist $t \geq 1$ and $u \in 1A^*$ such that*

1. $\mathrm{bin}(z_c), \mathrm{bin}(\overline{z}_c), \mathrm{bin}(w_{2j+1}^c), \mathrm{bin}(w_{2j+2}^c) \in A^* 0^s u 0^s A^t$ *and*
2. *for all $i \neq j$, $\mathrm{bin}(w_{2i+1}^c), \mathrm{bin}(w_{2i+2}^c) \in A^* 0^s 0^{|u|} 0^s A^t$.*

Note that, if $c = 1$ then $l \equiv 1 (\mathrm{mod}\, c)$ is always true and is therefore no restriction on the length of l.

6.3.2 NP-Hardness of Modified Sum-of-Subset Problems

We want to show that for $c, k \geq 1$, the generation problem $\mathrm{GEN}(x^c + ky)$ is $\leq_{\mathrm{m}}^{\mathrm{p}}$-hard for NP. The proof is such that the NP-hardness of modified sum-of-subset problems is shown first, and then this is reduced to the generation problems. Our argumentation for the modified sum-of-subset problems is restricted to instances that meet several requirements. Therefore, it is convenient to define these problems as pairs $(L_{c,s}, R_{c,s})$ of disjoint sets.

Definition 6.38. *Let $c, s \geq 1$.*

$$\begin{aligned} L_{c,s} \stackrel{\text{def}}{=} & \big\{(W,Z) : W = (w_1,\ldots,w_{2n}),\ Z = (z_1,\ldots,z_c), \\ & (W,Z) \text{ is } ns\text{-distinguishable, and} \\ & (\exists I \subseteq \{1,\ldots,2n\} \text{ s.t. for } 0 \leq i \leq n-1 \text{ holds } 2i{+}1 \in I \Leftrightarrow 2i{+}2 \notin I) \\ & (\forall m \in \{1,\ldots,c\})[\Sigma_{i\in I} w_i^m = z_m]\big\} \\ R_{c,s} \stackrel{\text{def}}{=} & \big\{(W,Z) : W = (w_1,\ldots,w_{2n}),\ Z = (z_1,\ldots,z_c), \\ & (W,Z) \text{ is } ns\text{-distinguishable, and} \\ & (\forall I \subseteq \{1,\ldots,2n\})(\forall m \in \{1,\ldots,c\})[\Sigma_{i\in I} w_i^m \neq z_m]\big\} \end{aligned}$$

Observe that for $c, s \geq 1$, $L_{c,s} \cap R_{c,s} = \emptyset$, $L_{c,s} \in$ NP, and $R_{c,s} \in$ coNP. We show NP-hardness for $c = 1$ first, and then inductively for higher c's.

Lemma 6.39. *For $s \geq 1$, $(L_{1,s}, R_{1,s})$ is $\leq_{\text{m}}^{\text{pp}}$-hard for* NP.

Proof. For $s \geq 1$, we show that 1-IN-3-SAT $\leq_{\text{m}}^{\text{pp}}$ $(L_{1,s}, R_{1,s})$ via reduction f. Let H be a 3-CNF formula with clauses $C_1,\ldots,C_m$ and variables $x_1,\ldots,x_n$ where $n \geq 2$. For $0 \leq i \leq n-1$ let

$$\begin{aligned} a &\stackrel{\text{def}}{=} 0^{sn}10^{sn}, \\ a_i &\stackrel{\text{def}}{=} 0^{i(2sn+1)}a0^{(n-i-1)(2sn+1)}, \\ w_{2i+1} &\stackrel{\text{def}}{=} \langle 1a_i c_{i1}\ldots c_{im}\rangle_2, \text{ and} \\ w_{2i+2} &\stackrel{\text{def}}{=} \langle 1a_i \bar{c}_{i1}\ldots \bar{c}_{im}\rangle_2 \end{aligned}$$

where

$$c_{ij} = \begin{cases} 0^{n-1}1 & \text{, if } x_i \text{ is a literal in } C_j \\ 0^n & \text{, otherwise} \end{cases}$$

and

$$\bar{c}_{ij} = \begin{cases} 0^{n-1}1 & \text{, if } \overline{x_i} \text{ is a literal in } C_j \\ 0^n & \text{, otherwise.} \end{cases}$$

Finally, define the reduction as $f(H) \stackrel{\text{def}}{=} ((w_1,\ldots,w_{2n}),(z))$ and for $d \stackrel{\text{def}}{=} |\text{bin}(w_i)| - 1 = n(2sn+1) + mn$, let

$$z \stackrel{\text{def}}{=} n2^d + \left\langle a^n (0^{n-1}1)^m \right\rangle_2.$$

Observe, that $z = \frac{1}{2}\Sigma_{i=1}^{2n} w_i$, and therefore $\overline{z} \stackrel{\text{def}}{=} \Sigma_{i=1}^{2n} w_i - z = z$. Hence, $((w_1,\ldots,w_{2n}),(z))$ is ns-distinguishable.

Let $H \in$ 1-IN-3-SAT. Then there exists an assignment $\Phi : \{x_1,\ldots,x_n\} \to \{0,1\}$ such that each clause in H is satisfied by exactly one literal. Let

$$I \stackrel{\text{def}}{=} \{2i+1 : 0 \leq i < n \text{ and } \Phi(x_i) = 1\} \cup \{2i+2 : 0 \leq i < n \text{ and } \Phi(x_i) = 0\}.$$

It follows $\Sigma_{i\in I} w_i = z$ and hence $((w_1,\ldots,w_{2n}),(z)) \in L_{1,s}$.

Let $H \notin$ 1-IN-3-SAT and suppose there exists $I \subseteq \{1,\ldots,2n\}$ such that $z = \Sigma_{i\in I} w_i$. For all i, $w_i > 2^d$. Also, $z < (n+1)2^d$, since $\left\langle a^n(0^{n-1}1)^m \right\rangle_2 < 2^d$. Therefore, I contains at most n elements. On the other hand, $w_i < 2^d + 2^{d-n}$, for all i. Since $(n-1)(2^d+2^{d-n}) < n2^d$ we obtain $|I| = n$.

Since $((w_1,\ldots,w_{2n}),(z))$ is ns-distinguishable, I must contain exactly one element from each pair (w_{2i+1}, w_{2i+2}). For every $k \in \{0,\ldots,m-1\}$ there exists exactly one $j \in I$ such that $w_j[d-kn+1] = 1$: Otherwise, in $\mathrm{bin}(\Sigma_{i\in I} w_i)$ there is a 1 at position $kn+t$ where $1 \le t < n$. This is impossible. Therefore, if Φ is defined such that $\Phi(x_i) = 1 \Leftrightarrow 2i+1 \in I$, then Φ satisfies exactly one literal in each clause. This contradicts our assumption. Hence, $((w_1,\ldots,w_{2n}),(z)) \in R_{1,s}$. ❑

So far we know that $(L_{1,s}, R_{1,s})$ is NP-hard. This is the induction base of our argumentation. Now we turn to the induction step and show how to reduce hardness to pairs $(L_{c,s}, R_{c,s})$ where $c > 0$.

Lemma 6.40. *For $c, s \ge 1$, $(L_{c,2s+c}, R_{c,2s+c}) \le^{\mathrm{pp}}_{\mathrm{m}} (L_{c+1,s}, R_{c+1,s})$.*

Proof. We describe the reduction f on input (W,Z) where $W = (w_1,\ldots,w_{2n})$ and $Z = (z_1,\ldots,z_c)$. Let $w = \max(W)$ and choose $l' \equiv 0(c+1)$ such that $b \stackrel{\mathrm{def}}{=} 2^{l'} > 4n(c+1)!\cdot w^{c+1}$. All w_i belong to A_b. For $1 \le k \le 2n$, define the following weights (where a means *accepted* weight and r means *rejected* weight).

$$\begin{aligned} a_k &\stackrel{\mathrm{def}}{=} \langle 110^c 0 w_k \rangle_b \\ r_k &\stackrel{\mathrm{def}}{=} \langle 100^c 1 w_k \rangle_b \end{aligned}$$

Fix any m such that $1 \le m \le c+1$. In the following we show how to define the right destination number y_m. After that we define $f(W,Z) = (W',Z')$ where $W' = (a_1,a_2,r_1,r_2,a_3,a_4,r_3,r_4,\ldots,r_{2n-1},r_{2n})$ and $Z' = (y_1,\ldots,y_{c+1})$. By binomial theorem,

$$a_k^m = \sum_{i=0}^{m}\sum_{j=0}^{i} \binom{m}{i}\binom{i}{j} w_k^{m-i}\cdot b^{(c+2)i+j}, \tag{6.8}$$

$$r_k^m = \sum_{i=0}^{m}\sum_{j=0}^{i} \binom{m}{i}\binom{i}{j} w_k^{m-i}\cdot b^{(c+2)j+i}. \tag{6.9}$$

Observe that in (6.8) each term $b^{(c+2)i+j}$ appears uniquely: If $(c+2)i+j = (c+2)i'+j'$, then (since $j < c+2$ and $j' < c+2$) $j = j'$ and $i = i'$. Similarly, in (6.9) each term $b^{(c+2)j+i}$ appears uniquely. Now the idea is, to let $a_k(t)$ denote the coefficient of b^t in equation (6.8), and to let $r_k(t)$ denote the coefficient of b^t in equation (6.9). First, we define $a_k(t)$ and $r_k(t)$ for $0 \le t \le (c+3)m \stackrel{\mathrm{def}}{=} d$ and then we show that this definition fits our idea.

$$a_k(t) \stackrel{\text{def}}{=} \begin{cases} \binom{m}{i}\binom{i}{j} w_k^{m-i} & \text{, if } t = (c+2)i + j \text{ for } 0 \le j < i \le m \\ 0 & \text{, if } t = (c+2)j + i \text{ for } 0 \le j < i \le m \text{ or} \\ & \text{if } t \ne (c+2)i + j \text{ for all } 0 \le i, j \le m \\ \binom{m}{i} w_k^{m-i} & \text{otherwise, i.e., if } t = (c+3)i \end{cases} \tag{6.10}$$

$$r_k(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{, if } t = (c+2)i + j \text{ for } 0 \le j < i \le m \text{ or} \\ & \text{if } t \ne (c+2)i + j \text{ for all } 0 \le i, j \le m \\ \binom{m}{i}\binom{i}{j} w_k^{m-i} & \text{, if } t = (c+2)j + i \text{ for } 0 \le j < i \le m \\ \binom{m}{i} w_k^{m-i} & \text{otherwise, i.e., if } t = (c+3)i \end{cases} \tag{6.11}$$

Note that $a_k(t)$ and $r_k(t)$ depend on m. We abstain from taking m as additional index, since m will always be clear from the context. Observe that the three cases in these definitions are indeed disjoint. So $a_k(t)$ and $r_k(t)$ are well-defined. It follows that $a_k(t)$ and $r_k(t)$ are the announced coefficients from equations (6.8) and (6.9). Hence

$$\begin{aligned} a_k^m &= \Sigma_{t=0}^d a_k(t) \cdot b^t \quad \text{and} \\ r_k^m &= \Sigma_{t=0}^d r_k(t) \cdot b^t. \end{aligned}$$

All $a_k(t)$ and all $r_k(t)$ are less than $b/4n$ and therefore belong to A_b. Hence,

$$a_k^m = \langle a_k(d) \cdots a_k(1) a_k(0) \rangle_b \quad \text{and} \tag{6.12}$$

$$r_k^m = \langle r_k(d) \cdots r_k(1) r_k(0) \rangle_b . \tag{6.13}$$

Equations (6.10) and (6.11) tell us that these representations to base b differ only at positions $t \not\equiv 0(c+3)$.

In order to define the destination number y_m, we show how to transfer a selection of weights w_k to a corresponding selection of weights a_k^m and r_k^m. Suppose $\Sigma w_k = z_1$ where the sum ranges over a suitable collection of n weights. Now choose a_k^m for every weight w_k that is used (i.e. k *accepted*) in the sum Σw_k; and choose r_k^m for every weight w_k that is not used (i.e. k *rejected*) in this sum. The choice of whether to take a_k^m or r_k^m only matters for positions $t \not\equiv 0(c+3)$. By equations (6.10) and (6.11), at these positions, either r_k^m has digit 0 and a_k^m has digit $\binom{m}{i}\binom{i}{j} w_k^{m-i}$, or a_k^m has digit 0 and r_k^m has digit $\binom{m}{i}\binom{i}{j} w_k^{m-i}$ (note that $i > 0$ since $i \ne j$). So when we consider the sum of all chosen a_k^m and r_k^m at such a position, then either we see digit

$$\binom{m}{i}\binom{i}{j} \sum_{k \text{ accepted}} w_k^{m-i} = \binom{m}{i}\binom{i}{j} z_{m-i},$$

or we see digit

$$\binom{m}{i}\binom{i}{j} \sum_{k \text{ rejected}} w_k^{m-i} = \binom{m}{i}\binom{i}{j} \overline{z}_{m-i}$$

where $z_0 \stackrel{\text{def}}{=} n$ and $\overline{z}_i = \sum_{w \in W} w^i - z_i$ as defined above. This motivates the following digits of the destination number y_m.

$$y(t) \stackrel{\text{def}}{=} \begin{cases} \binom{m}{i}\binom{i}{j} z_{m-i} & \text{, if } t = (c+2)i + j \text{ for } 0 \le j < i \le m \\ \binom{m}{i}\binom{i}{j} \overline{z}_{m-i} & \text{, if } t = (c+2)j + i \text{ for } 0 \le j < i \le m \\ 0 & \text{, if } t \ne (c+2)i + j \text{ for all } 0 \le i, j \le m \\ \sum_{w \in W} \binom{m}{i} w^{m-i} & \text{otherwise, i.e., if } t = (c+3)i \end{cases}$$

Here again we abstain from taking m as index, since m will be clear from the context. Define the m-th destination number as

$$y_m \stackrel{\text{def}}{=} \langle y(d) \cdots y(1) y(0) \rangle_b .$$

To finish f's definition, let $f(W, Z) \stackrel{\text{def}}{=} (W', Z')$ where $W' = (a_1, a_2, r_1, r_2, a_3, a_4, r_3, r_4, \ldots, r_{2n-1}, r_{2n})$ and $Z' = (y_1, \ldots, y_{c+1})$.

Claim 6.41. If (W, Z) is $(2s + c)n$-distinguishable, then $f(W, Z) = (W', Z')$ is $2ns$-distinguishable.

Proof: Fix $m = c + 1$ and let $d = (c + 3)m$. Observe that for every k, $a_k(d) = r_k(d) = 1$. By assumption, $b = 2^{l'}$ for $l' \equiv 0(c + 1)$. Hence one digit from A_b corresponds exactly to l' bits. By equations (6.12) and (6.13), for every k, $|\text{bin}(a_k^{c+1})| = |\text{bin}(r_k^{c+1})| = d \cdot l' + 1$. This number is $\equiv 1(c + 1)$.

We need to understand the structure of $\overline{y}_{c+1} = (\Sigma_{w \in W'} w^{c+1}) - y_{c+1}$, the complement of y_{c+1}. For this end, define

$$\overline{y}(t) \stackrel{\text{def}}{=} \begin{cases} \binom{m}{i}\binom{i}{j} \overline{z}_{m-i} & \text{, if } t = (c+2)i + j \text{ for } 0 \le j < i \le m \\ \binom{m}{i}\binom{i}{j} z_{m-i} & \text{, if } t = (c+2)j + i \text{ for } 0 \le j < i \le m \\ 0 & \text{, if } t \ne (c+2)i + j \text{ for all } 0 \le i, j \le m \\ \sum_{w \in W} \binom{m}{i} w^{m-i} & \text{otherwise, i.e., if } t = (c+3)i. \end{cases}$$

Observe that for all t, $y(t) + \overline{y}(t) = \Sigma_{k=1}^{2n}(a_k(t) + r_k(t))$. Hence

$$\langle y(d) \cdots y(1) y(0) \rangle_b + \langle \overline{y}(d) \cdots \overline{y}(1) \overline{y}(0) \rangle_b = \Sigma_{w \in W'} w^m$$

and therefore,

$$\overline{y}_{c+1} = \langle \overline{y}(d) \cdots \overline{y}(1) \overline{y}(0) \rangle_b .$$

Choose any $j < n$ and consider a_{2j+1} and a_{2j+2}. By assumption, (W, Z) is $(2s + c)n$-distinguishable. So there exist $t \ge 1$ and $u \in 1A^*$ such that

1. $\mathrm{bin}(z_c), \mathrm{bin}(\overline{z}_c), \mathrm{bin}(w^c_{2j+1}), \mathrm{bin}(w^c_{2j+2}) \in A^*0^{(2s+c)n}u0^{(2s+c)n}A^t$ and
2. for all $i \neq j$, $\mathrm{bin}(w^c_{2i+1}), \mathrm{bin}(w^c_{2i+2}) \in A^*0^{(2s+c)n}0^{|u|}0^{(2s+c)n}A^t$.

If one multiplies a binary number of the form $A^*0^{i'}u0^{i'}A^t$ by $m = c + 1 \leq 2^c$, then this yields a number of the form $A^*0^{i'-c}u'0^{i'-c}A^{t+c}$ where $u' \in A^{|u|+c}$. So in our case, there exist $t' \geq 1$ and $u' \in 1A^*$ such that

1. $\mathrm{bin}(mz_c), \mathrm{bin}(m\overline{z}_c), \mathrm{bin}(mw^c_{2j+1}), \mathrm{bin}(mw^c_{2j+2}) \in A^*0^{2sn}u'0^{2sn}A^{t'}$ and
2. for all $i \neq j$, $\mathrm{bin}(mw^c_{2i+1}), \mathrm{bin}(mw^c_{2i+2}) \in A^*0^{2sn}0^{|u'|}0^{2sn}A^{t'}$.

Let $t_a = c + 2$. For all i, $a_i(t_a) = mw^c_i$, $r_i(t_a) = 0$, $y(t_a) = mz_c$, and $\overline{y}(t_a) = m\overline{z}_c$. So for $t'' = t' + l' \cdot t_a$,

1. $\mathrm{bin}(y_m), \mathrm{bin}(\overline{y}_m), \mathrm{bin}(a^m_{2j+1}), \mathrm{bin}(a^m_{2j+2}) \in A^*0^{2sn}u'0^{2sn}A^{t''}$,
2. for all $i \neq j$, $\mathrm{bin}(a^m_{2i+1}), \mathrm{bin}(a^m_{2i+2}) \in A^*0^{2sn}0^{|u'|}0^{2sn}A^{t''}$, and
3. for all i, $\mathrm{bin}(r^m_{2i+1}), \mathrm{bin}(r^m_{2i+2}) \in A^*0^{2sn}0^{|u'|}0^{2sn}A^{t''}$.

We obtain the analogous three statements for r_{2j+1} and r_{2j+2} by looking at the position $t_r = 1$. Here for all i, $a_i(t_r) = 0$, $r_i(t_r) = mw^c_i$, $y(t_r) = m\overline{z}_c$, and $\overline{y}(t_r) = mz_c$. Hence (W', Z') is $2ns$-distinguishable. □

Claim 6.42. If $(W, Z) \in L_{c,2s+c}$, then $f(W, Z) = (W', Z') \in L_{c+1,s}$.

Proof: By Claim 6.41, (W', Z') is $2ns$-distinguishable. Let I be as in the definition of $L_{c,2s+c}$, and let $\overline{I} \stackrel{\text{def}}{=} \{1, \ldots, 2n\} - I$. Note $|I| = |\overline{I}| = n$. We choose all a_i such that $i \in I$ and all r_i such that $i \in \overline{I}$. Note that this collection of weights from W' is suitable to show that (W', Z') belongs to $L_{c+1,s}$ (i.e., when numbering the weights of W' from 1 to $4n$, then the indices of chosen weights form an I' where $2i + 1 \in I' \Leftrightarrow 2i + 2 \notin I'$). Fix any $m \in \{1, \ldots, c + 1\}$. Our selection of weights induces the following sum.

$$\begin{aligned} z' &\stackrel{\text{def}}{=} \Sigma_{k\in I}a^m_k + \Sigma_{k\in\overline{I}}r^m_k \\ &= \Sigma_{k\in I}\left\langle a_k(d)\cdots a_k(1)a_k(0)\right\rangle_b + \Sigma_{k\in\overline{I}}\left\langle r_k(d)\cdots r_k(1)r_k(0)\right\rangle_b \end{aligned}$$

We have seen that all $a_k(t)$ and all $r_k(t)$ are less than $b/4n$. So for every t,

$$z'(t) \stackrel{\text{def}}{=} \Sigma_{k\in I}a_k(t) + \Sigma_{k\in\overline{I}}r_k(t)$$

is less than b. This means that if we consider the weights to base b and sum up digit by digit, then there is no sum that is carried forward. It follows that

$$z' = \left\langle z'(d)\ldots z'(1)z'(0)\right\rangle_b.$$

From equations (6.10) and (6.11) we obtain

$$z'(t) = \begin{cases} \binom{m}{i}\binom{i}{j}\Sigma_{k\in I}w_k^{m-i} & \text{, if } t = (c+2)i + j \text{ for } 0 \leq j < i \leq m \\ \binom{m}{i}\binom{i}{j}\Sigma_{k\in\overline{I}}w_k^{m-i} & \text{, if } t = (c+2)j + i \text{ for } 0 \leq j < i \leq m \\ 0 & \text{, if } t \neq (c+2)i + j \text{ for all } 0 \leq i, j \leq m \\ \binom{m}{i}\Sigma_{w\in W}w^{m-i} & \text{otherwise, i.e., if } t = (c+3)i. \end{cases}$$

So for all t, $z'(t) = y(t)$ and therefore, $z' = y_m$. This shows $(W', Z') \in L_{c+1,s}$. □

Claim 6.43. If $(W, Z) \in R_{c,2s+c}$, then $f(W, Z) = (W', Z') \in R_{c+1,s}$.

Proof: By Claim 6.41, (W', Z') is $2ns$-distinguishable. Let us assume $(W', Z') \notin R_{c+1,s}$, i.e., there exists I_a and I_r, subsets of $\{1, \ldots, 2n\}$, and there exists some $m \in \{1, \ldots, c+1\}$ such that

$$\Sigma_{k \in I_a} a_k^m + \Sigma_{k \in I_r} r_k^m = y_m. \tag{6.14}$$

Let $t_a = (c + 2)(m - 1)$. For all k, $a_k(t_a) = mw_k$, $r_k(t_a) = 0$, and $y(t_a) = mz_1$. In Equation (6.14), we can consider the weights to base b and can sum up digit by digit without obtaining a sum that is carried forward. By looking at position t_a we obtain $y(t_a) = \Sigma_{k \in I_a} a_k(t_a)$ and hence

$$z_1 = \Sigma_{k \in I_a} w_k.$$

So we found a collection of weights from W whose sum is z_1. This is a contradiction. □
This complete the proof of Lemma 6.40. ❑

Corollary 6.44. *For $c, s \geq 1$, $(L_{c,s}, R_{c,s})$ is $\leq_{\mathrm{m}}^{\mathrm{pp}}$-hard for* NP.

Proof. The proof is by induction on c. The induction base is by Lemma 6.39 while the induction step follows from Lemma 6.40. ❑

Theorem 6.45. *For $c \geq 1$, the following sum-of-subset problem is $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete for* NP.

$$\mathrm{SOS}_c \stackrel{\mathrm{def}}{=} \{(a_1, \ldots, a_n, b) : \exists I \subseteq \{1, \ldots, n\}(\Sigma_{i \in I} a_i^c = b)\}.$$

Proof. Clearly, $\mathrm{SOS}_c \in$ NP. For given (W, Z) where $W = (w_1, \ldots, w_{2n})$ and $Z = (z_1, \ldots, z_c)$ let $f(W, Z) \stackrel{\mathrm{def}}{=} (w_1, \ldots, w_{2n}, z_c)$. Observe $(L_{c,1}, R_{c,1}) \leq_{\mathrm{m}}^{\mathrm{pp}} \mathrm{SOS}_c$ via f. So, by Corollary 6.44, SOS_c is NP-hard. ❑

6.3.3 NP-Hardness of GEN($x^c + ky$)

Starting from Corollary 6.44 we reduce NP-hardness to generation problems. First, we show this for $c > 1$ and then we treat $\mathrm{GEN}(x + ky)$ in a separate lemma.

Lemma 6.46. *For $c \geq 2$, $k \geq 1$ and $s \stackrel{\mathrm{def}}{=} 5k^2(c + 5)$, $(L_{c,s}, R_{c,s}) \leq_{\mathrm{m}}^{\mathrm{pp}} \mathrm{GEN}(x^c + ky)$.*

Proof. We describe the reduction f on input (W, Z) where $W = (w_1, \ldots, w_{2n})$ and $Z = (z_1, \ldots, z_c)$. We may assume that all w_i and z_j are divisible by 2^{cns}. Otherwise, use $W' = (2^{cns} w_1, \ldots, 2^{cns} w_{2n})$ and $Z' = (2^{cns} z_1, 2^{2cns} z_2, \ldots, 2^{ccns} z_c)$ instead of W and Z. Let $l \stackrel{\mathrm{def}}{=} |\mathrm{bin}(w_1^c)|$ and note that $l > cns$. If $k = 1$, then we use $a = 0$ as auxiliary weight. Otherwise, if $k \geq 2$, then we use $a = 2^{(l-1)/c}$. Observe, that $(l-1)$ is always divisible by c, since (W, Z) is ns-distinguishable.

$$B \stackrel{\text{def}}{=} \{a, 2^{l-1}+1\} \cup \{w_1 k, w_2 k, w_3 k^2, w_4 k^2, \ldots, w_{2n-1} k^n, w_{2n} k^n\} \tag{6.15}$$

$$d \stackrel{\text{def}}{=} k^{cn}(z_c + 2^{l-1} + 1) + a^c \cdot \sum_{i=1}^{c-1} k^i \cdot \sum_{i=0}^{n-1} k^{ic} \tag{6.16}$$

If $n2^{l-1} \leq z_c < n2^l$, then $f(W,Z) \stackrel{\text{def}}{=} (B,d)$, otherwise $f(W,Z) \stackrel{\text{def}}{=} (\emptyset, 0)$. In the following we show $(L_{c,s}, R_{c,s}) \leq_{\text{m}}^{\text{pp}} \text{GEN}(x^c + ky)$ via f.

Case 1: Assume $(W,Z) \in L_{c,s}$. Hence there exist weights $x_1, \ldots, x_n \in W$ such that $\sum_{i=1}^{n} x_i^c = z_c$ where $x_1 \in \{w_1, w_2\}$, $x_2 \in \{w_3, w_4\}$, and so on. Therefore, $n2^{l-1} \leq z_c < n2^l$ and so $f(W,Z) = (B,d)$. We describe the generation of d. Clearly, $y_0 \stackrel{\text{def}}{=} 2^{l-1} + 1$ can be generated. For $j \geq 1$, let

$$y_j \stackrel{\text{def}}{=} k^c \cdot y_{j-1} + (k^j x_j)^c + a^c \cdot \sum_{i=1}^{c-1} k^i. \tag{6.17}$$

If y_{j-1} can be generated, then so can y_j: For $k = 1$ this is trivial. For $k \geq 2$, start with y_{j-1} and apply the generation $y_{\text{new}} = a^c + k \cdot y_{\text{old}}$ for $c - 1$ times. Then apply the generation $y_{\text{new}} = (k^j x_j)^c + k \cdot y_{\text{old}}$ (note that $k^j x_j \in B$). This yields y_j. Hence y_n can be generated. From equation (6.17) we obtain

$$y_n = k^{cn} \sum_{i=1}^{n} x_j^c + k^{cn}(2^{l-1}+1) + a^c \cdot \sum_{i=1}^{c-1} k^i \cdot \sum_{i=0}^{n-1} k^{ic}.$$

It follows that $d = y_n$ and therefore, $(B,d) \in \text{GEN}(x^c + ky)$.

Case 2: Assume $(W,Z) \in R_{c,s}$. If $z_c < n2^{l-1}$ or $z_c \geq n2^l$, then $f(W,Z) = (\emptyset, 0) \notin \text{GEN}(x^c + ky)$ and we are done. So let us assume $n2^{l-1} \leq z_c < n2^l$ and $f(W,Z) = (B,d) \in \text{GEN}(x^c + ky)$. In the remaining proof we will derive a contradiction which will prove the lemma.

If $k \geq 2$, then from equation (6.16) and $z_c < n2^l$ we obtain

$$\begin{aligned}
d &= k^{cn}(z_c + 2^{l-1} + 1) + a^c \cdot \sum_{i=1}^{c-1} k^i \cdot \sum_{i=0}^{n-1} k^{ic} \\
&< k^{cn}(n2^l + 2^{l-1} + 1) + a^c \cdot k^c \cdot \sum_{i=0}^{n-1} k^{ic} \\
&= (n2^l k^{cn} + 2^{l-1} k^{cn} + k^{cn}) + a^c \cdot \sum_{i=0}^{n-1} k^c k^{ic} \\
&< (n+1)2^l k^{cn} + a^c \cdot \sum_{i=1}^{n} k^{ic} \\
&< (n+1)2^l k^{cn} + a^c k^{cn+1}
\end{aligned}$$

Hence

$$k = 1 \quad \Rightarrow \quad d < (n+1)2^l, \tag{6.18}$$
$$k \geq 2 \quad \Rightarrow \quad d < 2^l k^{n(c+3)}. \tag{6.19}$$

Claim 6.47. There exist $m \geq 1$, $y_0 \in B$ and $x_1, \ldots, x_m \in B - \{0, 2^{l-1}+1\}$ such that

$$d = k^m y_0 + \sum_{i=1}^{m} k^{m-i} x_i^c. \tag{6.20}$$

Proof: We have seen that $l > cns$. From equations (6.18) and (6.19) it follows that $d < 2^{l+ns-2} < 2^{2l-2}$. For all $x \in B - \{0\}$, $|\mathrm{bin}(x^c)| \geq l$. So if z can be generated and is not already in B, then $|\mathrm{bin}(z)| \geq l$ and therefore $z \geq 2^{l-1}$. If we apply the generation rule $x^c + ky$ for $x = z$ and any y, then, since $c \geq 2$, we obtain $z' \geq 2^{2l-2} > d$ which cannot be used to generate d. Similarly, if we apply the generation rule $x^c + ky$ for $x = 2^{l-1}+1 \in B$ and any y, then we obtain $z' \geq 2^{2l-2} > d$ which cannot be used to generate d. Hence, there exists a generation of d such that in each step, x is chosen from $B - \{0, 2^{l-1}+1\}$. From $z_c \geq 2^{l-1}$ and equation (6.16) it follows that $d \geq 2^l k^{cn}$ and hence $d \notin B$. Therefore, d can be generated in the following linear way: There exist $m \geq 1$, $y_0 \in B$, and $x_1, \ldots, x_m \in B - \{0, 2^{l-1}+1\}$ such that if $y_i \stackrel{\mathrm{def}}{=} x_i^c + k \cdot y_{i-1}$ for $1 \leq i \leq m$, then $y_m = d$. This is equivalent to the statement in the claim. □

Claim 6.48. 1. $y_0 = 2^{l-1}+1$.
2. If $k = 1$, then $m \leq 2n$.
3. If $k \geq 2$, then $m = cn$.

Proof: First, we show $m < ns/k^2$. Assume $m \geq ns/k^2$ and $k = 1$. By Claim 6.47, $d > m2^{l-1}$. From equation (6.18) it follows that $d > 2^{l-1}ns/k^2 \geq 2^{l+1} \cdot n(c+5) > d$ which is a contradiction. Assume $m \geq ns/k^2$ and $k \geq 2$. By Claim 6.47, $d > 2^{l-1}k^{m-1}$. From equation (6.19) it follows that $d > 2^l k^{(ns/k^2)-2} \geq 2^l k^{5n(c+3)} > d$ which is a contradiction. Therefore,

$$m < ns/k^2. \tag{6.21}$$

Assume $y_0 \neq 2^{l-1}+1$, i.e., $y_0 \in B - \{2^{l-1}+1\}$. By assumption, all w_i and z_i are $\equiv 0(2^{cns})$. So all elements in $B - \{2^{l-1}+1\}$ are $\equiv 0(2^{ns})$ (if $k \geq 2$, then $a = 2^{(l-1)/c} \geq 2^{ns}$). From Claim 6.47 we obtain $d \equiv 0(2^{ns})$. However, equation (6.16) says that $d \equiv k^{cn}(2^{ns})$. Since $0 < k^{cn} < 2^{kcn} < 2^{ns}$ we have $d \not\equiv 0(2^{ns})$. This is a contradiction and we obtain $y_0 = 2^{l-1}+1$.

We have seen that all elements in $B - \{2^{l-1}+1\}$ are equivalent to 0 modulo 2^{ns}. By Claim 6.47, $d \equiv k^m(2^{ns})$. By equation (6.16), $d \equiv k^{cn}(2^{ns})$. By equation (6.21), $k^m \leq 2^{km} < 2^{ns}$ and $k^{cn} < 2^{ns}$. Therefore, if $k \geq 2$, then $m = cn$. If $k = 1$, then by Claim 6.47, $d \geq (m+1)2^{l-1}$. So by equation (6.18), $m \leq 2n$. □

Claim 6.49. For every j, $1 \leq j \leq n$, there exists exactly one i such that $x_i \in \{w_{2j-1}k^j, w_{2j}k^j\}$. If $k \geq 2$, then this i is determined by $i = jc$.

Proof: Fix j. By assumption, (W, Z) is ns-distinguishable. So there exist $t \geq 1$ and $u \in 1A^*$ such that $\mathrm{bin}(z_c), \mathrm{bin}(w_{2j-1}^c), \mathrm{bin}(w_{2j}^c) \in A^*0^{ns}u0^{ns}A^t$ and for all $i \neq j$, $\mathrm{bin}(w_{2i-1}^c), \mathrm{bin}(w_{2i}^c) \in A^*0^{ns}0^{|u|}0^{ns}A^t$. Let $r \stackrel{\text{def}}{=} 2ns + |u| + t$. In the following calculation we are mainly interested in the lower r bits of all x_i^c. If $\alpha \stackrel{\text{def}}{=} \langle u \rangle_2$, then

$$(w_{2j-1}^c \bmod 2^r) = \alpha 2^{ns+t} + \beta_1 \quad \text{and} \tag{6.22}$$

$$(w_{2j}^c \bmod 2^r) = \alpha 2^{ns+t} + \beta_2, \tag{6.23}$$

where $\beta_1, \beta_2 < 2^t$. We partition the set of indices $\{1, \ldots, m\}$.

$$\begin{aligned} J_1 &\stackrel{\text{def}}{=} \{i : 1 \leq i \leq m \wedge x_i = w_{2j-1}k^j\} \\ J_2 &\stackrel{\text{def}}{=} \{i : 1 \leq i \leq m \wedge x_i = w_{2j}k^j\} \\ J_3 &\stackrel{\text{def}}{=} \{1, \ldots, m\} - (J_1 \cup J_2) \end{aligned}$$

From equation (6.20) we obtain

$$d = \sum_{i \in J_1} k^{m-i}(w_{2j-1}k^j)^c + \sum_{i \in J_2} k^{m-i}(w_{2j}k^j)^c + \sum_{i \in J_3} k^{m-i}x_i^c + k^m y_0. \tag{6.24}$$

Now we study equation (6.24) modulo 2^r. We start with the first two sums and consider w_{2j-1}^c and w_{2j}^c modulo 2^r. By equations (6.22) and (6.23), these terms consist of an upper part (i.e., $\alpha 2^{ns+t}$) and of a lower part (i.e., β_1 or β_2). Let e_1 (resp., e_2) denote the sum of the upper (resp., lower) parts:

$$e_1 \stackrel{\text{def}}{=} \sum_{i \in J_1} k^{m-i}k^{jc} \cdot \alpha 2^{ns+t} + \sum_{i \in J_2} k^{m-i}k^{jc} \cdot \alpha 2^{ns+t} \tag{6.25}$$

$$e_2 \stackrel{\text{def}}{=} \sum_{i \in J_1} k^{m-i}k^{jc}\beta_1 + \sum_{i \in J_2} k^{m-i}k^{jc}\beta_2 \tag{6.26}$$

Moreover, let e_3 denote the sum (this time modulo 2^r) of the last two terms in equation (6.24):

$$e_3 \stackrel{\text{def}}{=} ((\sum_{i \in J_3} k^{m-i}x_i^c + k^m y_0) \bmod 2^r) \tag{6.27}$$

Clearly, $d \equiv e_1 + e_2 + e_3 \ (2^r)$. We argue that $(d \bmod 2^r) = e_1 + e_2 + e_3$.

For all $i \in J_3$, either $x_i^c = a^c \neq 0$ or $x_i^c = x'k^{ci'}$ where $\mathrm{bin}(x') \in A^*0^{ns}0^{|u|}0^{ns}A^t$ and $1 \leq i' \leq n$. Therefore, for all $i \in J_3$,

$$(x_i^c \bmod 2^r) < 2^t k^{cn}. \tag{6.28}$$

Moreover, $(y_0 \bmod 2^r) = 1$. Equations (6.27) and (6.28) allow an estimation of e_3.

$$e_3 \leq \sum_{i \in J_3} k^{m-i}2^t k^{cn} + k^m$$

If $k = 1$, then by Claim 6.48, $e_3 \leq 2n2^t + 1$. If $k \geq 2$, then $e_3 \leq cn2^t k^{cn} k^m + k^m$ and $m = cn$. So for all k,

$$e_3 < 2^{ns+t-1}. \tag{6.29}$$

Estimate e_2 with help of equation (6.26) and Claim 6.48:

$$e_2 \leq mk^{m-1}k^{jc}2^t \leq 2^{5knc+t} < 2^{ns+t-1} \tag{6.30}$$

Together with (6.29) this yields

$$e_2 + e_3 < 2^{ns+t}. \tag{6.31}$$

Finally we turn to e_1. Equation (6.25) can be written as

$$e_1 = \alpha 2^{ns+t} k^{jc} \sum_{i \in J_1 \cup J_2} k^{m-i}. \tag{6.32}$$

Therefore, $e_1 < 2^{|u|+ns+t+5knc} \leq 2^{r-1}$. Together with (6.31) we obtain $e_1 + e_2 + e_3 < 2^r$ and hence

$$(d \bmod 2^r) = e_1 + e_2 + e_3. \tag{6.33}$$

By equation (6.16), $d \equiv k^{cn}(z_c + 1)\ (2^r)$. Recall that $\mathrm{bin}(z_c) \in A^* 0^{ns} u 0^{ns} A^t$. Therefore, $(z_c \bmod 2^r) = \alpha 2^{ns+t} + \gamma$ where $\gamma < 2^t$. Observe $k^{cn}(\alpha 2^{ns+t} + \gamma + 1) < 2^{kcn} 2^{|u|} 2^{ns+t+1} \leq 2^r$. This yields

$$(d \bmod 2^r) = \alpha 2^{ns+t} k^{cn} + k^{cn}(\gamma + 1). \tag{6.34}$$

Compare equations (6.33) and (6.34). The terms e_1 and $\alpha 2^{ns+t} k^{cn}$ are divisible by 2^{ns+t}, while the terms $e_2 + e_3$ and $k^{cn}(\gamma + 1)$ are less than 2^{ns+t} (see Equations 6.29 and 6.30). It follows that $e_1 = \alpha 2^{ns+t} k^{cn}$ and therefore, by equation (6.32),

$$\sum_{i \in J_1 \cup J_2} k^{m-i} = k^{c(n-j)}. \tag{6.35}$$

For $k = 1$ this implies $|J_1 \cup J_2| = 1$, while for $k \geq 2$ this implies $|J_1 \cup J_2| \geq 1$. Assume $k \geq 2$ and let i' be the maximum of $J_1 \cup J_2$. The left hand side of (6.35) is $\equiv k^{nc-i'}\ (k^{nc-i'+1})$. So it must be that $k^{c(n-j)} < k^{nc-i'+1}$ and therefore, $k^{nc-i'} = k^{c(n-j)}$. Hence $J_1 \cup J_2 = \{jc\}$. This proves Claim 6.49. □

Assume $k = 1$. By Claims 6.47, 6.48, and 6.49, there exist $\overline{x}_i \in \{w_{2i-1}, w_{2i}\}$ such that

$$d = (2^{l-1} + 1) + \sum_{i=1}^{n} \overline{x}_i^c. \tag{6.36}$$

Together with equation (6.16) this shows $z_c = \sum_{i=1}^{n} \overline{x}_i^c$. So $(W, Z) \notin R_{c,s}$ which contradicts our assumption.

Assume $k \geq 2$. By Claim 6.49, for every j, $x_{jc} = \overline{x}_j \cdot k^j$ where $\overline{x}_j \in \{w_{2j-1}, w_{2j}\}$. Moreover, it follows that for every i, if $i \not\equiv 0(c)$, then $x_i = a$. So equation (6.20) can be written as:

$$d = k^m y_0 + \sum_{j=1}^{n} k^{m-jc} x_{jc}^c + \sum_{\substack{i\in\{1,\ldots,m\},\\ i\not\equiv 0(c)}} k^{m-i} x_i^c \tag{6.37}$$

$$= k^{nc}(a^c+1) + k^{nc}\sum_{j=1}^{n} \overline{x}_j^c + a^c \sum_{\substack{i\in\{1,\ldots,nc\},\\ i\not\equiv 0(c)}} k^{nc-i} \tag{6.38}$$

Observe that the right-most sum in (6.38) can be written as

$$\sum_{i=1}^{c-1} k^i \cdot \sum_{i=0}^{n-1} k^{ic}.$$

So we can continue to transform d.

$$d = k^{nc}(a^c+1) + k^{nc}\sum_{j=1}^{n} \overline{x}_j^c + a^c \cdot \sum_{i=1}^{c-1} k^i \cdot \sum_{i=0}^{n-1} k^{ic}$$

Together with equation (6.16),

$$z_c = \sum_{j=1}^{n} \overline{x}_j^c.$$

So again $(W, Z) \notin R_{c,s}$ which contradicts our assumption. ❑

Lemma 6.50. *If $p(x, y) = x + ky$ where $k \geq 1$ then $\mathrm{GEN_s}(p)$ is $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete for* NP.

Proof. We have already seen the upper bound (Lemma 6.18) and the lower bound for the case $k = 1$ in Theorem 6.14, so let us focus on the lower bound for $k \geq 2$. We $\leq_{\mathrm{m}}^{\mathrm{pp}}$-reduce $(L_{1,2k}, R_{1,2k})$ to $\mathrm{GEN_s}(x + ky)$. Let $W \stackrel{\mathrm{def}}{=} (w_1, \ldots, w_{2n})$, $Z \stackrel{\mathrm{def}}{=} (z)$ such that $w_i, z \in \mathbb{N}$ $(1 \leq i \leq 2n)$. Let $\ell \stackrel{\mathrm{def}}{=} |\mathrm{bin}(k\Sigma_{i=1}^{2n} w_i)|$ and $G \stackrel{\mathrm{def}}{=} 2^{\ell+1}$. Define

$$\begin{aligned} v_1 &\stackrel{\mathrm{def}}{=} k(G + w_1), \\ v_2 &\stackrel{\mathrm{def}}{=} k(G + w_2), \\ v_i &\stackrel{\mathrm{def}}{=} G + w_i, \text{ for } 3 \leq i \leq 2n, \text{ and} \\ z' &\stackrel{\mathrm{def}}{=} k(nG + z). \end{aligned}$$

Now let $(W, Z) \in L_{1,2k}$. Then there is an $I = \{i_1, \ldots, i_n\} \subseteq \{1, \ldots, 2n\}$ such that for all $i \in \{0, \ldots, n-1\}$ exactly one of $\{2i+1, 2i+2\}$ is in I and $\Sigma_{i\in I} w_i = z$. Assume that $i_j < i_t$ if $j < t$. Then $p(p(\ldots p(p(v_{i_1}, v_{i_2}), v_{i_3}), \ldots, v_{i_{n-1}}), v_{i_n}) = k(G+w_{i_1}) + k\Sigma_{j=2}^{n} G + w_{i_j} = k(nG + z) = z'$.

Now let $(W, Z) \in R_{1,2k}$ and assume that $(v_1, \ldots, v_{2n}, z') \in \mathrm{GEN_s}(p)$. Observe that $v_i \geq G$ for all $i \in \{1, \ldots, 2n\}$. Let T be a generation tree for z' from $\{v_1, \ldots, v_{2n}\}$ with m leaves. Then obviously $z' \geq \Sigma_{q\,\in\,\mathrm{fpath}(T)} k^{r(q)} G$. Since for every leaf in T except one there is a path q with $r(q) \geq 1$ we have $nkG + G > nkG + kz = z' \geq (m-1)kG + G$ and therefore

$m \leq n$. Suppose there is an $i \in \{0, \ldots, n-1\}$ such that neither v_{2i+1} nor v_{2i+2} is a value of a leaf in T. We know that (W, Z) is $2kn$ distinguishable. Adding G to a w_j $(1 \leq j \leq 2n)$ and nG to z does not interfere with the distinguishing gaps of the values by the choice of G. Multiplying some of the values with k, decreases the size of the distinguishing gaps by at most $\lfloor \log k + 1 \rfloor$. Hence there is a $u \in 1A^*$ and a $t \geq 1$ such that $\text{bin}(z') \in A^*0^{kn}u0^{kn}A^t$ and for all $j \neq i$, both $\text{bin}(v_{2j+1})$ and $\text{bin}(v_{2j+2})$ are in $A^*0^{kn}0^{|u|}0^{kn}A^t$. Since in every step of the generation the size of the distinguishing gap is reduced by at most $\lfloor \log k + 1 \rfloor$ and since there are at most $n-1$ steps in the whole generation process, z' can not be generated. Hence for all $i \in \{0, \ldots, n-1\}$ exactly one element of $\{v_{2i+1}, v_{2i+2}\}$ is a value of a leaf in T and $m = n$. If there were a path q in T with $r(q) > 1$ then

$$nkG + G > z' \geq k^2G + (n-2)kG + G \geq nkG + G \tag{6.39}$$

would hold. Therefore $\text{fpath}(T) = \{l^{n-1}\} \cup \{l^i r : 0 \leq i \leq n-2\}$. Since $v_1, v_2 \geq kG$ the value of the leaf with the path l^{n-1} has to be one of $\{v_1, v_2\}$ otherwise again Equation 6.39 would hold. So there are $\{i_1, \ldots, i_n\}$ such that $i_1 \in \{1, 2\}$ and

$$\begin{aligned} z' &= p(p(\ldots p(p(v_{i_1}, v_{i_2}), v_{i_3}), \ldots, v_{i_{n-1}}), v_{i_n}) \\ &= k(G + w_{i_1}) + k\Sigma_{j=2}^{n} G + w_{i_j} \\ &= k(nG + \Sigma_{j=1}^{n} w_{i_j}) \\ &= k(nG + z) \end{aligned}$$

and therefore $\Sigma_{j=1}^{n} w_{i_j} = z$ which is a contradiction. Hence $(v_1, \ldots, v_{2n}, z') \notin \text{GEN}_\text{s}(p)$. ❑

We combine the auxiliary results proved so far and formulate the main result of this section that follows from Corollary 6.44 and Lemmas 6.46 and 6.50.

Theorem 6.51. *For $c, k \geq 1$, $\text{GEN}_\text{s}(x^c + ky)$ is $\leq_\text{m}^\text{p}$-hard for* NP.

6.4 A Summary for the Generation Problem

We summarize our results on the complexity of $\text{GEN}_\text{s}(f)$ in the following table. Every lower bound is given by the fact that there exists an f from the considered class of operations whose generation problem is complete for the respective class. All operations are polynomial-time computable.

The complexity of the more general problem $\text{GEN}(f)$ is the same in nearly all cases. However, in the case of a polynomial p where $p(x, y) = x + q(y)$ for some univariate polynomial q, we found an NP upper bound for $\text{GEN}_\text{s}(p)$ which we cannot prove for $\text{GEN}(p)$. The best upper bound we can give for this case is the rather obvious $\text{ASPACE}(O(n))$ and it is an open question whether $\text{GEN}(p) \in \text{NP}$.

operation	lower bound	Thm	upper bound	Thm
arbitrary	recursively enumerable	6.4	recursively enumerable	6.3
length-monotonic	EXPTIME	6.7	EXPTIME	6.5
length-monotonic and commutative	EXPTIME	6.7	EXPTIME	6.5
length-monotonic and associative	PSPACE	6.12	PSPACE	6.11
length-mon., assoc., and commutative	NP	6.14	NP	6.13
all Polynomials $\neq q(x) + ky$	NP	6.14	NP	6.25
$x + y$	NP	6.14	NP	6.13
$x \cdot y$	NP	6.28	NP	6.13
$x^a y^b c$	NP	6.35	NP	6.22
all Polymomials $= q(x) + y$	NP	6.50	NP / ATIME($O(n)$)	6.18
all Polynomials $= q(x) + ky$ $(k \geq 2)$	NP	6.51	NTIME($2^{\log^2 n}$)	6.36
$x^c + ky$	NP	6.51	NTIME($2^{\log^2 n}$)	6.36

The gap between NP and NTIME($2^{\log^2 n}$) in the last rows of the table below calls the attention to another interesting open question: Does GEN($q(x) + ky$) or GEN$_s$($q(x) + ky$) belong to NP if q is non-linear and $k \geq 2$? Since the generation trees for these polynomials may be of super-polynomial size, the obvious algorithm of guessing and verifying the tree is not applicable. Also, we could not find more compact representations as in Lemma 6.18. There are generation trees where almost all nodes take different values. Therefore it may be possible that we really have to calculate all of them. A possibility to solve the problem could be to have a closer look at the restricted alternating machines we describe in Section 6.3. What are the exact capabilities of these machines?

References

[Bab85] L. Babai. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 421–429. ACM Press, 1985.

[BBC+05] M. Bauland, E. Böhler, N. Creignou, S. Reith, H. Schnoor, and H. Vollmer. Quantified constraints: The complexity of decision and counting for bounded alternation. Technical Report 353, Universität Würzburg, 2005.

[BCRV03] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part I: Post's lattice with applications to complexity theory. *ACM-SIGACT Newsletter*, 34(4):38–52, 2003.

[BCRV04] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with Boolean blocks, part II: Constraint satisfaction problems. *ACM-SIGACT Newsletter*, 35(1):22–35, 2004.

[BDG90] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II.* Springer Verlag, Berlin Heidelberg New York, first edition, 1990.

[BDG95] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer Verlag, Berlin Heidelberg New York, 2nd edition, 1995.

[BG82] A. Blass and Y. Gurevich. On the unique satisfiability problem. *Information and Control*, 55:80–88, 1982.

[BGSW] E. Böhler, C. Glaßer, B. Schwarz, and K. W. Wagner. Generation problems. *to appear in Theoretical Computer Science.*

[BKLM01] D. M. Barrington, P. Kadau, K. Lange, and P. McKenzie. On the complexity of some problems on groups input as multiplication tables. *Journal of Computer and System Sciences*, 63, 2001.

[BLS87] L. Babai, E. Luks, and A. Seress. Permutation groups in NC. In *Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 409–420. ACM Press, 1987.

[Böh05] E. Böhler. On the lattice of clones below the polynomial time functions. Technical Report 352, Universität Würzburg, 2005.

[BS84] L. Babai and E. Szemerédi. On the complexity of matrix group problems. In *25th Annual Symposium on Foundations of Computer Science*, pages 229–240, 1984.

[BSRV05] E. Böhler, H. Schnoor, S. Reith, and H. Vollmer. Bases for Boolean co-clones. *Information Processing Letters*, 96:59–66, 2005.

[CF91] J.-Y. Cai and M. Furst. PSPACE survives constant-width bottlenecks. *International Journal of Foundations of Computer Science*, 2:67–76, 1991.

[CKS81] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[CKS01] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems.* Monographs on Discrete Applied Mathematics. SIAM, 2001.

[Coh65] P. M. Cohn. *Universal Algebra*. Harper & Row, New York, Evanston, London and John Wheatherhill, Inc., Tokyo, first edition, 1965.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings 3^{rd} Symposium on Theory of Computation*, pages 151–158, 1971.

[Dal97] V. Dalmau. Some dichotomy theorems on constant-free boolean formulas. Technical report, Universitat Politècnica de Catalunya, 1997.

[Dal00] V. Dalmau. *Computational complexity of problems over generalized formulas*. PhD thesis, Department de Llenguatges i Sistemes Informàtica, Universitat Politécnica de Catalunya, 2000.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge, 1990.

[FHL84] M. Furst, J. Hopcroft, and E. Luks. Polynomial time algorithms for permutation groups. In *21th Annual Symposium on Foundations of Computer Science*, pages 36–41, 1984.

[Grä79] G. Grätzer. *Universal Algebra*. Springer Verlag, Berlin Heidelberg New York, second edition, 1979.

[Hem04] E. Hemaspaandra. Dichotomy theorems for alternation-bounded quantified Boolean formulas. Technical report, Rochester Institute of Technology, 2004.

[HS66] F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape turing machines. *Journal of the ACM*, 13:533–546, 1966.

[JCG97] P. G. Jeavons, D. A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–548, 1997.

[JCG99] P. G. Jeavons, D. A. Cohen, and M. Gyssens. How to determine the expressive power of constraints. *Constraints*, 4:113–131, 1999.

[JGK70] S. W. Jablonski, G. P. Gawrilow, and W. B. Kudrajawzew. *Boolesche Funktionen und Postsche Klassen*. Akademie-Verlag, 1970.

[JK04] P. Jonsson and A. Krokhin. Recognizing frozen variables in constraint satisfaction problems. *Theoretical Computer Science*, 329:93–113, 2004.

[Lad75a] R. Ladner. On the structure of polynomial-time reducibility. *Journal of the ACM*, 22:155–171, 1975.

[Lad75b] R. E. Ladner. The circuit value problem is log-space complete for P. *SIGACT News*, pages 18–20, 1975.

[Lew79] H. R. Lewis. Satisfiability problems for propositional calculi. *Mathematical Systems Theory*, 13:45–53, 1979.

[Mat70] Y. V. Matiyasevich. Enumerable sets are diophantine. *Doklady Akad. Nauk SSSR*, 191:279–282, 1970. Translation in Soviet Math. Doklady, 11:354–357, 1970.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

[Pos41] E. L. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematics Studies*, 5, 1941.

[Pös79] R. Pöschel. *Funktionen- und Relationenalgebren*. Birkäuser Verlag, Basel, Stuttgart, first edition, 1979.

[Raz85] A. A. Razborov. Lower bounds on the monotone complexity of some Boolean functions. *Dokl. Akad. Nauk*, 281:798–801, 1985.

[Rei01] S. Reith. *Generalized Satisfiability Problems*. PhD thesis, University of Würzburg, 2001.

[Rei03] S. Reith. On the complexity of some equivalence problems for propositional calculi. In *Proceedings 28th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 632–641, 2003.

[RW00] S. Reith and K. W. Wagner. The complexity of problems defined by Boolean circuits. Technical Report 255, Institut für Informatik, Universität Würzburg, 2000. To appear in *Proceedings International Conference Mathematical Foundation of Informatics*, Hanoi, October 25–28, 1999.

[Sch78] T. J. Schaefer. The complexity of satisfiability problems. In *Proccedings* 10^{th} *Symposium on Theory of Computing*, pages 216–226. ACM Press, 1978.

[Sch82] U. Schöning. A uniform approach to obtain diagonal sets in complexity classes. *Theoretical Computer Science*, 18:95–103, 1982.

[Sim70] C. C. Sims. Computational methods in the study of permutation groups. In J. Leech, editor, *Computational problems in abstract algebra*, Proc. Conf. Oxford 1967, pages 169–183, London, 1970. Pergamon.

[Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings 19th Symposium on Theory of Computations*, pages 77–82. ACM Press, 1987.

[vEB79] P. van Emde Boas. Complexity of linear problems. In *Proceedings of the Fundamentals of Computation Theory Conference*, pages 117–120, 1979.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity*. Springer, Berlin Heidelberg New York, 1999.

[Wra77] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1977.

[WW86] K. W. Wagner and G. Wechsung. *Computational Complexity*. Mathematics and its applications / East European ser. Reidel, 1986.

Notations

2^A	13	Power set $\{B : B \subseteq A\}$ of A.
A^*	13	Set of all words over the set A.
A^+	13	Set of all words over the set A with length ≥ 1.
A^c	13	Set of all words of length c over the set A where $c \in \mathbb{N}$ is a constant.
$\alpha[i]$	13	If $\alpha = (a_1, \ldots, a_n)$, then $\alpha[i] = a_i$ for $1 \leq i \leq n$.
AND	18	Boolean function "and" with $\text{AND}(x, y) = 1$ if and only if $x = y = 1$; formula: $x \wedge y$, $x \cdot y$, xy.
AND^m	42	Relation $\text{AND}^m = \{(a_1, \ldots, a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i = m\}$.
$\wedge^n$	76	Boolean function with $\wedge^n(x_1, \ldots, x_n) = 1$ if and only if $\Sigma_{i=1}^n x_i = n$.
$A \times B$	13	Cross product $\{(a, b) : a \in A \text{ and } b \in B\}$ of A and B.
$\text{AUDIT}(B)$	51	Auditing problem for B-circuits.
BF	31	Clone of all Boolean functions.
$\text{bin}(x)$	14	Binary encoding of x, i.e., $\text{bin}(x) = \text{bin}_n(x)$ where $n = \lfloor \log x + 1 \rfloor$ if $x > 0$ and $n = 1$ if $x = 0$.
$\text{bin}_n(x)$	14	The n-ary binary encoding of x, i.e., the word $w_{n-1} \ldots w_0 \in \{0,1\}^n$ with $\Sigma_{i=0}^n w_i 2^i = x$.
Ξ_k^{p}	60	If k is odd, then $\Xi_k^{\text{p}} = \Sigma_k^{\text{p}}$, and $\Xi_k^{\text{p}} = \Pi_k^{\text{p}}$ otherwise.
$[A]_\Omega$	15	Closure of A under Ω, i.e., the smallest set containing A and being closed under Ω.
$[B]_{+\text{fict}}$	77	Set B that is derived from the set of unary functions B by adding functions that are modified versions of those from B but with fictive variables.
$\text{co}\mathcal{K}$	22	$\text{co}\mathcal{K} = \{\bar{A} : A \in \mathcal{K}\}$
$\text{conc}(x, y)$	74	Concatenation function for binary inputs.
$\text{CF}(\Gamma)$	36	Set of relations that are describable by Γ-constraint formulas.
$\mathcal{C} \oplus \mathcal{K}$	22	$\mathcal{C} \oplus \mathcal{K} = \{A \triangle B : A \in \mathcal{C} \text{ and } B \in \mathcal{K}\}$
$\text{CSAT}(B)$	47	Satisfiability problem for B-circuits.
CSAT	47	Satisfiability problem for Boolean circuits.
CSP	37	Constraint satisfaction problem.
DIAG	28	Diagonalizaton of relation R with $\text{DIAG}(R) = \{(a_1, \ldots, a_{n-1}) : (a_1, \ldots, a_{n-1}, a_{n-1}) \in R\}$.
D_φ	14	Domain $\{x : \varphi(x) \text{ is defined}\}$ of φ.
DP	22	Second level of the Boolean hierarchy over NP: $\text{DP} = \text{NP} \oplus \text{NP}$.
$\text{dp}(A)$	52	The function dp builds a DP-complete set from an NP-complete set A: $\text{dp}(A) \stackrel{\text{def}}{=} (A \times \Sigma^*) \triangle (\Sigma^* \times A)$.
$\text{DSPACE}(\mathcal{K})$	21	Class of sets A for which there is a function $f \in \mathcal{K}$ such that A can be decided by a deterministic Turing machine with space f.

DTIME($\mathcal{K}$)	21	Class of sets A for which there is a function $f \in \mathcal{K}$ such that A can be decided by a deterministic Turing machine in time f.
dual(A)	34	Set $\{\text{dual}(f) : f \in A\}$, if A is a set of Boolean functions.
dual(f)	34	Dual function of f, i.e., $\text{dual}(f)(x_1, \ldots, x_n) = \neg f(\bar{x}_1, \ldots, \bar{x}_n)$.
DUP	42	Relation $\text{DUP} = \{0,1\}^3 - \{(0,1,0),(1,0,1)\}$.
D	34	Clone of all self-dual Boolean functions.
ECSP	38	Constraint satisfaction problem for existentially quantified constraints.
$\exists\text{FV}(B)$	50	Existential frozen variable problem.
$\text{enc}_{n,k}$	14	Bijection between $\{0, \ldots, k-1\}^n$ and $\{0, \ldots, k^n - 1\}$ with $\text{enc}_{n,k}(a_{n-1}, \ldots, a_0) = \sum_{i=0}^{n-1} a_i k^i$.
EQCF(Γ)	37	Set of relations describable by existential quantified Γ-constraint formulas.
EQ(B)	52	Circuit equivalence problem for B-circuits.
$C_1 \equiv C_2$	18	Two Boolean circuits C_1, C_2 are equivalent ($C_1 \equiv C_2$) if and only if they describe the same Boolean function.
$F_1 \equiv F_2$	37	Two constraint formulas F_1 and F_2 are equivalent ($F_1 \equiv F_2$) if and only if they describe the same relation.
EQ	18	Boolean equivalence function with $\text{EQ}(x,y) = 1$ if and only if $x = y$; formula: $x \leftrightarrow y$.
eq_A	13	For a set A is $\text{eq}_A = \{(a,a) : a \in A\}$, the equality relation on A.
EVEN^m	42	Relation $\text{EVEN}^m = \{(a_1, \ldots, a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i \text{ is even}\}$.
EXPTIME	21	Class of sets that can be decided deterministically in time $2^{\mathcal{P}}$, where $\mathcal{P}$ is the set of polynomials.
FCPS_B	73	Set of functions computable by circuits of polynomial size.
FDSPACE($\mathcal{K}$)	20	Set of functions f such that there is a $g \in \mathcal{K}$ and f is computable by a deterministic Turing machine with space g.
FDTIME($\mathcal{K}$)	20	Set of functions f such that there is a $g \in \mathcal{K}$ and f is computable by a deterministic Turing machine in time g.
FL	20	Set of functions that are computable in logarithmic space.
fpath	84	Set of paths leading to a leaf in a binary tree T.
FPSPACE	20	Set of functions that are computable in polynomial space and produce outputs of polynomial size.
FP	20	Set of functions that are computable in polynomial time.
f^{m*}	14	Given the m-ary function $f : A^m \to A$, the function $f^{m*} : (A^n)^m \to A^n$ is the coordinate-wise application of f on n-tuples.
$\mathcal{F}^n(A)$	13	Set $\{f : f : A^n \to A\}$ of n-ary functions on A.
$\mathcal{F}(A)$	13	Set $\{f :$ there is a n such that $f : A^n \to A\}$ of functions on A.
FV(B)	50	Frozen variable problem.
$f : A^n \to A$	13	An n-ary function from A^n to A.
f^{-1}	13	The inverse function of f.
GAPP_g	78	Set of functions all of which are in FP and have (g,k)-gaps.
$\text{GEN}_s(g)$	84	Generation problem for single carriers.
GEN(g)	84	Generation problem.
LVF	27	Identification of the last variables in a function. For an n-ary function f holds $\text{LVF}(f)(x_1, \ldots, x_{n-1}) = f(x_1, \ldots, x_{n-1}, x_{n-1})$.
id	14	Unary identity function with $\text{id}(x) = x$ for all x.

I_i^n	14	Short for $\mathrm{I}_i^{A,n}$, if A is clear from the context.
$\mathrm{I}_i^{A,n}$	14	The n-ary identity function on A at position i with $\mathrm{I}_i^{A,n}(a_1,\ldots,a_n) = a_i$ for all $a_1,\ldots,a_n \in A$.
$\mathrm{id}_k(x)$	74	Identity functions of arity n with $\mathrm{id}_k = \mathrm{I}_m^n$, where $m = \min\{k,n\}$.)
IMP	18	Boolean implication function with $\mathrm{IMP}(x,y) = 0$ if and only if $x = 1$ and $y = 0$; formula: $x \to y$.
FVR	28	Introduction of a fictive variable in a relation. For an n-ary relation R over A holds $\mathrm{FVR}(R) = \{(b,a_1,\ldots,a_n) : b \in A \text{ and } (a_1,\ldots,a_n) \in R\}$
$\mathrm{Inv}(B)$	28	Set $\{R :$ for all $f \in B$ holds $f \sim R\}$ of relations invariant to all functions from B.
$\mathrm{ipath}(T)$	84	Set of paths leading not to a leaf but to an inner node in the binary tree T.
$\bigsqcup Y$	15	Join operation in a lattice: The minimum of the set of upper bounds of Y.
$a \sqcup b$	15	Short for $\bigsqcup\{a,b\}$: The minimum of the set of upper bounds of $\{a,b\}$; the join of a and b.
$\mathcal{K}(k)$	22	The k-th level of the Boolean hierarchy over $\mathcal{K}$ with $\mathcal{K}(k) = \underbrace{\mathcal{K} \oplus \cdots \oplus \mathcal{K}}_{k}$
$\mathcal{L}(\mathcal{A})$	16	Lattice of algebras below the algebra A.
$\mathrm{ld}(i,n)$	66	Multiple applications of log: $\mathrm{ld}(0,n) = n$ and $\mathrm{ld}(i,n) = \lg \mathrm{ld}(i-1,n)$.
$\mathrm{Leaf}(T)$	84	Set of leaves of a binary tree T.
$\lvert w \rvert$	13	Length of word w.
$\lvert x \rvert$	14	Length of the binary representation of $x \in \mathbb{N}$, i.e., $\lvert x \rvert = \lvert \mathrm{bin}(x) \rvert$.
$\leq_{\mathrm{ae}}$	14	If $f(x) \leq g(x)$ for all but finite $x \in \mathbb{N}$, then $f \leq_{\mathrm{ae}} g$.
$A \leq_{\mathrm{m}}^{\log} B$	25	Logarithmic space, many-one reduction. $A \leq_{\mathrm{m}}^{\log} B$ if and only if there is an $f \in \mathrm{FL}$ such that for all x holds: $x \in A$ if and only if $f(x) \in B$.
$\leq_{\mathrm{m}}^{\mathrm{p}}$	25	Polynomial-time, many-one reduction. $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ if and only if there is an $f \in \mathrm{FP}$ such that for all x holds: $x \in A$ if and only if $f(x) \in B$.
$\leq_{\mathrm{m}}^{\mathrm{pp}}$	111	Reduction of disjoint pairs.
$\alpha \leq \beta$	34	If $\alpha,\beta \in \{0,1\}^n$, then $\alpha \leq \beta$ if and only if $\alpha[i] \leq \beta[i]$ for all $1 \leq i \leq n$.
$f \leq g$	14	If $f(x) \leq g(x)$ for all $x \in \mathbb{N}$, then $f \leq g$.
$\mathrm{lg}(x)$	66	Expansion of log to all natural numbers with $\mathrm{lg}(0) = 0$ and $\mathrm{lg}(x) = \log x$, if $x \geq 1$.
log	14	Logarithm $\log_2$ to the base of 2.
ltn_i^n	66	Tower of logarithms multiplied with n: $\mathrm{ltn}_i^k(n) = n \cdot \mathrm{lt}_i^k(n)$.
lt_i^n	66	Tower of logarithms with $\mathrm{lt}_i^k(n) = \mathrm{tp}_k(\mathrm{ld}(1,n),\mathrm{ld}(2,n),\ldots,\mathrm{ld}(i,n),1)$.
L	34	Clone of all linear Boolean functions.
L	21	Class of sets that can be decided deterministically in logarithmic space.
$l(q)$	84	Number of left turns in a path q to a node in a binary tree.
$\bigsqcap Y$	15	Meet operation in a lattice: The maximum of the set of lower bounds of Y.
$a \sqcap b$	15	Short for $\bigsqcap\{a,b\}$: The maximum of the set of lower bounds of $\{a,b\}$; the meet of a and b.
MOD_p	76	Boolean function with $\mathrm{MOD}_p = \{\neg\} \cup \bigcup_{n\in\mathbb{N}}\{\mathrm{mod}_p^n, \vee^n, \wedge^n\}$.
mod_p^n	76	Boolean function with $\mathrm{mod}_p^n(x_1,\ldots,x_n) = 1$ if and only if $\Sigma_{i=1}^n x_i \equiv 0 (\mathrm{mod}\ p)$.
mod_p	76	Boolean function with $\mathrm{mod}_p(y) = \mathrm{mod}_p^m(a_1,\ldots,a_m)$.

M	34	Clone of all monotonic Boolean functions.
NAE-3-SAT	39	Not all equal 3-SAT problem.
NAE^m	42	Relation $\mathrm{NAE}^m = \{0,1\}^m - \{(0,\ldots,0),(1,\ldots,1)\}$.
NAND	18	Boolean function "nand" with $\mathrm{NAND}(x,y) = 0$ if and only if $x = y = 1$; formula: $\mathrm{ND}(x,y)$.
NAND^m	42	Relation $\mathrm{NAND}^m = \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i \neq m\}$.
n-IN-m	42	Relation n-IN-$m = \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i = n\}$.
NL	21	Class of sets that are decidable nondeterministically with logarithmic space.
$\mathrm{Node}(T)$	84	Set of nodes of a binary tree T.
NOR	18	Boolean function "nor" with $\mathrm{NOR}(x,y) = 1$ if and only if $x = y = 0$; formula: $\mathrm{NR}(x,y)$.
NOR^m	42	Relation $\mathrm{NOR}^m = \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i = 0\}$.
NOT	18	Boolean negation function with $\mathrm{NOT}(x) = 1$ if and only if $x = 0$; formula: $\bar{x}$, $\neg x$.
NP	21	Class of sets that are decidable nondeterministically in polynomial time.
$\mathrm{NSPACE}(\mathcal{K})$	21	Class of sets A for which there is a $g \in \mathcal{K}$ such that A is decidable nondeterministically with space g.
$\mathrm{NTIME}(\mathcal{K})$	21	Class of sets A for which there is a $g \in \mathcal{K}$ such that A is decidable nondeterministically in time g.
ODD^m	42	Relation $\mathrm{ODD}^m = \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i \text{ is odd}\}$.
OR	18	Boolean function "or" with $\mathrm{OR}(x,y) = 0$ if and only if $x = y = 0$; formula: $x \vee y$.
OR^m	42	Relation $\mathrm{OR}^m = \{(a_1,\ldots,a_m) \in \{0,1\}^m : \Sigma_{i=1}^m a_i > 0\}$.
$\vee^n$	76	Boolean function with $\vee^n(x_1,\ldots,x_n) = 1$ if and only if $\Sigma_{i=1}^n x_i > 0$.
$\mathrm{path}(T)$	84	Set of paths leading to nodes in a binary tree T.
$\mathrm{Pol}(B)$	28	Set $\{f : f \sim R \text{ for all } R \in B\}$ of polymorphisms of B.
PE	27	Presence of the equality relation. A 0-ary operation on the set of relations with $\mathrm{PE} = \mathrm{eq}$.
PI	27	Presence of identity. A 0-ary operator on functions with $\mathrm{PI} = \mathrm{I}_1^2$.
$f \sim R$	15	For all $\alpha_1,\ldots,\alpha_n$ in the m-ary relation R holds $f^{m*}(\alpha_1,\ldots,\alpha_n) \in R$.
PRO	28	Projection of a relation. For an n-ary relation R over A holds $\mathrm{PRO}(R) = \{(a_1,\ldots,a_{m-1}) :$ there is a $b \in A$ such that $(b,a_1,\ldots,a_{m-1}) \in R\}$.
PSPACE	21	Class of sets that can be decided with polynomial space.
P	21	Class of sets that can be decided deterministically in polynomial time.
$\mathrm{QSAT}_k(\Gamma)$	60	Quantified Boolean constraint satisfaction problem.
R_0	34	Clone of all 0-reproducing Boolean functions.
R_1	34	Clone of all 1-reproducing Boolean functions.
$\mathcal{R}^n(A)$	13	Set $\{B : B \subseteq A^n\}$ of n ary relations over A.
$\mathcal{R}(A)$	13	Set $\{B :$ there is a n with $B \subseteq A^n\}$ of relations with finite arity over A.
$\mathrm{rgb}_g(k,\ell)$	77	Right gap bound in a gap-defining function.
$\mathrm{R}_{\mathrm{nae}}$	39	Relation $\mathrm{R}_{\mathrm{nae}} = \{0,1\}^3 - \{(0,0,0),(1,1,1)\}$.
$\mathrm{Root}(T)$	84	Root of a binary tree T.
RF	27	Rotation of variables in a function. For an n-ary function f holds $\mathrm{RF}(f)(x_1,x_2,\ldots,x_n) = f(x_2,\ldots,x_n,x_1)$.

RR	27	Rotation of variables in a relation. For an n-ary relation R is $\mathrm{RR}(R) = \{(a_2, \ldots, a_{n-1}, a_1) : (a_1, \ldots, a_{n-1}, a_n) \in R\}$.
$r(q)$	84	Number of right turns in a path q to a node in a binary tree.
$\mathrm{SAT}^*(B)$	48	Satisfiability with exception of $(1, \ldots, 1)$.
SAT	47	Satisfiability problem for propositional formulas.
$\mathrm{SelectSAT}(B)$	47	Satisfiability problem with selected exceptions.
SUBP	81	Set of functions computable in subpolynomial time.
SUB	27	Substitution of functions. If f is an m-ary function and g is n-ary function, then $\mathrm{SUB}(f,g)(x_1, \ldots, x_{m-1}, y_1, \ldots, y_n) = f(x_1, \ldots, x_{m-1}, g(y_1, \ldots, y_n))$.
$[A]$	27	Superposition closure $[A] = [A]_{\mathrm{SUP}}$.
$\langle B \rangle$	28	Relational superposition closure of B; $\langle B \rangle = [B]_{\mathrm{SUP_R}}$.
$\mathrm{SUP_R}$	28	Set of relational superposition operators $\mathrm{SUP_R} = \{\mathrm{PE}, \mathrm{RR}, \mathrm{TR}, \mathrm{FVR}, \mathrm{PRO}, \cap\}$.
SUP	27	Set of superposition operators $\mathrm{SUP} = \{\mathrm{PI}, \mathrm{RF}, \mathrm{TF}, \mathrm{LVF}, \mathrm{SUB}\}$.
tp	66	Short for tp_1.
$\mathrm{tp}_k(a_1, \ldots, a_n)$	66	Tower of Powers: $\mathrm{tp}_k(a_1, \ldots, a_i) = k a_1^{k a_2^{\cdots^{k a_i}}}$.
$C_1 \xrightarrow{M,*} C_2$	19	Turing machine M transforms configuration C_1 to configuration C_2 in a finite number of steps.
$C_1 \xrightarrow{M,n} C_2$	19	Turing machine M transforms configuration C_1 to configuration C_2 in n steps.
TF	27	Transposition of the last variables in a function. For an n-ary function f holds $\mathrm{TF}(f)(x_1, \ldots, x_{n-1}, x_n) = f(x_1, \ldots, x_n, x_{n-1})$.
TR	28	Transposition of the last variables in a relation. For an n-ary relation R holds $\mathrm{TR}(R) = \{(a_1, \ldots, a_n, a_{n-1}) : (a_1, \ldots, a_{n-1}, a_n) \in R\}$.
$\mathrm{U_L}$-uniform	24	A family of circuits $(C^n)_{n \in \mathbb{N}}$ is $\mathrm{U_L}$-uniform, if there is a function $f \in \mathrm{FL}$ such that $f(1^n) = C^n$ for all $n \in \mathbb{N}$.
$\mathrm{UNIQUE\text{-}SAT}(B)$	50	Unique satisfiability problem.
$w[i]$	13	If $w = w_1 \ldots w_n$ is a word, then $w[i] = w_i$ for $1 \leq i \leq n$.
Xor	18	Boolean function "xor" with $\mathrm{Xor}(x, y) = 1$ if and only if $x \neq y$; formula: $x \oplus y$.

Index