On the Space and Circuit Complexity of Parameterized Problems

Christoph Stockhusen

From the Institute of Theoretical Computer Science
of the Universität zu Lübeck
Director: Prof. Dr. math. Rüdiger Reischuk

On the Space and Circuit Complexity of Parameterized Problems

Dissertation
for the Fulfillment of
Requirements for the
Doctoral Degree
of the Universität zu Lübeck
from the Department of Computer Science

Submitted by

Christoph Stockhusen
from Hamburg

Lübeck, 2016

1

First referee	Prof. Dr. Till Tantau
Second referee	Prof. Dr. Heribert Vollmer
Date of oral examination	19. 01. 2017
Approved for printing	Lübeck, 24. 01. 2017

Preface

In this dissertation I present the core results from the research I did at the Institute of Theoretical Computer Science at the Universität zu Lübeck – something that I would never have thought of when I started my studies in Lübeck. My first contact with theoretical computer science was in the fourth semester. Outside the classroom, just before the first lecture, lots of students were gathering, more than usual. Soon we found out that many of them were actually in higher semesters, but they had to repeat the class since they did not pass the final exam the year before. Most of them were angry about this, telling us that "Einführung in die Informatik 4", the actual name of the class, was the hardest of the whole studies, the nearly invincible barrier that only few were able to overcome, held by the most demanding professor they ever met, filling the lecture with the most unnecessary stuff you can imagine. They did not fail to make us a little nervous. However, things turned out to be completely different. Prof. Rüdiger Reischuk gave one of the most interesting classes I had during my studies, challenging us with hard and demanding, but also addicting problems. What impressed me the most was something that none of the professors of the classes I previously attended was able to show us: The limits of human knowledge. During Prof. Reischuk's class we continuously walked on the border of what mankind knew and what mankind did not know. He presented us numerous questions that were easy to understand, but that nobody knew how to answer. This was something fascinating to me. Even more fascinating were all the theorems and researchers that pushed the boundary of knowledge that far as it was back then. In the remaining semesters I thus attended many of the classes given by the "theoreticians" of the university, not seldomly with only a handful of other interested students. When it came to writing my diploma thesis, I immediately knew that I wanted to write about theoretical computer science, especially complexity theory. Prof. Till Tantau from the institute had just published a strong paper, presenting logspace versions of the Theorems of Bodlaender and Courcelle, see Elberfeld et al. (2010), and he offered me the opportunity to write about new applications of these theorems. While working on this thesis, we started looking at these theorems from the perspective of parameterized complexity instead of classical computational complexity, and we noticed that there were no complexity-theoretic counterparts developed in this theory so far. This observation was the initial momentum of this thesis.

I am glad that I had the opportunity to do my part on pushing the border of human knowledge further and further. During the last years I have often been frustrated by the many drawbacks I have been going through, but now that I am writing this dissertation, I am happy and proud of my work. However, without the help of many, many people, this dissertation would not exist. In alphabetical ordering I am deeply thankful to my collegues Max Bannach and Michael Elberfeld for many fruitful discussions at numerous coffee breaks, Rüdiger Reischuk for the great introduction to theoretical computer science back then and the possibility to write this thesis at his institute, my parents Andree and Heike Stockhusen for giving me the opportunity to study and write this thesis, my advisor Till Tantau for all the lessons that I learned from him, and my collegue Oliver Witt for being a nice guy and always cheering me up.

Especially, I would like to thank my son Jonte and my wife Katharina for their love.

Christoph Stockhusen
Lübeck, 2016

Abstract

Parameterized complexity theory studies the computational complexity of computational problems from a multi-variate point of view: Instead of only measuring the amount of resources required to solve a problem relative to the size of the input, parameterized complexity also measures them relative to many other parameters of the input. Since its introduction, parameterized complexity theory has been a fruitful field, giving many important insights into the complexity of computational problems, especially by refining our knowledge about NP-complete problems. Being a relatively young field, however, parameterized complexity theory has not yet properly investigated two important aspects of complexity: space complexity and circuit complexity. While parameterized complexity theory mainly builds on time as the central resource, this thesis discusses the importance of space and circuit complexity in a parameterized setting.

After an introduction of natural parameterized space and circuit classes, we study their relation to parameterized time classes, and we will see that many parameterized problems can be better classified using parameterized space and circuit classes instead of parameterized time classes. Inspired by the Weft-Hierarchy, we study the concept of bounded nondeterminism with respect to space and circuits from a parameterized point of view, and we show that the resulting classes capture the complexity of natural problems like the associative generability problem exactly. Finally, we study classes of simultaneous time-space bounds, something that is not possible in classical computational complexity if we focus on well-established classes like polynomial time and logarithmic space, and we show that these classes capture exactly the complexity of natural problems like the longest common subsequence problem, answering a long-standing open problem of parameterized complexity.

Zusammenfassung

Die Parametrisierte Komplexitätstheorie beschäftigt sich mit der multivariaten Untersuchung der Berechnungskomplexität von algorithmischen Problemen: Statt die zur Lösung eines Problems benötigten Ressourcen ausschließlich in Abhängigkeit von der Eingabegröße zu messen, werden in der Parametrisierten Komplexitätstheorie weitere Parameter der Eingabe zur Messung herangezogen. Seit ihrer Vorstellung hat die Parametrisierte Komplexitätstheorie viele wichtige Einblicke in die Komplexität von Berechnungsproblemen ermöglicht, insbesondere in die Komplexität von NP-vollständigen Problemen. Zwei wichtige komplexitätstheoretische Aspekte wurden im relativ jungen Gebiet der Parametrisierten Komplexitätstheorie bisher jedoch nicht tiefergehend untersucht: Platzkomplexität und Schaltkreiskomplexität. Während die Parametrisierte Komplexitätstheorie bisher auf der Zeit als zentrale Resource aufbaut, wird in dieser Arbeit die Wichtigkeit von Platz- und Schaltkreiskomplexität für diese Theorie erörtert.

Nach einer Einleitung in parametrisierte Platz- und Schaltkreisklassen werden wir deren Beziehung zu den bisherigen Zeitklassen untersuchen und sehen, dass viele parametrisierte Probleme mithilfe von Platz- und Schaltkreisklassen besser klassifiziert werden können als mit Zeitklassen. Auf Basis der Weft-Hierarchie untersuchen wir weiter das Konzept des beschränkten Nichtdeterminismus' aus der Perspektive der Parametrisierten Komplexitätstheorie und zeigen, dass mit den davon abgeleiteten Klassen die Komplexität vieler natürlicher Probleme wie beispielweise dem assoziativen Generatorproblem formalisiert werden können. Zuletzt betrachten wir noch Klassen mit simultanen Zeit-Platz-Schranken, einem Konzept welches in der klassischen Komplexitätstheorie bei Betrachtung der verbreitetsten Komplexitätsklassen wie polynomieller Zeit oder logarithmischem Platz nicht sinnvoll ist. Wir zeigen dann, dass diese Klassen genau die Komplexität natürlicher Probleme wie dem Problem der längsten gemeinsamen Teilsequenz formalisieren, ein Ergebnis welches eine lange unbeantwortete Frage der Parametrisierten Komplexitätstheorie beantwortet.

Contents

1. Introduction

1.1. My Thesis

Parameterized space and circuit complexity are fundamental and essential concepts of parameterized complexity theory. Parameterized space and circuit classes give us the ability to understand the reasons behind parameterized tractability and intractability results in a deeper way than classes of parameterized time complexity alone because many important problems are deeply connected to the concepts of space and parallel computation. Therefore, to gain a full understanding of the complexity of parameterized problems, it is necessary to study – in addition to their time complexity – their space and circuit complexity.

The focus of the study of the complexity of computational problems in both classical as well as parameterized complexity theory has always lain on the time required to solve a problem. The space or circuit complexity required to solve it has often been considered as being of scholarly interest, at best. In the context of classical computational complexity theory, there may be three reasons for this:

1. The classes of problems solvable in deterministic or nondeterministic logarithmic space are contained in the class of polynomial time, and most of these problems can be solved practically by very fast polynomial time algorithms that make use of a polynomial amount of space. The additional requirement to only use a logarithmic amount of space to solve such problems often results in algorithms whose runtime is much higher than the runtime of algorithms that are allowed to use a polynomial amount of space, thus yielding algorithms that are not considered practically relevant. Therefore, the study of logarithmic space is – even in the field of theoretical computer science – often regarded as being less important than the study of polynomial time.
2. The class of problems solvable in polynomial space contains a huge diversity of problems: problems that are solvable very fast and problems that

are highly intractable, like the class of NP-complete or PSPACE-complete problems. Therefore, the criterion that a problem is solvable via an algorithm requiring a polynomial amount of space does not give us any information about whether this problem is solvable efficiently in terms of the time required to solve it. Only the information that a problem is hard or complete for the class of polynomial space shows us that this problem can be considered intractable, but this insight can also be derived from showing that the problems is hard for NP.

3. Circuits are mainly used for proving lower bounds. While this is clearly an important task when studying the computational complexity of problems, circuit complexity had only a short time in limelight because the vast majority of lower bounds that could be proved using circuits are all inside P, thus not showing that a problem is intractable from the perspective of time. On the other hand, designing circuits to solve computational problems and, hence, prove their tractability, can be a tedious task, while showing that a problem can be solved in polynomial time is comparatively easy.

The fact that the study of space and circuit complexity is not in vogue in classical computational complexity theory carried over to the world of parameterized complexity theory: Parameterized complexity theory as far as it has been developed by Downey and Fellows is based on the concept of polynomial time. Its fundamental class FPT is a class of parameterized polynomial time. This also holds for classes like para-NP and XP, and even the classes of the Weft-Hierarchy are based on the concept of polynomial time since they are defined as the FPT-closure of weighted satisfiability problems. I believe that relying on polynomial time as the only underlying notion of parameterized tractability, and, moreover, using time classes to define parameterized intractability is insufficient. In this thesis I want to convince you that space and circuit complexity must be studied in order to complement parameterized complexity theory to an even more fruitful theory, explaining the intriguing world of problems, computations, and complexity.

1.2. Results

In this dissertation I show that parameterized space and circuit complexity are indeed a reasonable and fruitful complementation of “traditional” parameterized complexity theory. For this, the dissertation contains two main contributions:

1. I develop a framework that extends the existing one of the theory of parameterized complexity. It contains parameterized complexity classes and appropriate reduction notions that augments the hitherto existing framework of parameterized complexity from the perspective of parameterized space and circuits.
2. I apply these classes to natural parameterized problems. This gives more insights into their complexity and, at best, allows to capture the complexity of problems that were not exactly classifyable beforehand.

More detailed, I present the following results: I generalize the relaxed notion of polynomial time, i. e., of FPT to relaxed notions of classical circuit and space classes, yielding classes like para-L, para-NL, para-AC^0, para-TC^0, para-NC^1, etc., which are all subclasses of FPT. For these classes I show that they allow a refined view on the complexity of natural problems that have only been classified as "contained in FPT" beforehand. The main result here is that p-VERTEX-COVER lies in para-AC^0, i. e., it lies in a very small subclass of FPT and, moreover, admits a highly parallel algorithm running in parameterized constant time. I also show that the parameterized feedback vertex set problems for directed and undirected graphs, p-DFVS and p-FVS, respectively, do not admit such algorithms, i. e., both are provably not in para-AC^0. Moreover, I show that we can use para-classes to obtain lower bounds that are related to questions from classical computational complexity theory by showing that p-FVS $\in$ para-NC^1 implies $NC^1 = L$, and p-DFVS $\in$ para-L implies $L = NL$.

A second way to relax the notion of tractability is to use X-classes. In analogy to the systematical introduction of para-classes, I introduce the X-classes XL, XNL, XAC^0, XTC^0, XNC^1, etc., and I show how these classes provide an even more detailed view on the complexity of parameterized problems. For example, I show that p-VERTEX-COVER $\in XAC^0$ and p-CLIQUE $\in XAC^0$, but that p-FVS $\notin XAC^0$ and p-DFVS $\notin XAC^0$.

Alongside with the study of computational complexity using complexity classes always comes the need for appropriate reduction notions, thus I discuss para-AC^0-, para-L-, and para-P-reductions. While these reductions are clearly relevant for the classes above, I show that they are very important for the Weft-Hierarchy, for which I give a critical review in this thesis. Most importantly, I show that there is no para-AC^0-reduction from p-DFVS to p-CLIQUE, which is suprising because p-DFVS $\in$ FPT, p-CLIQUE $\in$ W[1], and FPT $\subseteq$ W[1], and from classical computational complexity theory we know that most classes are closed under very weak reductions. I thus discuss the importance of the Weft-Hierarchy and suggest an alternate view on para-classes and Weft-classes as notions of parameterized tractability or intractability when studying parameterized problems.

Following the discussion of the Weft-Hierarchy, I combine the notion of bounded nondeterminism with the notion of para-classes I introduced before, yielding classes of read-once bounded nondeterminism like para_β-L where we allow to read a bounded amount of nondeterministic bits only once and classes of read-again bounded nondeterminism like para_W-L where we allow to read a bounded amount of nondeterministic bits arbitrarily often. Concerning read-again bounded nondeterminism, I show that three types of problems, p-FAMILY-UNION-A, p-SUBSET-UNION-A, and p-WEIGHTED-UNION-A for languages A are deeply connected to these classes: First of all, p-FAMILY-UNION-A is complete for a class para_W-C if the underlying language A is complete for C under a certain reduction notion which is both natural and in fact often used in practice. Secondly, we can often reduce p-FAMILY-UNION-A to p-SUBSET-UNION-A as well as p-SUBSET-UNION-A to p-WEIGHTED-UNION-A quite easily, and all of these problems lie in para_W-C. I also show that many important problems like the weighted satisfiability problem p-WEIGHTED-SAT, the reachability problem in edge-colored directed gaphs p-COLORED-REACH and undirected graphs p-COLORED-UNDIRECTED-REACH, or the associative generability problem p-AGEN can be expressed as such union problems and, thus, are complete for classes of bounded nondeterminism (p-WEIGHTED-SAT is complete for $\text{para}_W\text{-NC}^1$, p-COLORED-REACH is complete for para_W-NL, p-COLORED-UNDIRECTED-REACH is complete for para_W-L, and p-AGEN is complete for para_W-NL). Moreover, I use classes of read-again bounded nondeterminism for proving additional lower and upper bounds for the feedback vertex set problem: p-FVS $\in \text{para}_W$-L, p-DFVS $\in \text{para}_W$-NL, p-FVS $\in \text{para}_W\text{-NC}^1$ if, and only if, $\text{NC}^1 = \text{L}$, and p-DFVS $\in \text{para}_W$-L if, and only if, $\text{L} = \text{NL}$. Concerning read-once bounded nondeterminism, I discuss variants of the parameterized distance problem p-DISTANCE. The main results here are that both p-DISTANCE and p-UNDIRECTED-DISTANCE are complete for para_β-L, p-LARGE-DISTANCE is complete for para-NL, the undirected version p-UNDIRECTED-LARGE-DISTANCE is complete for $\text{para}_{\beta\forall}$-L, a variant of para_β-L but with universal instead of existential nondeterminism, p-EXACT-DISTANCE is complete for $\text{para}_{D\beta}$-L, a parameterized logspace variant of DP, and that p-LONGEST-PATH is complete for para_β-L.

Finally, I study simultaneous time-space classes, especially para-P/XL, the class of simultaneous “para-P-time” and “XL-space”, and para-NP/XNL, the nondeterministic variant of para-P/XL. I show that these classes capture the complexity of several natural problems, forming a reduction chain that leads to the main result that the longest common subsequence problem p-LCS is complete for para-NP/XNL, thus finally settling a long-standing open problem. Furthermore, I show that p-FVS $\in$ para-P/XL, and if p-DFVS $\in$ para-P/XL, then L $=$ NL, two non-trivial upper and lower bounds for the feedback vertex set problem.

I published most of these results in Elberfeld et al. (2012), Stockhusen and Tantau (2013), Elberfeld et al. (2015), and Bannach et al. (2015).

1.3. Organisation of this Thesis

My dissertation consists – besides this introductory chapter and the conclusion – of three main chapters. Chapter 2 introduces the basic notions that will be used in the subsequent chapters, and, should be read before them. Chapters 3 and 4 are, as far as possible, independent from each other and can be read separately.

Parameterized Space and Circuits

Chapter 2 reviews the basics of parameterized complexity theory and connects them with the concepts of space and parallelism using circuits. Space and circuit analogues of the fundamental parameterized time classes FPT, para-NP, and XP are defined, and basic properties and relations of them are proved. Moreover, first problems like the vertex cover problem and the feedback vertex set problem are studied with respect to these classes. A reduction notion for these classes is given under which all the classes of this dissertation are closed, and, finally, the Weft-Hierarchy is reviewed from the perspective of parameterized space and time.

Bounded Nondeterminism

Chapter 3 is devoted to parameterized computations using a bounded amount of nondeterminism. The classes of the preceeding chapter are combined with this notion, and natural complete problems for these classes are presented, namely the weighted satisfiability problem, the reachability problem in edge-colored graphs, the associative generability problem, and several variants of distance problems. Futhermore, this section investigates problems that are not complete for these classes like the feedback vertex set problem, but that can be separated from each other using classes of bounded nondeterminism.

Simultaneous Time-Space Classes

Chapter 4 interweaves the concepts of time and space by studying simultaneous time-space classes. While in classical computational complexity the study of simultaneous time-space bounds with respect to fundamental classes like polynomial time or logarithmic space does not make much sense, we will see why the situation is different in the setting of parameterized complexity theory. Completeness results for these classes are given, for example for the the longest common subsequence problem that could not be classified exactly before, when considering only time or only space.

2. Parameterized Space and Circuits

Computational complexity theory is the part of computer science that studies reasons for why computational problems are easy or hard to solve for computers, that is, why some problems can be solved by very simple computers in fractions of seconds while others require days or weeks or longer even on huge supercomputers. A central approach for this study is to *measure* and *compare* the resources used by algorithms that solve problems. The standard way of measuring is to use functions that map the size of the input to the required amount of the resource. Using this approach, scientists have compared and *classified* a huge variety of computational problems. However, the approach of measuring the complexity of a problem in terms of the input size has its disadvantages. For example, consider the vertex cover problem:

Input: An undirected graph G with $G = (V, E)$ and a natural number k.

Question: Is there a set C with $C \subseteq V$ and $|C| \leq k$ such that every edge of the graph is incident to at least one vertex of C, i. e., for every edge $\{u, v\}$ of the graph we have $|C \cap \{u, v\}| > 0$?

Karp (1972) showed that the vertex cover problem is complete for the complexity class NP of problems solvable in nondeterministic polynomial time. Up to now, it is unknown whether these problems can be solved in deterministic polynomial time, thus the known deterministic algorithms for these problems require an exponential amount of time with respect to the size of the input, which is unacceptable in practice. But this is not the end of the road. Downey and Fellows (1995a) introduced the concept of *fixed-parameter tractability* based on an idea that is described in Downey and Fellows (1997) as a "deal with the devil": They regard algorithms as acceptable when the runtime of the algorithm is polynomial with respect to the input size and only some aspects, the so-called *parameters* of the problem, contribute in a "devilish" way, for example exponential or even worse. The hope with this approach is that the "devilish" parameters do not get very large in real-life applications. An example for a problem that admits such an algorithm is the above-mentioned vertex cover problem where we consider the size of the desired vertex cover as the parameter. Downey and Fellows (1995c) showed that this problem can be solved in time $O(2^k \cdot n)$. Downey and Fellows (1995a) also observed that not every graph problem seems to admit such algorithms, for instance the clique problem:

Instance: An undirected graph G with $G = (V, E)$ and a natural number k.

Question: Is there a set C with $C \subseteq V$ and $|C| = k$ such that between every two vertices from C there is an edge, i. e., for every u and v with $u \neq v$ and $u \in C$ and $v \in C$ we have $\{u, v\} \in E$?

To capture the intractability of such problems in terms of the relaxed and "devilish" notion of tractability, Downey and Fellows (1995a) introduced the so-called *Weft-Hierarchy* or *W-Hierarchy*, a hierarchy of classes defined in terms of reduction closures of circuit satisfiability problems. They showed that the clique problem is complete for the first class W[1] of this hierarchy if parameterized via the size of the desired clique. Even though it is still unknown whether this rules out the fixed-parameter tractability of the clique problem, it is still a strong hint into this direction because the fixed-parameter tractability of a W[1]-hard problem would immediately answer long-standing open questions like the one of whether the strong exponential time hypothesis holds, see Chen and Meng (2008).

Besides these classes, Downey and Fellows (1997) and Flum and Grohe (2006) defined further classes of parameterized intractability, and classified, together with numerous other researchers, hundreds of problems as fixed-parameter tractable or intractable. Despite this huge success of parameterized complexity theory, the theory had two major problems: First, many problems that have been classified as fixed-parameter tractable seemed to actually have different complexities, which was not reflected by the theory. While for some of them only algorithms with huge runtimes were known, others admitted algorithms that were extremely fast. Back then, it was unknown for many problems whether the explanation of these discrepancies was the lack of efficient algorithmic techniques showing that these problems do not have different complexities or a lack of the framework to appropriately formalize the different complexities. Second, for many parameterized problems it was known that they are hard for, say, W[1], but, despite all effort, it was not possible to show that they are complete for any class of parameterized intractability. Again, it was unknown whether the reason for this was the lack of adequate proof techniques or a lack of expressibility of the theory. In this thesis we will see that, as a matter of fact, in many cases the expressibility of the framework is the reason for the problems described above and that parameterized space and circuit classes are reasonable extensions of parameterized complexity theory that help to overcome these problems. Thus, the rest of this chapter is a review of the basics of parameterized complexity theory interwoven with a presentation of parameterized space and circuit classes that I derived from their time-based companions. Section 2.1 reconsiders the concept of parameterized problems in the context of space and circuit complexity. Section 2.2 defines space and circuit analogues of the class FPT of fixed-parameter tractable problems and XP of slice-wise polynomial-time solvable problems. Section 2.3 discusses reduction notions appropriate for the classes presented in this section.

2.1. Parameterized Problems

The core idea of parameterized complexity is to not only measure the complexity of a problem in terms of the algorithmic resource consumption with respect to the size of the problem input, but to refine this measure by also considering different parameters of the problem. For instance, think of the vertex cover problem from above. In order to refine our algorithm analysis by measuring the resource consumption with respect to the graph size and the parameter k, we make use of the concept of parameterized problems.

▷ *Definition 1 (Parameterized Problems).* A *parameterized problem* consists of a tuple (Q, κ) where Q with $Q \subseteq \Sigma^*$ for an alphabet Σ is a language and $\kappa\colon \Sigma^* \to \mathbb{N}$ is a function, the so-called *parameterization*, that maps instances to its *parameter values*.

The parameterized version of the vertex cover problem thus can be stated as follows:

▷ *Problem 2 (Parameterized Vertex Cover).*

Input: An undirected graph G with $G = (V, E)$ and a natural number k.

Parameter: k.

Question: Is there a set C with $C \subseteq V$ and $|C| \leq k$ such that every edge of the graph is incident to at least one vertex of C, i. e., for every edge $\{u, v\}$ of the graph we have $|C \cap \{u, v\}| > 0$?

To distinguish between the parameterized and unparameterized versions of problems, it is common to prepend the name of the problem with "p-"

in the parameterized case. Hence, let us denote the parameterized version of the problem VERTEX-COVER by p-VERTEX-COVER. While in the case of p-VERTEX-COVER the parameterization via k is quite natural, for many other problems there exist several natural parameterizations. For example consider the problem of computing longest common subsequences:

▷ *Problem 3 (Longest Common Subsequence).*

Instance: A set S of strings over an alphabet Σ and a natural number l.

Question: Is there a string $s \in \Sigma^l$ such that s is a common subsequence of the strings in S, i. e., from every s' with $s' \in S$ we can obtain s by only deleting symbols from s'.

For example, given the set S with

$$S = \{\mathsf{abcabcabc}, \mathsf{cbacbacba}, \mathsf{aaabbbccc}\},$$

there is a common subsequence s of length 3, for example aaa, but no common subsequence of length 4.

The longest common subsequence problem has several natural parameterizations: The length l of the subsequence asked for, the size $|S|$ of the given set of strings, the size $|\Sigma|$ of the alphabet, or combinations of these parameters. If it is not clear from the context, I will denote the parameterization used as the index of the prefix "p-". For example, I will denote the different parameterizations of the longest common subsequence problem from above by p_l-LCS, $p_{|S|}$-LCS, $p_{|\Sigma|}$-LCS, and so on, respectively.

As can already be seen from the above example of the longest common subsequence problem, the parameter values of an instance do not necessarily have to be given explicitly with the instance like in the case of p-VERTEX-COVER, but can come along with the input rather implicitly. However, many algorithms require that the parameter value of an instance is known in order to work correctly. Hence, if the parameter is not given explicitly with the input, algorithms must be able to compute it. While Downey and Fellows (1995a) only considered parameterized problems where the parameter is explicitly given along with the input or can be computed somehow from it, Flum and Grohe (2006) demanded that the parameterization is computable in polynomial time, i. e., there is a polynomial-time bounded Turing machine that computes the binary representation of the parameter value from the input instance. However, since this thesis discusses results on parameterized complexity classes that have their origin in classical complexity classes that lie deep in P, we have to impose even stronger restrictions and, thus, augment our definition from above: For the rest of this thesis, we require that the parameterization is first-order computable or, equivalently, computable by logarithmic-time uniform AC^0-circuits. For an introduction into first-order computations see the textbook of Immerman (1999), for an introduction into circuit complexity see the textbook of Vollmer (1999).

2.2. Para-Classes and X-Classes

The central idea of parameterized complexity is to use a relaxed notion of tractability: Instead of requiring that a problem has to be solvable in polynomial time, this relaxed notion demands that the runtime has to be polynomial in the overall input size, but may be worse than polynomial in the problem parameter. The hope is that for real-life instances the parameter values are small and therefore the overall runtime behaves polynomial. This idea has been formalized by Downey and Fellows (1997), thus shaping the notion of *fixed-parameter tractability*:

▷ *Definition 4 (FPT).* A parameterized problem (Q, κ) lies in the class *FPT* of *fixed-parameter tractable problems* if there is an algorithm deciding $x \in Q$ for any instance x in time $f\big(\kappa(x)\big) \cdot p\big(|x|\big)$ where $f \colon \mathbb{N} \to \mathbb{N}$ is an arbitrary function, p is a polynomial, and $|x|$ is the size of the input x.

One of the most famous problems that lies in FPT is p-Vertex-Cover. A rather simple algorithm showing that this problem is fixed-parameter tractable uses the observation that for every edge of the graph at least one of its incident vertices has to be in the vertex cover. Hence, on input of an undirected graph G with $G = (V, E)$ and a parameter k, we make use of a binary search tree of depth k where every node represents a partial vertex cover. The root of the tree is the empty set. We then append children recursively to the nodes of the tree: If the partial vertex cover is C, we pick an edge $\{u, v\}$ of $G - C$ which denotes the graph G with all vertices of C and its incident edges removed. Since at least one of the vertices u and v has to be in the vertex cover, we append two children to the current node of the search tree: One with label $C \cup \{u\}$ and one with label $C \cup \{v\}$. If in the search tree of depth at most k we find a vertex labeled with C such that $G - C$ has no edges, we have found a vertex cover. Otherwise, there is no vertex cover of size k. Since the search tree is binary and has depth at most k, it has at most 2^k nodes. For every node of the search tree with label C we have to compute $G - C$ and find an edge of the remaining graph, which can be done in polynomial time. Overall, the algorithm on input x then has the runtime $O\big(2^{\kappa(x)} \cdot p(|x|)\big)$ for a polynomial p, showing the fixed-parameter tractability of p-VERTEX-COVER.

The vertex cover problem is not the only fixed-parameter tractable problem. To the present day, literally hundreds of problems have been shown to be fixed-parameter tractable. However, breaking the "FPT-barrier" usually marks the end of the structural complexity-theoretic analysis of a problem; the subsequent studies of a problem after showing its fixed-parameter tractability typically focus on improving the runtime of the corresponding algorithms. In classical complexity theory, however, showing that a problem is tractable, i. e., showing that the problem is solvable in polynomial time immediately rises the question whether we can show that the problem is actually solvable in nondeterministic or deterministic logarithmic space or with even less computational power. In the same way we can ask in the context of parameterized complexity if we can do better than "only" showing that a problem is fixed-parameter tractable. This motivates the study of parameterized complexity classes that are subclasses of FPT. To formalize and study such classes, we make use of a generalization of the relaxed tractability notion given by Flum and Grohe (2003):

▷ *Definition 5 (Para-Classes).* Let C be a classical complexity class. Then, para-C is the class of parameterized problems (Q, κ) with $Q \subseteq \Sigma^*$ for an alphabet Σ such that there exists an alphabet Π, a computable function $\pi\colon \mathbb{N} \to \Pi^*$, and a language X with $X \subseteq \Sigma^* \times \Pi^*$ such that $X \in C$ and for every instance x of (Q, κ) we have $x \in Q$ if, and only if, $\big(x, \pi(\kappa(x))\big) \in X$.

More intuitively, Flum and Grohe (2003) defined para-C as the class of parameterized problems that *are in* C *after a precomputation that only depends on the parameter*, i. e., a problem (Q, κ) lies in para-C if, and only if, there exists an algorithm working in two stages: In the first stage, the algorithm does an arbitrary complex computation based on the parameter value. In the second stage, the algorithm does a computation whose resource consumption is bound due to C that decides the problem on the remaining instance.

A classical example for an algorithm using the concept of precomputation on the parameter is the model-checking problem of strings and monadic second-order formulas:

Instance: A logical structure S from the class of strings together with a monadic second-order formula φ of an appropriate vocabulary.
Parameter: $|\varphi|$.
Question: Is S a model for φ, i. e., $S \models \varphi$?

Büchi (1960) showed that a language L is regular if, and only if, it is definable in monadic second-order logic, i. e., the fragment of second-order logic where we restrict the second-order variables to have arity 1. Moreover, Büchi showed that, given a monadic second-order sentence φ, a nondeterministic finite automaton A with $L(\varphi) = L(A)$ can be computed and vice versa. With this observation we can solve the problem above using an algorithm that makes a precomputation on the parameter: In the first stage, the given sentence φ is translated into a corresponding nondeterministic automaton A' using Büchi's Theorem, which is then transformed into an equivalent deterministic automaton A such that $L(\varphi) = L(A)$. In the second stage, this automaton is simulated on the input string encoded in the given logical structure and the input instance is accepted if, and only if, the automaton accepts the input string. While the only known algorithms for the generation of a deterministic finite automaton A for a given sentence φ with $L(A) = L(\varphi)$ generates automata of size more than superexponential in $|\varphi|$, the simulation of the resulting automaton requires only polynomial time. Hence, the first stage of the algorithm above is a precomputation on the parameter, and the second stage runs in polynomial time after the precomputation. Altogether, this algorithm shows that the problem lies in para-P.

Using the view of precomputations on the parameter, Flum and Grohe (2003) showed that the class FPT is exactly the class of problems that lie in P after a precomputation on the parameter, thus underlining the naturalness of their definition:

▷ *Fact 6 (Flum and Grohe (2003)).* FPT = para-P.

If we, instead of polynomial time, insert natural subclasses of P like (nondeterministic) logarithmic space or circuit classes into the definition above, we get parameterized complexity classes with the following properties (for better readability let us for the rest of this thesis abbreviate $\kappa(x)$ with k, $f(\kappa(x))$ with f_k, and $|x|$ with n):

para-L The class of languages that are decidable via a deterministic Turing machine that uses at most $O\big(f_k + \log(n)\big)$ many read-write cells.

para-NL The class of languages that are decidable via a nondeterministic Turing machine that uses at most $O\big(f_k + \log(n)\big)$ many read-write cells.

para-AC^i The class of languages that are decidable via family of circuits over the standard base, with unbounded fan-in, size $O\big(f_k \cdot p(n)\big)$ for some polynomial p, and depth $O\big(f_k + \log^i(n)\big)$ if $i > 0$ and depth $O(1)$ if $i = 0$.

para-TC^{i} The class of languages that are decidable via family of circuits over the standard base together with threshold gates, with unbounded fan-in, size $O\big(f_k \cdot p(n)\big)$ for some polynomial p, and depth $O\big(f_k + \log^{i}(n)\big)$ if $i > 0$ and depth $O(1)$ if $i = 0$.

para-NC^{i} The class of languages that are decidable via family of circuits over the standard base, with bounded fan-in, size $O\big(f_k \cdot p(n)\big)$ for some polynomial p, and depth $O\big(f_k + \log^{i}(n)\big)$.

On first sight, one would correctly expect that the space bound and the depths of the circuits should be of the form $O\big(\log^{i}(f_k + n)\big)$ because we desire a logarithmic amount of space or logarithmic depth after a preprocessing on the parameter, but basic calculus shows that we have $O\big(\log^{i}(f_k + n)\big) = O\big(f'_k + \log^{i}(n)\big)$ for $i > 0$, and the second form reflects the properties of the circuits in a more intuitive form.

Before we start investigating the structural properties of these classes, let us briefly discuss the circuit classes above from the perspective of parallelism. Circuit classes like AC and NC capture the notion of problems that admit fast parallel algorithms. The idea behind this is that the depth of the circuit corresponds to the parallel time and the size of the circuit corresponds to the parallel work required to solve a problem. Thus, a problem solvable via AC^1 circuits is solvable in parallel time in $O\big(\log(n)\big)$ and polynomial work. This carries over to parameterized circuit classes: A problem solvable in para-AC^1 is solvable in parameterized parallel time $O\big(f_k + \log(n)\big)$ and parameterized parallel work $O\big(f_k \cdot p(n)\big)$ for a polynomial p. For more details on the connections between circuits and parallelism see Vollmer (1999).

From the definition of para-classes we can directly conclude that they inherit their inclusion structure from the underlying classical complexity classes, i. e., we have for two classes C and C' that $C \subseteq C'$ if, and only if, para-$C \subseteq$ para-C', and $C \subsetneq C'$ if, and only if, para-$C \subsetneq$ para-C'. Hence, we get the inclusion chain

$$\begin{array}{llll} & \text{para-AC}^0 \subseteq \text{para-TC}^0 & \subseteq \text{para-NC}^1 & \\ \subseteq \text{para-L} & \subseteq \text{para-NL} & & \\ \subseteq \text{para-AC}^1 & \subseteq \text{para-TC}^1 & \subseteq \text{para-NC}^2 & \\ \subseteq \text{para-AC}^i & \subseteq \text{para-TC}^i & \subseteq \text{para-NC}^{i+1} & \quad \text{with } i > 1 \\ \subseteq \text{para-P} & \subseteq \text{para-NP} & \subseteq \text{para-PSPACE}, & \end{array}$$

with the known proper inclusions

$$\text{para-AC}^0 \subsetneq \text{para-TC}^0, \qquad \text{para-NL} \subsetneq \text{para-PSPACE}.$$

Equipped with these classes, we can continue our investigation of the structural complexity of parameterized problems inside FPT. This has partially already been done by Cai et al. (1997). They investigated p-VERTEX-COVER and showed that p-VERTEX-COVER lies in para-L, although they used a different definition of para-L and probably were not aware of the inclusion chain presented above. For their result, they used a technique, that is now known as *kernelization*. The main idea behind kernelization is to turn the idea of a precomputation on the parameter upside down: In the first stage, a fast algorithm is used on the input instance, computing a *kernel* which is an equivalent instance whose size only depends on the parameter. Then, in the second stage, an algorithm with a possibly much worse runtime is run on the kernel.

▷ *Definition 7 (Kernelization).* A *kernelization* for a parameterized problem (Q, κ) is an algorithm K that, on input x, computes an instance $K(x)$, the *kernel*, such that $|K(x)| \leq f\big(\kappa(x)\big)$ for an arbitrary function f and $x \in Q$ if, and only if, $K(x) \in Q$.

One of the most well-known kernelizations is Buss' kernelization[1] for the vertex cover problem. Buss noticed that if a graph G has a vertex cover of size k, then any vertex v with at least $k + 1$ adjacent vertices has to be in the vertex cover, since otherwise all of v's neighbors have to be in the cover, immediately exceeding the maximal allowed size k of the vertex cover. Moreover, if a graph has only vertices with degree at most k and more than k^2 edges, then it has no vertex cover of size k, because every selection of k vertices can, due to the low degree of at most k, cover at most k^2 edges. We can apply these observations as follows: On input of a graph G and a natural number k, we search for the first vertex with degree larger than k. Since this vertex has to be in the vertex cover, we remove this vertex and its incident edges from the graph (let us call the resulting graph G') and repeat the procedure for G' and the new parameter value $k - 1$ because it remains to find a vertex cover of size $k - 1$ for the remaining edges. We continue this process until we either have that the graph still has edges and the parameter is 0, or we obtain a graph H and a parameter l such that no vertex of H has degree larger than l. In the first case we output a graph that consists of two vertices that are connected with an edge and the parameter 0, i. e., an instance that has no solution. In the second case we remove all isolated vertices from H because there is no need to add them to the vertex cover, and then we check if the number of remaining edges

[1]This observation has never been actually published, it only was mentioned by Buss and Goldsmith (1993), but since then it is known as Buss' kernelization

exceeds l^2. If this is the case, there is no vertex cover for the graph and we, again, output the instance without a solution mentioned above. If the graph has less than l^2 edges, then we output H and l.

We can now observe that the algorithm above outputs a graph H and a parameter l such that H has a vertex cover of size l if, and only if, the original input graph G has a vertex cover of size k. Moreover, the size of the resulting graph H is bounded by a function that only depends on k: If the fixed instance without a solution is output, this is clearly the case. If a possibly solvable instance H and l is computed, then H has no isolated vertices, at most l^2 edges, and, thus, at most $2 \cdot l^2$ vertices. The size of this instance is therefore bounded by a function that only depends on l with $l \leq k$. Altogether, the algorithm above is a kernelization algorithm for vertex cover.

Kernelizations play an important role in parameterized complexity theory because there is a deep connection between polynomial-time computable kernelizations and fixed-parameter tractability: Niedermeier (2002) showed that a problem is fixed-parameter tractable if, and only if, it admits a polynomial-time computable kernelization. Thus, the example above gives us another proof of the fixed-parameter tractability of p-VERTEX-COVER.

Above, we generalized the concept of fixed-parameter tractability to other complexity classes using para-classes, which is reasonable from the perspective of kernelization as the following lemma shows:

$\triangleright$ *Lemma 8.* Let (Q, κ) be a parameterized problem and C be one of the complexity classes AC^i, NC^i, TC^i, L. Then, $(Q, \kappa) \in$ para-C if, and only if, (Q, κ) has a kernelization that is computable within the resource bounds defined by C, i. e., using AC^i-, NC^i-, TC^i-circuits, or in logarithmic space, respectively.

Proof Idea. Instead of proving this lemma for the classes mentioned above, let us take a look at the technique that can be used to prove this lemma (the proof itself is then straight-forward).

If the problem (Q, κ) lies in para-C via an algorithm A, we construct a kernelization: On input x, compute the value of the parameter and compare it to the input size:

- If the value of the parameter is "large", then the parameter dominates the input size and, therefore, the input is already the kernel.
- If the value of the parameter is "small", we can use A to compute the answer using resources that are only bound in terms of the input size. Therefore, we compute the solution using A, and output, depending on

the solution, fixed positive or negative instances of the decision problem Q, whose sizes are clearly bound by the parameter.

For example, assume that we have $(Q, \kappa) \in$ para-L via an algorithm A using space $O(2^k + \log(n))$. In this case, our decision would, for instance, depend on whether the parameter k is large or small compared to $\log(\log(n))$: If we have $k \geq \log(\log(n))$ for a given instance x, then we also have $2^{2^k} \geq n$, thus the input is already a kernel. On the other hand, if we have $k < \log(\log(n))$, then we can use A to decide $x \in Q$, which effectively requires space $O(\log(n))$, and, depending on the result, we output a fixed positive or negative instance.

The backward direction essentially works the same way: We compute the parameter value, and, depending on its value compared to the input size, we either argue that the whole computation can be seen as an arbitrary complex precomputation on the parameter or as a computation within the resource bounds of the underlying class C. □

Using this lemma, we can show that the vertex cover problem is not only fixed-parameter tractable, but lies deep inside FPT:

▷ *Theorem 9 (Bannach et al. (2015)).* p-VERTEX-COVER $\in$ para-AC^0.

Proof. We, again, make use of Buss' kernelization idea, but now we cannot use the algorithm from above, because we are only allowed to use circuits of constant depth and an implementation of the kernelization described above requires depth polynomial in the parameter value, as it deals with high-degree vertices one after another. However, Buss' kernelization also works if we remove all high-degree vertices in parallel![2] Now we only have to argue that all of these steps can be done via AC^0 circuits. For this, note that checking whether a vertex has high degree, computing the reduced graph, and counting the remaining edges can be done using threshold gates, yielding that the kernelization can be computed using TC^0 circuits. However, looking more closely reveals that the thresholds computed by these gates are all bounded by the parameter and, thus, are independent of the input size. This allows us to apply an involved result from Newman et al. (1990) showing that thresholds of polylogarithmic size can be computed using AC^0 circuits: Before we start the kernelization process, we compute the parameter value. If $k \leq \log(n)$, i. e., the parameter is at most logarithmic in the input size, we apply Buss' kernelization where we substitute the threshold gates with small AC^0 circuits in the manner of

[2] In fact, Buss' kernelization mentioned in the original paper of Buss and Goldsmith (1993) works in parallel.

Newman et al. (1990). If $k > \log(n)$, the overall input size is bounded by the parameter, and, thus, we already have a kernel. Overall, this gives us a kernelization for p-VERTEX-COVER that is computable using AC^0 circuits, which immediately gives us p-VERTEX-COVER $\in$ para-AC^0. $\square$

The previous theorem, stating that p-VERTEX-COVER lies in the smallest reasonable para-class, underlines an empirical observation that has been made by many researchers: p-VERTEX-COVER belongs to the easiest problems in parameterized complexity theory. Additionally, this theorem – and thus parameterized space and circuit complexity – shows one more aspect of the vertex cover problem: It is perfectly parallelizable! The algorithm above is in fact a parallel algorithm solving p-VERTEX-COVER in parameterized parallel constant time and parameterized parallal polynomial work because the depth of the circuit is in $O(1)$ and the size of the circuit is in $O(f_k \cdot p(n))$. Parameterized space and circuit complexity theory thus provided us with insights into the complexity of the vertex cover problem that where not revealed by parameterized time complexity theory alone.

While p-VERTEX-COVER lies deep inside para-P, there are also examples of problems that are, either under reasonable assumptions or even provably, placed much higher inside para-P. Two of them are the directed and undirected versions of the feedback vertex set problem:

▷ *Problem 10 (Directed and Undirected Feedback Vertex Set).*

Instance: A graph G with $G = (V, E)$ and a natural number k.

Parameter: k.

Question: Is there a set C with $C \subseteq V$ and $|C| = k$ such that every cycle in G contains at least one vertex of C? In other words: Is the graph that we obtain if we remove C and its incident edges from G cycle-free?

Let us call the version of this problem restricted to undirected graphs p-FVS and the version for directed graphs p-DFVS.

▷ *Fact 11 (Bodlaender (1993), Chen et al. (2008)).* p-FVS and p-DFVS lie in para-P.

While both the directed and the undirected version of the feedback vertex set problem are fixed-parameter tractable, applying parameterized space and circuit classes show first differing lower bounds for these problems, thus giving a first hint on their different complexity:

▷ *Theorem 12.*

1. If p-FVS $\in$ para-NC^1, then $\mathrm{NC}^1 = \mathrm{L}$.
2. If p-DFVS $\in$ para-L, then $\mathrm{L} = \mathrm{NL}$.
3. p-FVS $\notin$ para-AC^0.
4. p-DFVS $\notin$ para-AC^0.

Proof. For the first part, assume that p-FVS $\in$ para-NC^1 holds. Then there is a uniform family of circuits of size $O\big(f_k \cdot p(n)\big)$ and depth $O\big(\log(f_k + n)\big)$ that decides p-FVS. Now consider the following trivially parameterized problem p_0-Cycle-Free:

▷ *Problem 13 (Trivially Parameterized Cycle-Free Graph).*

Instance: An undirected graph G with $G = (V, E)$.
Parameter: 0.
Question: Is G cycle-free?

Since an undirected graph is cycle-free if, and only if, it has a feedback vertex set of size 0, the circuit-family for p-FVS also decides p_0-Cycle-Free. However, due to the fixed parameter value of p_0-Cycle-Free, there is thus a circuit family with circuits of size $O\big(p(n)\big)$ for a polynomial p and depth $O\big(\log(n)\big)$, i. e., an NC^1 circuit family, deciding the unparameterized problem Cycle-Free. Cook and McKenzie (1987) showed that Cycle-Free is in fact complete for L. Thus, p-FVS $\in$ para-NC^1 implies $\mathrm{NC}^1 = \mathrm{L}$.

For the second part, we can proceed in a similar way using the fact that Cycle-Free for directed graphs, Directed-Cycle-Free, is complete for NL, see Jones (1975) together with Immerman (1988).

The last two items are now simple corollaries of the first two items. Let us prove the first of them, the other one can be shown in a similar way. Assume that p-FVS $\in$ para-AC^0. With the arguments of the first point this implies that $\mathrm{L} = \mathrm{AC}^0$, which is well known to be not the case, as was shown by Furst et al. (1984). □

Later in this thesis, we will see that we can adjust this proof to obtain even stronger lower bounds for much larger classes. However, even from the theorems above we can see that parameterized space classes reveal a fine structure of para-P that would be invisible without them: p-Vertex-Cover has a provably smaller complexity than p-FVS and p-DFVS, and even p-FVS and p-DFVS are presumably of different complexity. Interestingly, we can even observe effects in the world of parameterized time that are presumably related to our space- and circuit-based results. For example, p-VERTEX-COVER, p-FVS, and p-DFVS are all in para-P, but there are huge differences in the complexity of their fixed-parameter tractability. The following table shows the time requirements of the currently fastest algorithms for the mentioned problems:

p-VERTEX-COVER	$O(1.2738^k + k \cdot n)$	Chen et al. (2006)
p-FVS	$O(3.83^k k \cdot n^2)$	Cao et al. (2010)
p-DFVS	$O(4^k k^3 k! \cdot n^4)$	Chen et al. (2008)

We can see that the problem with the lowest computational complexity admits the fastest algorithm, the one with the highest computational complexity the slowest. However, if or how the space and circuit complexity of a parameterized problem in para-P is connected to its time complexity is unknown.

Up to now, we used para-classes to study the fine structure of para-P. We considered para-versions of well-known space and circuit classes, and showed that several problems lie in these subclasses of para-P and others, under reasonable assumptions, do not. However, there is another way of studying the fine structure of para-P: X-classes. While para-classes provide a relaxed notion of tractability by allowing nearly arbitrary resource consumption in terms of the parameter but staying efficient with respect to the overall input size, a second way to relax tractability is to connect the degree of efficiency with respect to the overall input size to the parameter. To formalize this, we use X-classes:

▷ *Definition 14 (X-Classes).* Let C be a classical complexity class. Then, XC is the class of parameterized problems (Q, κ) with $Q \subseteq \Sigma^*$ for an alphabet Σ such that for every language Q_k with $Q_k = \{x \mid x \in Q \land \kappa(x) = k\}$ we have $Q_k \in C$ via a single algorithm A.

The most famous X-class is presumably XP, the class of parameterized problems (Q, κ) such that every language Q_k we have $Q_k \in P$. If we turn to the O-notation, we get that XP is the class of parameterized problems that are decidable in time $O(n^{f_k})$. Inserting standard space and circuit classes, we get the following X-classes:

XL The class of languages that are decidable via a deterministic Turing machine that uses at most $O\big(f_k \cdot \log(n)\big)$ many read-write cells.

XNL The class of languages that are decidable via a nondeterministic Turing machine that uses at most $O\big(f_k \cdot \log(n)\big)$ many read-write cells.

XAC^i The class of languages that are decidable via a family of circuits over the standard base, with unbounded fan-in, size $O(n^{f_k})$, and depth $O\big(f_k \cdot \log^i(n)\big)$.

XTC^i The class of languages that are decidable via a family of circuits over the standard base together with threshold gates, with unbounded fan-in, size $O(n^{f_k})$, and depth $O\big(f_k \cdot \log^i(n)\big)$.

XNC^i The class of languages that are decidable via a family of circuits over the standard base, with bounded fan-in, size $O(n^{f_k})$, and depth $O\big(f_k \cdot \log^i(n)\big)$.

Like in the case of para-classes, X-classes inherit their inclusion structure from the underlying classical complexity classes. More interesting relations can be observed if we compare X-classes with para-classes. First, we get the following lemma if we consider para-classes and X-classes with the same underlying classical complexity class:

▷ *Lemma 15.* Let C be a classical complexity class. Then we have para-C $\subseteq$ XC. If C is one of the classes L, NL, P, NP, PSPACE, then the inclusion is strict.

The inclusion of the lemma above follows directly from the definition. To show that the inclusion is strict, one can make use of the well-known time and space hierarchy theorems, see for example Papadimitriou (1994). Since these proofs are straight-forward, we omit them here, but an example proof showing that para-P $\subsetneq$ XP can be found in the text book of Flum and Grohe (2006). Instead, let us have a look at two examples of problems in XAC^0. The first example is our well known victim p-Vertex-Cover. We have already seen that p-Vertex-Cover $\in$ para-AC^0, and, since para-AC^0 $\subseteq$ XAC^0, this immediately implies that p-Vertex-Cover $\in$ XAC^0. The other example is a problem that we briefly discussed before, namely the parameterized clique problem p-Clique:

▷ *Problem 16 (Parameterized Clique).*

Instance: An undirected graph G with $G = (V, E)$ and a natural number k.

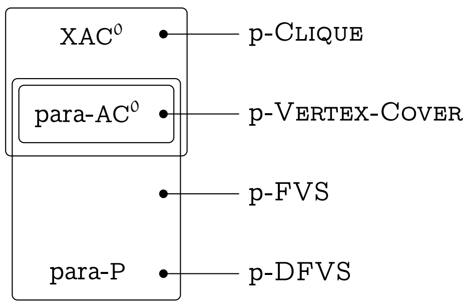
Parameter: k.

Question: Is there a set C with $C \subseteq V$ and $|C| = k$ such that between every two vertices from C there is an edge, i. e., for every u and v with $u \neq v$ and $u \in C$ and $v \in C$ we have $\{u, v\} \in E$?

▷ *Theorem 17.* p-Clique $\in$ XAC^0.

Proof. To find a vertex cover of size k, we construct a circuit family where each circuit consists of $\binom{|V|}{k}$ subcircuits, each checking for a selection of k vertices from the vertex set V of the input graph whether it is a clique, i. e., whether they are pairwise connected. Using gates of unbounded fan in, the circuits of the circuit family have constant depth and size $O\binom{|V|}{k}$, which can be bounded by $O(|V|^k)$. □

Comparing this result of "X-complexity" with the "para-complexity" of problems like p-VERTEX-COVER, p-FVS, or p-DFVS above, we can observe something that will later become a fundamental approach for studying the complexity of a problem: X-classes and para-classes of different underlying complexity classes lie often orthogonal to each other: As we have discussed briefly before, p-CLIQUE is presumably not fixed-parameter tractable, but lies in XAC^0. On the other hand, p-FVS and p-DFVS are fixed-parameter tractable, but do not lie in XAC^0, which can easily be seen from the same arguments used in the proof of Theorem 12. p-VERTEX-COVER, however, lies in both para-P and XAC^0. Illustrated as a Venn diagram, the situation is, under the assumption that p-CLIQUE is not fixed-parameter tractable, as follows:



To wrap up this section, Figure 2.1 illustrates the classes introduces so far together with the classes that will be introduced in the remaining part of this thesis.

2.3. Parameterized Reductions

To work with the classes we have seen so far, we need one more ingredient: *reductions*. Although this notion is very basic to complexity theory, it is worth reviewing them in the context of this thesis because some of the classes we study later are defined with respect to different reduction notions and we have to deal with the resulting effects. Let us therefore briefly review how reductions are used in classical computational complexity theory looking at the notion of *many-one reductions*: We say that a language A over alphabet Σ many-one-reduces to a language B over alphabet Γ if there is a computable

$$\begin{tikzcd}
 & & & & & & \mathrm{XNP} \arrow[dl] \arrow[d] \\
 & \text{para-PSPACE} \arrow[rrrr] \arrow[d] & & & & \mathrm{XNP/para\text{-}PSPACE} \arrow[dl] \arrow[d] & \mathrm{XP} \arrow[dl] \arrow[ddd] \\
 & \text{para-NP} \arrow[d] \arrow[drr, dash] & & = & \text{para}_{\mathrm{W}}\text{-NP} \arrow[d] & \mathrm{XP/para\text{-}PSPACE} \arrow[dl] & \\
 & \text{para}_{\beta}\text{-P} \arrow[ddd] & & = \mathrm{W[P]} = \arrow[d] \arrow[ddrr] & \text{para}_{\mathrm{W}}\text{-P} \arrow[ddd] & & \\
 & & & \mathrm{W[SAT]} \arrow[d] \arrow[ddddddr, bend left] & & & \mathrm{XNL} \arrow[dl] \arrow[dd] \\
 & & & \mathrm{W[t]} \arrow[dll] & & \text{para-NP/XNL} \arrow[dl] \arrow[ddd] & \\
 & \text{para-P} \arrow[d] \arrow[ddrrrr] & & & \text{para}_{\mathrm{W}}\text{-NL} \arrow[dlll] \arrow[dd] & & \mathrm{XL} \arrow[ddll] \arrow[ddl] \arrow[dd] \\
 & \text{para-NL} \arrow[d] & & & & & \\
 & \text{para}_{\mathrm{D}\beta}\text{-L} \arrow[dl] \arrow[dr] & & & \text{para}_{\mathrm{W}}\text{-L} \arrow[dll] \arrow[dd] & \text{para-P/XL} \arrow[ddllll] & \mathrm{XNC}^1 \arrow[ddll] \arrow[d] \\
 \text{para}_{\beta\forall}\text{-L} \arrow[dr] & & \text{para}_{\beta}\text{-L} \arrow[dl] & & & & \mathrm{XTC}^0 \arrow[ddll] \arrow[d] \\
 & \text{para-L} \arrow[d] & & & \text{para}_{\mathrm{W}}\text{-NC}^1 \arrow[dlll] \arrow[d] & & \mathrm{XAC}^0 \arrow[ddll] \\
 & \text{para-NC}^1 \arrow[d] & & & \text{para}_{\mathrm{W}}\text{-TC}^0 \arrow[dlll] \arrow[d] & & \\
 & \text{para-TC}^0 \arrow[d] & & & \text{para}_{\mathrm{W}}\text{-AC}^0 \arrow[dlll] & & \\
 & \text{para-AC}^0 & & & & &
\end{tikzcd}$$

Figure 2.1: Diagram of the most important complexity classes discussed in this thesis together with their inclusions where $A \rightarrow B$ denotes the inclusion $A \supseteq B$.

function f with $f\colon \Sigma^* \to \Gamma^*$ such that for every x with $x \in \Sigma^*$ we have that $x \in A \Leftrightarrow f(x) \in B$. If we can reduce A to B, we also write $A \leq B$. For an example, let us consider the vertex cover problem and the independent-set problem, where the independent-set problem is defined as follows:

Input: An undirected graph G with $G = (V, E)$ and a natural number k.

Question: Is there a set C with $C \subseteq V$ and $|C| = k$ such that for every edge $\{u, v\}$ of the graph we have $|C \cap \{u, v\}| < 2$, i. e., between the vertices of C there are no edges?

In the form of languages, we can define these problems by

$$\begin{aligned} \textsc{Vertex-Cover} &= \{\, \mathrm{code}(G, k) \mid G \text{ has a vertex cover of size } k\}, \\ \textsc{Independent-Set} &= \{\, \mathrm{code}(G, k) \mid G \text{ has an independent-set of size } k\}, \end{aligned}$$

where code denotes an appropriate encoding function mapping into the underlying alphabet. Then, we can reduce INDEPENDENT-SET to VERTEX-COVER by using a reduction function that maps $\mathrm{code}(G, k)$ to $\mathrm{code}(G, |V| - k)$ where $|V|$ denotes the number of vertices of G (and, of course, respects the underlying alphabets). It is easy to see that this reduction is correct, i. e., we have $\mathrm{code}(G, k) \in \textsc{Vertex-Cover} \Leftrightarrow \mathrm{code}(G, |V| - k) \in \textsc{Independent-Set}$ because a set of vertices is vertex cover if, and only if, every edge of the graph is incident to one of the vertex cover's vertices, and, hence, there are no edges between the vertices that are not in the vertex cover. Thus, the remaining vertices form an independent set.

Having a reduction that reduces A to B, we can conclude that if we are able to solve B, then we can also solve A by first computing the reduction and then deciding B on the output of the reduction. Moreover, if we can compute the reduction and decide B efficiently, then we can also decide A efficiently. On the other hand, if we already know that we are not able to decide A (efficiently), but have an (efficiently) computable reduction from A to B, then it is impossible that we can decide B (efficiently). Let us return to the example above. The reduction from INDEPENDENT-SET to VERTEX-COVER is computable very effciently because we only have to count the number of vertices of the graph and subtract k. Hence, if we find a polynomial-time computable algorithm for VERTEX-COVER, we immediately have a polynomial-time computable algorithm for INDEPENDENT-SET, and, on the other hand, if we can prove that there is no such algorithm for INDEPENDENT-SET, we immediately know that there is no such algorithm for VERTEX-COVER. Based on these observations, problems, classes, and reductions are studied with respect to the concepts of *closedness*, *hardness*, and *completeness*: We say that a class C of problems is *closed* under a fixed reduction notion if for every language A and B we have that $A \leq B$ and $B \in C$ implies that $A \in C$. For example, the class NP is well known to be closed under polynomial-time computable reductions. We say a problem B is *hard* for a class C of problems if for every problem A with $A \in C$ we have $A \leq B$, and, moreover, a problem B is *complete* for a class C if B is hard for C and we have $B \in C$. With these notions from classical computational complexity theory, let us now turn to the parameterized world.

As we have seen above, the independent set problem reduces to the vertex cover problem by simply changing the parameter to $|V| - k$. If we study the parameterized versions of these problems where we parameterize by the size of the requested independent set and the size of the vertex cover, there is a problem: While we still have the property that the complement of an independent set is a vertex cover, the reduction has a drastic influence on the parameter by making the new parameter depend on the input size. Hence, if we allow this reduction, we cheat! By reducing p-INDEPENDENT-SET to p-VERTEX-COVER in the mentioned way, we secretly place the size of the graph in the parameter, and the conclusion that since p-VERTEX-COVER is fixed-parameter tractable also p-INDEPENDENT-SET is fixed-parameter tractable is wrong because we then use the size of the graph within the quickly growing function that we originally only allowed to depend on the parameter. This issue is resolved by the following reduction notion:

▷ *Definition 18 (Parameterized Reductions).* Let (Q_1, κ_1) with $Q_1 \subseteq \Sigma_1$ and (Q_2, κ_2) with $Q_2 \subseteq \Sigma_2$ two parameterized problems. We say that (Q_1, κ_1) *many-one-reduces to* (Q_2, κ_2) if there is a computable function r with $r\colon \Sigma_1 \to \Sigma_2$ and a function g with $g\colon \mathbb{N} \to \mathbb{N}$ such that

1. for every x with $x \in \Sigma_1$ we have $x \in Q_1 \Leftrightarrow r(x) \in Q_2$,
2. $\kappa_2(r(x)) \leq g(\kappa_1(x))$, i. e., the new parameter value is bound only in terms of the old parameter value.

To ensure the closedness of our classes, we also have to restrict the computational power of the reductions we use. In this thesis we will consider the following three reductions:

▷ *Definition 19 (para-AC^0-, para-L-, and para-P-Reductions).*

para-AC^0-Reductions The reduction function is computable by a logarithmic-time uniform para-AC^0-circuit family.

para-L-Reductions The reduction function is computable by a para-L-restricted Turing machine.

para-P-Reductions The reduction function is computable by a para-P-restricted Turing machine.

While every class discussed in this thesis is closed with respect to para-AC^0-reductions, it is only known that para-L and its superclasses are closed under para-L-reductions and para-P and its superclasses are closed under para-P-reductions.

However, let us return to the reduction of parameterized independent set to parameterized vertex cover. From the definition above, we can immediately see that the reduction is *not* a parameterized reduction, since the reduction violates the rule that the new parameter has to be bound in terms of the old parameter alone. In fact, it is unknown whether a para-P-reduction from p-INDEPENDENT-SET to p-VERTEX-COVER exists, and, more interestingly, the existence or non-existence of such a reduction is connected to very important open questions of computational complexity, as we will see in the next section.

2.4. Review of the Weft-Hierarchy

Up to now we used para-P as a relaxed notion of tractability and considered subclasses like para-L and para-AC^0 with the aim of giving more structure to this tractability notion. Much effort has been spend by many researchers to show that problems are tractable within this notion, but, however, there a problems that refuse to admit parameterized tractability. For these problems we require a notion of *parameterized intractability.* Over time several such notions have been studied, the so-called *Weft-Hierarchy* introduced by Downey and Fellows (1995a) being undoubtedly the most successful among them. Since its introduction, many equivalent definitions of the Weft-Hierarchy have been developed, in this thesis I will stick to the one of Flum and Grohe (2006):

▷ *Definition 20 (Weft-Hierarchy).* For every t with $t \geq 1$ the t-th level of the Weft-Hierarchy $W[t]$ is defined as

$$W[t] = \bigcup_{\varphi \in \Pi_t} [\text{p-WD}_\varphi]^{\text{para-P}}$$

i.e. the closure under para-P-reductions of the family of parameterized problems p-WD_φ that are defined by

▷ *Problem 21 (p-*WD_φ*).*

Instance: A logical structure S with universe U and a natural number k.

Parameter: k.

Question: Is there a relation A with $A \subseteq U^s$ and $|A| = k$ such that $S \models \varphi(A)$?

Here, Π_t denotes the set of first-order formulas with a single free second-order variable in prenex normal form where, starting with universal quantifiers as the outermost quantifiers, there are at most $t-1$ alternations of universal and existential quantifiers. Moreover, $\varphi(A)$ denotes the formula φ where we replace every occurrence of the free second-order variable with the relation A.

One of the most famous problems of the Weft-Hierarchy is p-CLIQUE. Given an instance of p-CLIQUE, we can easily reduce this instance to p-$\mathrm{WD}_{\varphi_{\mathrm{Clique}}}$ with

$$\varphi_{\mathrm{Clique}}(X) = \forall x \forall y\big((Xx \wedge Xy \wedge x \neq y) \to Exy\big)$$

using a para-P-computable reduction (in fact, our reduction does essentially nothing because the given input graph is already the desired logical structure and the parameter value does not change at all). Since $\varphi_{\mathrm{Clique}}$ has no quantifier alternation, we can conclude that p-CLIQUE $\in \mathrm{W}[1]$.

Another famous example for a problem that is placed on a higher level of the Weft-Hierarchy is p-DOMINATING-SET, the parameterized dominating-set problem.

▷ *Problem 22 (Parameterized Dominating Set).*

Instance: An undirected graph G with $G = (V, E)$ together with a natural number k.

Parameter: k.

Question: Is there a set C with $C \subseteq V$ and $|C| = k$ such that for every vertex v of the graph we have that either $v \in C$ or there is a vertex u with $u \in C$ that is connected with v via a direct edge.

Using the formula $\varphi_{\mathrm{Dominating\ Set}}(X)$ with

$$\varphi_{\mathrm{Dominating\ Set}}(X) = \forall x \exists y\big(Xx \vee (Xy \wedge Exy)\big)$$

we can conclude that p-DOMINATING-SET $\in \mathrm{W}[2]$.

From the definition of the Weft-Hierarchy we get the inclusion chain

$$\text{para-P} \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \cdots \subseteq \mathrm{W}[t] \subseteq \mathrm{W}[t+1] \subseteq \cdots \subseteq \mathrm{XP} \cap \text{para-NP}$$

where $W[t] \subseteq XP \cap \text{para-NP}$ follows from the facts that for a fixed Π_t formula an XP machine has enough time to iterate over all possible relations of size k searching for the one that satisfies φ, and a para-NP machine can use its nondeterminism to just guess it.

The Weft-Hierarchy has been widely regarded as the parameterized version of intractability because, as Chen and Meng (2008) state it in their survey paper, extensive computational experience and practice have given strong evidence that $\text{para-P} \not\subseteq W[1]$. Hence, showing that a problem is complete for one of the classes of the Weft-Hierarchy has been widely accepted as a very strong hint that the problem is not fixed-parameter tractable. Two early examples of problems that are complete for classes of the Weft-Hierarchy are the already mentioned problems p-Clique and p-Dominating-Set:

▷ *Fact 23 (Downey and Fellows (1995a,b)).* The problems p-Clique and p-Dominating-Set are complete for W[1] and W[2] under para-P-reductions, respectively.

Under the assumption that $\text{para-P} \neq W[1]$, we can immediately conclude from this result that there is no para-P-computable reduction from p-Clique to any problem in para-P, for example p-DFVS. This is all we can see using parameterized time classes. However, since we are interested in parameterized space and circuit classes, we usually consider much weaker reductions like para-L-reductions or para-AC^0-reductions. With these reductions in mind we can observe something interesting:

▷ *Theorem 24.* There exists no para-AC^0-reduction from p-DFVS to p-Clique.

Proof. The statement follows with arguments similar to the arguments of Theorem 12 together with the fact that $\text{p-Clique} \in XAC^0$: If there was a reduction, we could immediately conclude that the reachability problem in undirected graphs is solvable in AC^0, which is absurd since we know that $AC^0 \neq L$ from Furst et al. (1984). □

This is something, one would not expect at first sight: Immerman (1987, 1999) noticed that in computational complexity theory 'natural' problems that are complete via polynomial-time reductions for some complexity class tend to remain complete via first-order reductions. However, from the theorem above we see that this does not carry over into the parameterized setting of the Weft-Hierarchy: p-Clique is not complete for W[1] under para-AC^0-reductions! One can argue that, since the Weft-Hierarchy is defined via para-P-reductions and $AC^0 \neq P$, it is somehow natural that there are problems that are not complete for $W[1]$ under para-AC^0-reductions but under para-P-reductions – and this is right. However, using para-P-reductions to define the Weft-Hierarchy hides an important fact that can only be revealed using parameterized space complexity: p-DFVS is not a "Weft problem". Imagine that the Weft-Hierarchy was defined in terms of para-AC^0-reductions, para-NC^1-reductions, and para-L-reductions instead of para-P-reductions, and the t-th level of these hierarchies was denoted by $W[t]^{\text{para-}AC^0}$, $W[t]^{\text{para-}NC^1}$, and $W[t]^{\text{para-}L}$, respectively, i. e., formally we have

$$\begin{aligned} W[t]^{\text{para-}AC^0} &= [\text{p-WD}_\varphi]^{\text{para-}AC^0}, \\ W[t]^{\text{para-}NC^1} &= [\text{p-WD}_\varphi]^{\text{para-}NC^1}, \\ W[t]^{\text{para-}L} &= [\text{p-WD}_\varphi]^{\text{para-}L} \qquad \text{with } \varphi \in \Pi_t. \end{aligned}$$

Then, p-DFVS does not even lie in the Weft-Hierarchy if we consider para-AC^0-reductions as the underlying reduction notion, and if we consider para-L-reductions as the underlying reduction notion, then the question whether p-DFVS lies in the Weft-Hierarchy is directly related to one of the most important open questions from classical computational complexity theory:

▷ *Theorem 25.*

1. p-DFVS $\notin W[t]^{\text{para-}AC^0}$ for any t.
2. If p-DFVS $\in W[t]^{\text{para-}L}$ for some t, then $L = NL$.

Proof. Let us start with the first statement. First, note that $W[t]^{\text{para-}AC^0} \subseteq XAC^0$ for every t because XAC^0 is closed under para-AC^0 reductions and for every fixed first-order formula φ there is a uniform AC^0 circuit family that is equivalent to φ, see Vollmer (1999). Now, to decide for a given structure whether there is a satisfying assignment for φ of size k we use a circuit that, in parallel, tests for every of the $O(|U|^k)$ subsets of the universe U of size k whether it satisfies φ and accepts if there is one such subset.

For the sake of contradiction let us assume that p-DFVS $\in W[t]^{\text{para-}AC^0}$ for some t. Then we have that p-DFVS $\in XAC^0$ and can conclude with arguments similar to the ones in the proof of Theorem 12 that the reachability problem for directed graphs can be decided using AC^0 circuits, which is absurd because $AC^0 \neq NL$.

Let us now turn to the second statement. The proof is mostly identical to the proof of the first statement, but now we cannot conclude that if p-DFVS $\in W[t]^{\text{para-}L}$, then we also have p-DFVS $\in XAC^0$, because XAC^0 is not closed under para-L-reductions. However, we can conclude that p-DFVS $\in XL$ because XL is closed under para-L-reductions, and with the same arguments as in the proof of Theorem 12 we then have that the reachability problem for directed graphs can be decided in logarithmic space which implies $\mathrm{L} = \mathrm{NL}$. □

Figure 2.2 gives an overview of the classes and problems we have discussed in this section so far together with some classes we will discuss in the next chapter.

So, what makes a problem a "Weft problem"? Since its introduction, many attempts to obtain a deeper understanding of the Weft-Hierarchy have been made. These studies are based on weighted satisfiability problems for Boolean circuits, see Downey and Fellows (1995a), weighted definability problems for first-order formulas, see Flum and Grohe (2006), and alternating Turing machines and bounded nondeterminism, see Buss and Islam (2006, 2007); Chen and Flum (2003); Chen et al. (2003, 2005). All of these studies reveal a common algorithmic approach to solve "Weft problems" that consists of three phases:

1. a preprocessing phase,
2. a guessing phase,
3. a verification phase.

Recall that we defined the t-th level of the Weft-Hierarchy, namely $\mathrm{W}[t]$, as the para-P-computable reduction closure of the problems p-WD_φ for every fixed formula φ with $\varphi \in \Pi_t$. Here, the reduction closure corresponds to the preprocessing phase, finding a monadic relation of parametric size that satisfies φ to the guessing phase, and the evaluation of φ on the guessed relation to the verification phase. For an example, let us consider p-Clique. From the theorems and examples above, it is easy to see that p-Clique $\in \mathrm{W}[1]^{\text{para-AC}^0}$ because the input for p-Clique is already a logical structure that we can apply our formula φ_{Clique} on, thus we only have to guess a satisfying assignment. Hence, we have a para-AC^0-preprocessing, a guessing phase that guesses k vertices of the graph which requires $O\big(k \cdot \log(n)\big)$ nondeterministic bits, and a Π_1-expressible verification phase.

If we turn to p-DFVS, a natural decision procedure would be to guess the vertices of the feedback vertex set and then verify that the input graph is cycle-free if we remove the feedback vertex set. However, testing whether a directed graph is cycle-free is an NL-complete problem and not expressible in first-order logic, see Papadimitriou (1994). Hence, there is no hope to decide p-DFVS with this approach. Up to now, the only known way to decide p-DFVS in the "Weft way" is by using a para-P-preprocessing phase, but since we have p-DFVS $\in$ para-P, the guessing phase and the verification phase are not required afterwards. Moreover, from the theorems above, we also see that under

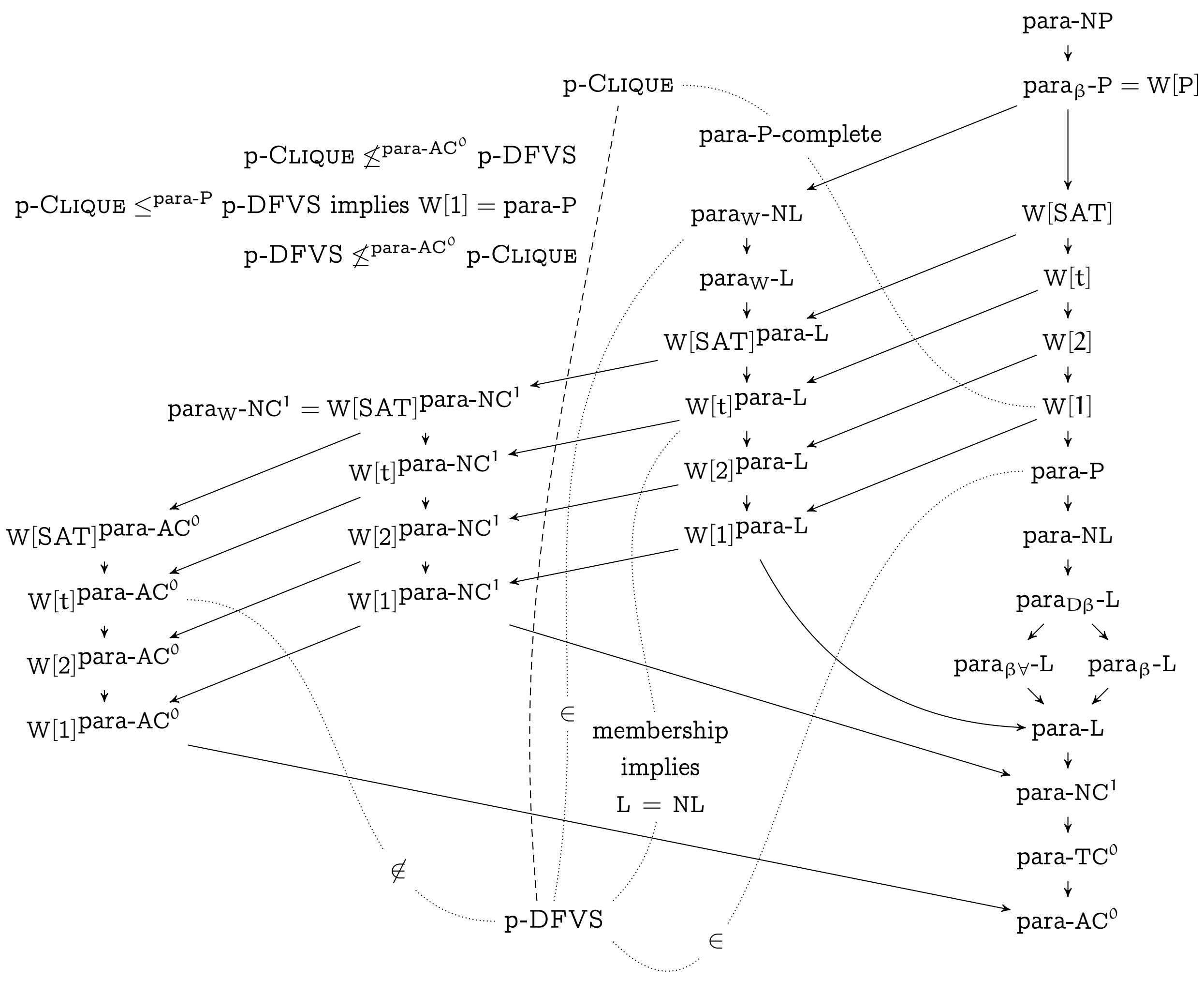

para-NP
$\text{para}_{\beta}\text{-P} = W[P]$
W[SAT]
W[t]
W[2]
W[1]
para-P
para-NL
$\text{para}_{D\beta}\text{-L}$
$\text{para}_{\beta\forall}\text{-L}$
$\text{para}_{\beta}\text{-L}$
para-L
para-NC^1
para-TC^0
para-AC^0
p-Clique
para-P-complete
$\text{para}_W\text{-NL}$
$\text{para}_W\text{-L}$
$W[SAT]^{\text{para-L}}$
$W[t]^{\text{para-L}}$
$W[2]^{\text{para-L}}$
$W[1]^{\text{para-L}}$
$\text{p-Clique} \not\leq^{\text{para-AC}^0} \text{p-DFVS}$
$\text{p-Clique} \leq^{\text{para-P}} \text{p-DFVS implies } W[1] = \text{para-P}$
$\text{p-DFVS} \not\leq^{\text{para-AC}^0} \text{p-Clique}$
$\text{para}_W\text{-NC}^1 = W[SAT]^{\text{para-NC}^1}$
$W[t]^{\text{para-NC}^1}$
$W[2]^{\text{para-NC}^1}$
$W[1]^{\text{para-NC}^1}$
$W[SAT]^{\text{para-AC}^0}$
$W[t]^{\text{para-AC}^0}$
$W[2]^{\text{para-AC}^0}$
$W[1]^{\text{para-AC}^0}$
$\in$
membership implies L = NL
$\notin$
p-DFVS
$\in$

Figure 2.2: Diagram of Weft-classes with respect to different reductions together with their surrounding classes and inclusions where $A \rightarrow B$ denotes the inclusion $A \supseteq B$. Moreover, the relations between p-DFVS and p-Clique as well as their relations to the drawn classes that were discussed in the previous section are illustrated.

the assumption that $\mathrm{L} \neq \mathrm{NL}$ a para-L-preprocessing is not sufficient to decide p-DFVS. In Chapter 3 we will study the requirements to decide problems like p-DFVS in more detail.

What can we learn from this? Hardness of a problem for a level of the Weft-Hierarchy is a strong evidence, that the problem is not fixed-parameter tractable. However, the Weft concept lies not at the core of parameterized intractability, but is an additional ingredient, that makes a problem intractable: Up to our knowledge, the Weft concept lies orthogonal to the concept of para-classes. Some problems are deeply connected to this concept, for instance p-CLIQUE, while other problems like p-DFVS are not. Hence, in order to study the computational complexity of parameterized problems, we should stop to see para-classes and Weft-classes as a linear hierarchy, and start to see the Weft concept as one dimension of a grid together with the concept of para-classes on the second dimension. While this makes Weft-classes appear somehow unattractive, they still serve as a good basis for further studies of parameterized space and circuit complexity as we will see in the next chapter.

3. Bounded Nondeterminism

One of the most intensively studied and still most poorly understood and thus most challenging concepts in computational complexity is the concept of *nondeterminism.* Since its introduction by Rabin and Scott (1959), literally thousands of research papers and theses that investigate nondeterminism have been published, and this thesis is one of them.

Undoubtedly, the most famous application of nondeterminism is the study of NP-hard and NP-complete problems. These problems are of special interest because, firstly, many of them are very natural, and, secondly, showing that a problem is NP-hard is widely regarded as a strong indication that the problem is not solvable in polynomial time. Since Cook (1971) showed that the satisfiability problem for propositional formulas is complete for NP, hundreds of NP-complete problems have been discovered. An early but already impressive list of more than 300 such problems is contained in the famous "black book" of Garey and Johnson (1979). Since then, the number of problems known to be NP-complete has grown enormously. Besides the study of time in terms of polynomial time and nondeterministic polynomial time, nondeterminism also plays an important role in connection with the concept of space, most importantly in form of deterministic and nondeterministic logarithmic space. Again, many natural problems are complete for NL, see Jones et al. (1976).

An immediate question rising from the importance of nondeterminism when studying the classical computational complexity of problems is: What is the role of nondeterminism in the world of parameterized complexity? At first, this question may seem easy to answer because from the generic definition of para-classes and X-classes we can easily define parameterized analogues of classes like NP and NL, for example para-NP and XNL, thus carrying over the knowledge about nondeterminism obtained from classical computational complexity to parameterized complexity. Moreover, Flum and Grohe (2006) already noticed, that "all para-NP-complete problems are, in some sense, uninteresting from the parameterized point of view because their hardness is already witnessed by finitely many parameter values." More formally, they show the following fact:

▷ *Fact 26.* Let (Q, κ) be a nontrivial parameterized problem in para-NP. Then the following two statements are equivalent:

1. (Q, κ) is para-NP-complete under para-P-reductions.
2. There is a finite set I of natural numbers such that $\{x \mid x \in Q \wedge \kappa(x) \in I\}$ is NP-complete under polynomial-time computable reductions.

It is not hard to see from the proof in Flum and Grohe (2006) that this also holds for other classes, for example if we focus on para-NL and para-L-reductions instead of para-NP and para-P reductions. (We will prove this result later for para-NL as a useful lemma when proving Theorem 51.) Hence, only X-classes may seem to be of interest, but many problems do not appear to be "X-problems". Is thus nondeterminism in general uninteresting in the parameterized setting?

In our discussion of the Weft-Hierarchy in the last chapter, we have already seen an appearance of nondeterminism: In the guessing phase, Weft algorithms make use of $O\big(k \cdot \log(n)\big)$ nondeterministic bits where k is the parameter value and x is the input. Hence, the amount of nondeterminism is not only bound by the input length but also by the parameter. In the following, we will call this *bounded nondeterminism*. It turns out that this concept gives us a large variety of useful and interesting classes that capture the complexity of many natural parameterized problems.

In this chapter, we will first study classes of bounded nondeterminism and their structural properties in Section 3.1. Moreover, we will consider a generic way of translating completeness results from classical computational complexity to the world of parameterized complexity that we will call *union operation*. We will see that many interesting and natural parameterized problems are in fact *union problems* and, therefore, deeply connected to classical computational complexity theory. In Sections 3.2 and 3.3, we will study already known and new natural problems from the perspective of bounded nondeterminism and union problems that are inside and outside of para-P, respectively.

3.1. Classes and Structural Properties

Nondeterminism appears in many different shapes. Since in this thesis we are mainly considering Turing machines and Boolean circuits, our working definition of nondeterminism will be in terms of these computational models. Let us turn to Turing machines first.

We consider nondeterministic Turing machines as deterministic Turing machines that are augmented with additional read-only tapes that we will call *choice tapes.* We then say that a nondeterministic Turing machine accepts an input if there is at least one bitstring for each choice tape that we can place on the choice tape at the beginning of the computation such that the (apart from the bitstrings deterministic) machine accepts. Using this model, we can define the class NP in a straight-forward fashion as the class of problems that are decidable by deterministic Turing machines in polynomial time that have access to a choice tape. However, this does not work for NL. If we consider Turing machines having a logarithmic space bound that are augmented with choice tapes (note that the length of the bitstring on the choice tape is unbound), then these machines can decide far more than only problems in NL, they can even decide every problem in PSPACE. This is because logarithmic space suffices to check that the choice tape contains the encoding of a valid computation of a PSPACE Turing machine, consisting of a sequence of configurations of polynomial size each. Hence, to obtain a machine model that characterizes NL, we have to impose further restrictions. The central idea for this is to limit the access to the choice tape: In the following, we distinguish between *one-way access* and *two-way access* to the choice tape, i. e., between the possibilities to move the head on the choice tape in one fixed direction only or in both directions. In other words, two-way access can be seen as the ability to read nondeterministic bits repeatedly while one-way access can be seen as read-once access to these bits. Now we can redefine NL: It is the class of problems that are decidable by Turing machines with logarithmic space bounds and one-way access to their choice tapes.

Based on these observations, let us formally define nondeterminism in the context of parameterized complexity.

▷ *Definition 27 (Sequential Classes of Bounded Nondeterminism).* Let C be a classical complexity class that is defined in terms of a deterministic Turing machine model. We then define para$\exists^{\leftrightarrow}$-C as the class of parameterized problems (Q, κ) over an alphabet Σ such that there is a para-C-machine M with a single two-way accessible choice tape and for every input x with $x \in \Sigma^*$ we have $x \in Q$ if, and only if, there is a string b with $b \in \{0, 1\}^*$ such that M accepts x with b on its choice tape. We define para$\exists^{\rightarrow}$-C similarly, but now we only allow one-way access to the choice tape. Moreover, for the classes para$\exists^{\leftrightarrow}_{f\log}$-C and para$\exists^{\rightarrow}_{f\log}$-C the length of b may be at most $O\big(f_k \cdot \log(n)\big)$ for a computable function f. Let us call the classes where we impose a bound on the length of the nondeterministic bitstrings *classes of bounded nondeterminism.*

Analogously, let us define the classes where, instead of an existential quantifier, we have a universal quantifier in the natural way: The input is accepted, if the machine accepts it for every possible content of the choice tape.

Finally, we define classes with sequences of quantifiers: For every quantifier the machine is equipped with a choice tape, and the conditions of the quantifiers must be met in order the quantifiers appear. For example $\mathrm{para}\forall^{\leftrightarrow}\exists^{\rightarrow}_{\mathrm{f}\log}$-L is the class of parameterized problems (Q, κ) decidable via a para-L-machine M with a two-way and a one-way choice tape such that for an input x we have $x \in Q$ if, and only if, for every binary string on the two-way choice tape, there is a binary string of length $O\big(f_k \cdot \log(n)\big)$ on the one-way choice tape such that M accepts.

Using this definition, we can uniformly define the classes we previously studied: $\mathrm{para}\exists^{\leftrightarrow}$-P is the same as para-NP and $\mathrm{para}\exists^{\rightarrow}$-L is the same as para-NL. Moreover, we can express parameterized complexity classes that we did not mention until now, for example the important class W[P] defined by Downey and Fellows (1995a) is the same as $\mathrm{para}\exists^{\leftrightarrow}_{\mathrm{f}\log}$-P, which can be seen from the many definitions of W[P], for example in Flum and Grohe (2006). Now we can express the classes we will focus on in the rest of this chapter:

$$\mathrm{para}\exists^{\rightarrow}_{\mathrm{f}\log}\text{-L}, \qquad \mathrm{para}\exists^{\leftrightarrow}_{\mathrm{f}\log}\text{-L}, \qquad \mathrm{para}\exists^{\leftrightarrow}_{\mathrm{f}\log}\exists^{\rightarrow}\text{-L}.$$

However, despite the fact that the notation introduced above expresses the type of nondeterminism we make use of in a formal and clear way, it is very cumbersome. Hence, we make use of the following abbreviated versions:

$$\mathrm{para}_{\beta}\text{-L}, \qquad \mathrm{para}_{W}\text{-L}, \qquad \mathrm{para}_{W}\text{-NL}.$$

Following the notation from Buss and Goldsmith (1993), we use β to denote read-once bounded existential nondeterminism. The use of W is inspired by the Weft-Hierarchy: As we have seen in the previous chapter, Weft algorithms have a guessing phase and, during the verification, repeated access to the guessed elements is possible.

The notation using the subscripted W allows us to extend the concept of bounded nondeterminism to other computational models, for example circuits. For this, note that the subscripted W effectively augments the input of the Turing machine with $O\big(f_k \cdot \log(n)\big)$ nondeterministic bits. From this view, we can immediately define circuit classs of bounded nondeterminism in a reasonable way:

▷ *Definition 28 (Circuit Classes of Bounded Nondeterminism).* Let C be a classical complexity class that is defined in terms of circuits. We then define para_W-C as the class of parameterized problems (Q, κ) that are decidable using para-C circuits that are given $O(f_k \cdot \log(n))$ nondeterministic bits along with their input.

For example, $\text{para}_W\text{-AC}^0$ is the class of parameterized problems decidable via circuits of size $O(f_k \cdot p(n))$ for a polynomial p and depth $O(1)$ with gates over the standard base and unbounded fan-in that have access to $O(f_k \cdot \log(n))$ nondeterministic bits. These nondeterministic bits prove to be quite powerful: It is not hard to see that p-CLIQUE $\in \text{para}_W\text{-AC}^0$ holds.

With these classes, we are only missing one final notion before we can study the complexity of actual problems in depth: the *union operation.*

The *union operation* is a generic method to show completeness for parameterized complexity classes that rely on the concept of bounded nondeterminism. This operation allows us to connect the world of classical computational complexity with the parameterized world by turning problems complete for classes like P, L, or NL into problems complete for para_W-P, para_W-L, or para_W-NL, respectively. It is based on the consideration that many problems studied in computational complexity contain the task of selecting elements from the input that form a solution. For example, the clique problem asks for a set of vertices of the input graph that are pairwise connected, the satisfiability problems asks for a set of literals that have to be set to true in order to satisfy the input formula, and the subset sum problem asks for a set of numbers from the input that add up to a specified target sum. The union operation enables us to formalize these problems in a uniform way and connect them to our classes and, more generally, to the W-operator introduced above.

▷ *Definition 29 (Union Operation).* Let Σ be an alphabet that does not contain the symbols ?, 0, and 1. We then call a word t with $t \in (\Sigma \cup \{?\})$ a *template* and any word that is obtained from a template t by replacing every occurrence of the symbol ? by either 0 or 1 an *instantiation* of t. Given instantiations $s_1, \dots, s_n$ of the same template t, we call the instantiation of t that has a 1 at those positions where at least one of the s_i has a 1 and otherwise 0 the *union* of $s_1, \dots, s_n$.

More intuitively, we can think of a template t as a checkered transparency with symbols written in some of its boxes. Then, every instantiation of t is such a transparency where some of the empty boxes of the original transparency are additionally marked. We obtain the union of the instantiations by placing the transparencies on top of each other such that the symbols of the template align. Based on this intuition, let us define three *union problems* that, as we will see in a moment, are deeply connected to the W-operator and connect problems of classical complexity theory to parameterized problems.

▷ *Problem 30 (p-*Family-Union-A *for a language* $A \subseteq \left(\Sigma \cup \{0,1\}\right)^*$*).*

Instance: A family of k sets $S_1, \ldots, S_k$ of instantiations of a common template t with $t \in \left(\Sigma \cup \{?\}\right)^*$

Parameter: k.

Question: Are there $s_1, s_2, \ldots, s_k$ with $s_i \in S_i$ such that their union lies in A?

▷ *Problem 31 (p-*Subset-Union-A *for a language* $A \subseteq \left(\Sigma \cup \{0,1\}\right)^*$*).*

Instance: A set S of instantiations of a template t with $t \in \left(\Sigma \cup \{?\}\right)^*$ and a natural number k.

Parameter: k.

Question: Is there a set $S' \subseteq S$ with $|S'| = k$ such that the union of the instantiations in S' lies in A?

▷ *Problem 32 (p-*Weighted-Union-A *for a language* $A \subseteq \left(\Sigma \cup \{0,1\}\right)^*$*).*

Instance: A template t with $t \in \left(\Sigma \cup \{?\}\right)^*$ and a natural number k.

Parameter: k.

Question: Is there an instantiation s of t with exactly k many occurences of the symbol 1 that lies in A?

While the origin of the names p-Family-Union-A and p-Subset-Union-A are easy to see, it might be unclear where the name p-Weighted-Union-A comes from. One reason for its name lies in the notion of the *Hamming Weight* where the weight of a string is the number of non-zero elements of the string (we only focus on the non-zero elements on positions where the template has a ?). The second reason lies in its connection to fundamental parameterized problems like the weighted satisfiability problem p-Weighted-SAT:

▷ *Problem 33 (Parameterized Weighted Satisfiability).*

Instance: A propositional formula ϕ and a natural number k.

Parameter: k.

Question: Is there a weight-k satisfying assignment of ϕ, i. e., an assignment that sets exactly k of the variables of ϕ to true?

In fact, p-Weighted-SAT is a weighted-union problem: Let A be the Boolean formula value problem BFVP discussed in Buss et al. (1992) and Buss (1987) where we are asked to decide whether a given Boolean formula together with a given truth assignment for its variables is satisfied. Moreover, let the symbols 0 and 1 of the alphabet Σ solely encode the assignment to the variables. For example, the formula

$$(x_0 \wedge x_1) \vee (\overline{x_1} \wedge x_2) \vee (\overline{x_0} \wedge (\overline{x_1} \wedge \overline{x_2}))$$

together with an assignment where x_0 and x_1 are set to be true and x_2 is set to be false could be encoded as

$$110\#\,\mathsf{code}\big((x_0 \wedge x_1) \vee (\overline{x_1} \wedge x_2) \vee (\overline{x_0} \wedge (\overline{x_1} \wedge \overline{x_2}))\big)$$

where the first three symbols encode the variable assignment and the string after the # encodes the formula using an alphabet that does not contain the symbols ?, 0, and 1. Now, if we turn to p-Weighted-Union-BFVP, inputs are encodings of formulas like the one above where the symbols 0 and 1 are replaced by ?, together with a natural number k. For the formula above and the parameter 2, the input would be

$$\Big(???\#\,\mathsf{code}\big((x_0 \wedge x_1) \vee (\overline{x_1} \wedge x_2) \vee (\overline{x_0} \wedge (\overline{x_1} \wedge \overline{x_2}))\big), 2\Big).$$

Now, the question is whether we can replace k of the symbols ? with 0 (and the remaining occurences of ? with 1) such that the resulting word is an element of BFVP. It is easy to see that this problem is equivalent to p-Weighted-SAT.

Besides p-Weighted-SAT, many other parameterized problems can be expressed using union operations, for example p-Weighted-Circuit-SAT, p-Clique, p-Vertex-Cover, or p-Subset-Sum. How can we benefit from this? First of all, for many union problems it is easy to see that they are members of classes of bounded nondeterminism. Recall that, when using bounded nondeterminism, we are provided with $O\big(f_k \cdot \log(n)\big)$ many nondeterministic bits that suffice to select $O\big(f_k\big)$ many positions within the input, for example k of the given instantiations of a subset-union problem. We then only have to compute the union of the instantions and check whether the result is a member of the underlying language. Secondly, besides being a member of a class of bounded nondeterminism, union problems are also often complete for such classes. Before we formalize this in Lemma 35, we require the following definition of *format-preserving projections* as the central ingredient for the lemma:

▷ *Definition 34 (Format-Preserving Projections).* Let A with $A \subseteq \Sigma^*$ and B with $B \subseteq \Gamma^*$ be two languages. A function $p\colon \Sigma^* \to \Gamma^*$ is a *format-preserving projection* if

1. p is a reduction from A to B, i. e., for every w with $w \in \Sigma^*$ we have $w \in A \Leftrightarrow p(w) \in B$,
2. p is a projection, i. e., every symbol of $p(w)$ depends on at most one symbol of w,
3. p is format-preserving, i. e., for every n, the set $\{ p(w) \mid w \in \Sigma^n\}$ is a set of instantiations of the same template t.

We say that p is an *AC^0-computable format-preserving projection* if p is computable via a uniform family of AC^0 circuits.

Intuitively, if there is a AC^0-computable format-preserving projection from A to B, then words of the same length are mapped to words of the same coarse structure because the underlying template stays the same and only details change. In other words, the occurences of ? within the template are filled differently with 0 or 1.

Why do we consider such a cumbersome notion of reduction? First of all, many classical problems that are complete for some reasonable complexity class like NP or L are also complete for these classes with respect to format-preserving projections as we will see later. Secondly, if a problem is complete with respect to these reductions for some complexity class C, the accompanying union problems can often shown to be complete for the corresponding class of bounded nondeterminism:

▷ *Lemma 35 (Union-Lemma).* Let C be a complexity class with $C \in \{L, NL, P, AC^i, TC^i, NC^{i+1}\}$ and $i \geq 0$. Moreover, let A be a language with $A \subseteq \left(\Sigma^* \cup \{0,1\}\right)^*$ where Σ does not contain the symbols ?, 0, 1. If A is complete for C with respect to AC^0-computable format-preserving projections, then p-FAMILY-UNION-A is complete for para_W-C under para-AC^0-reductions.

Proof. Let us first consider membership. On input of a template t together with the family $(S_1, \ldots, S_k)$, we interpret the nondeterministic bits given along with the input as the description of k elements s_i with $s_i \in S_i$. We compute the union u of these s_i and run the underlying machine or circuit to decide whether $u \in A$.

Our proof plan for the hardness of p-FAMILY-UNION-A is as follows: First, we show that any problem (Q, κ) in para_W-C can be seen as a family-union problem by showing that we can reduce (Q, κ) to p-FAMILY-UNION-L where L is a certain unparameterized problem depending on (Q, κ). In the second step, we reduce p-FAMILY-UNION-L to p-FAMILY-UNION-A.

Let (Q, κ) be a problem from para_W-C with $Q \subseteq \Delta^*$ for an alphabet Δ. How can we interpret such a problem as a family-union problem? We get the answer for this question by looking at the definition of $\text{para}_W\text{-C}$: Since we have $(Q,\kappa) \in \text{para}_W\text{-C}$, we know from the definition of $\text{para}_W\text{-C}$ that there is a language X with $X \subseteq \Gamma^*$ for an alphabet Γ, a computable function $\pi\colon \mathbb{N} \to \Pi^*$ for an alphabet Π, and a computable function $f\colon \mathbb{N} \to \mathbb{N}$, such that for every x with $x \in \Delta^*$ we have $x \in Q$ if, and only if, there is string b of length $O\big(f_k \cdot \log(n)\big)$ over the alphabet $\{0,1\}$ with $(x, \pi(k), b) \in X$. Our language L from the theorem will be X, i. e., we will reduce (Q,κ) to p-FAMILY-UNION-X. The main idea for this is to translate the search for an appropriate b into a search for elements of a family and to make sure that the union of the elements gives us b. More formally, for an input x, we define a family $(S^x_1, \dots, S^x_{f_k})$ by

$$S^x_i = \Big\{ (x, \pi(k), v) \mid v = \overbrace{0\dots0}^{(i-1)\cdot\lceil\log(n)\rceil} w \underbrace{0\dots0}_{(f_k-i)\cdot\lceil\log(n)\rceil} \land w \in \{0,1\}^{\lceil\log(n)\rceil} \Big\}.$$

Now, for an input x there is a b such that $(x, \pi(k), b) \in X$ if, and only if, there is a selection of s_i with $s_i \in S^x_i$ such that the union of these s_i lies in X. In other words, there is a b with $(x, \pi(k), b) \in X$ if, and only if, we have $(S^x_1, \dots, S^x_{f_k}) \in$ p-FAMILY-UNION-X. Moreover, it is not hard to see that the family $(S^x_1, \dots, S^x_{f_k})$ can easily be computed using a para-AC^0 circuit family.

Now that we have reduced (Q,κ) to p-FAMILY-UNION-X, we have to reduce this problem to p-FAMILY-UNION-A, i. e., we have to map the $(S^x_1, \dots, S^x_{f_k})$ to a new family $(T^x_1, \dots, T^x_{f'_k})$ such that there is a selection of elements s_i with $s_i \in S^x_i$ whose union lies in X if, and only if, there is is a selection t_j with $t_j \in T^x_j$ whose union lies in A. We define T^x_i by

$$T^x_i = \{ p_i(w) \mid w \in S^x_i \}$$

where p_i is based on p: For an input w with $w = (x, \pi(k), v)$ the projection p_i effectively computes $p(w)$, but it additionally obeys the following rules:

1. If the r-th symbol of $p(w)$ depends on a symbol from v that belongs to the i-th of the f_k blocks of length $\lceil\log(n)\rceil$, the r-th symbol of $p_i(w)$ is the same as the r-th symbol of $p(w)$.
2. If the r-th symbol of $p(w)$ depends on a symbol from v that does not belong to the i-th of the f_k blocks of length $\lceil\log(n)\rceil$, the r-th symbol of $p_i(w)$ is 0.
3. In all other cases, the r-th symbol of $p_i(w)$ is the r-th symbol of $p(w)$.

Since p is format-preserving, we have that the elements of the new family are instantiations of the same template.

Altogether, we obtain a reduction from (Q, κ) to p-FAMILY-UNION-A by first reducing a given instance of (Q, κ) to p-FAMILY-UNION-X and then reducing it to p-FAMILY-UNION-A. Moreover, this reduction is computable by a para-AC^0 circuit family because the projection p is AC^0-computable. □

The union operation provides us with a useful framework to show that a problem is complete for a class para-C by only showing that the underlying classical problem is complete for C under AC^0-computable format-preserving projections. Fortunately, many classical problems have been studied with respect to projections, and, indeed, many problems are complete under projections for natural complexity classes. Let us have a look at such problems in the next section.

3.2. Natural Problems for para$_W$-Classes

The union operation together with the notion of AC^0-computable format-preserving projections provides us with tools for proving the completeness of problems for classes of parameterized bounded nondeterminism by only looking at their complexity from the perspective of classical computational complexity theory. In this section, we will have a look at examples that illustrate the application of the union operation, and, moreover, provide natural complete problems for our classes. We will start with satisfiability problems, continue our study with graph problems, and end with the most involved example of this section, namely the *associative generabilty problem*. While this selection covers some of the most natural topics in computational complexity, we will also come across some of the most important complexity classes and their parameterized analogs.

Let us start over with two well-known satisfiability problems: the Boolean formula value problem BFVP and the circuit value problem CVP. In the previous section, we already had a glimpse at BFVP and realized that the well-known problem p-WEIGHTED-SAT is in fact p-WEIGHTED-UNION-BFVP. So, what can we learn about this problem if we take a look at BFVP from the perspective of AC^0-computable format-preserving projections? First of all, BFVP has been shown to be complete for NC^1 under AC^0-reductions by Buss (1987) and Buss et al. (1992), and, moreover, this completeness can be established by projections. To be more precise, Buss et al. (1992) showed that NC^1 can equivalently be characterized by alternating Turing machines with a logarithmic time bound and that these machines can be simulated using NC^1 circuits. The important detail here is that, for all the instances of a fixed length n, this simulation can be performed by the same propositional formula where only the values of the input gates change with the input of the machine. Hence, BFVP is complete for NC^1 under AC^0-computable format-preserving projections. With the union lemma above we can then immediately conclude that the family-union problem p-FAMILY-UNION-BFVP is complete for para_W-NC^1 with respect to para-AC^0-reductions. However, we can extend this result:

▷ *Theorem 36.* p-WEIGHTED-UNION-BFVP is complete for para_W-NC^1 with respect to para-AC^0-reductions.

Proof. Since BFVP lies in NC^1, membership in para_W-NC^1 can easily be achieved: On input of a template t encoding a propositional formula ϕ together with a parameter k, the circuit interprets its nondeterministic bits as the description of k variables of ϕ that have to be set to true in order to satisfy ϕ. Our circuit replaces every occurence of such a variable in ϕ with the value true and every other variable with false and evaluates ϕ afterwards. If ϕ evaluates to true, then the circuit accepts, otherwise it rejects.

For hardness, we show the reduction chain

$$\begin{aligned}\text{p-FAMILY-UNION-BFVP} &\leq \text{p-SUBSET-UNION-BFVP}\\ &\leq \text{p-WEIGHTED-UNION-BFVP}.\end{aligned}$$

For the first reduction, let $(S_1, \dots, S_k)$ be an input for p-FAMILY-UNION-BFVP and let ϕ be the encoded formula. Our aim is to compute a new set T such that there are k elements in T whose union encodes a satisfied propositional formula if, and only if, there are k elements s_i with $s_i \in S_i$ whose union encodes a satisfied formula. To achieve this, we cannot simply let $T = \cup_{i=1}^{k} S_i$, because then it would be possible to select elements from T that originate from the same S_i. To avoid this, we do a simple trick: We extend the formula ϕ to a new formula $\phi \wedge \psi$ where ψ ensures that we do not pick two elements from the same S_i. The formula ψ introduces a fresh variable v_i for every S_i and is defined by

$$\psi = \bigwedge_{i=1}^{k} v_i.$$

Moreover, we adjust the elements of the sets of the family in such a way that exactly those elements assign true to the variable v_i that originate from S_i. Setting the new parameter to k then enforces that a solution to the problem has to contain exactly one element originating from each S_i. For an example, let the original instance be (S_1, S_2) with

$$\begin{aligned} S_1 &= \{\phi 000, \phi 001\}, \\ S_2 &= \{\phi 001\}, \\ S_3 &= \{\phi 101, \phi 100, \phi 110\} \end{aligned}$$

where the strings after the ϕ encode the assignment, for instance, 001 denotes the assignment where the first and the second variable are set to false and the third variable is set to true. Since we have three sets, the new formula ψ is defined by

$$\psi = \phi \wedge v_1 \wedge v_2 \wedge v_3.$$

Furthermore, the set T is then defined by

$$\begin{aligned} T = \{&\psi 000100, \psi 001100, \\ &\psi 001010, \\ &\psi 101001, \psi 100001, \psi 110001\}, \end{aligned}$$

where the last three bits of the assignment strings encode the assignment of the variables v_1, v_2, and v_3.

Since the new variables do not occur in ϕ, they do not have any influence on the satisfiability of ϕ, and, hence, there is a selection of k elements of S whose union satisfies $\phi \wedge \psi$ if, and only if, there are k elements, one from each S_i, whose union satisfies ϕ. Moreover, this reduction is clearly a para-AC^0-reduction.

Let us now turn to the second part of the reduction chain, namely the reduction from p-SUBSET-UNION-BFVP to p-WEIGHTED-UNION-BFVP. Hence, let S with $S = \{s_1, \ldots, s_n\}$ together with the parameter k be our input instance. The main problem is that a selection of k elements of S can set an arbitrary number of variables of the encoded formula to true, but in the weighted-union problem we are only allowed to set a number of variables to true that may only depend on the parameter k. Hence, we somehow have to "compress" the variables that can be set by one of the s_i to a single variable. For this, we introduce a variable v_i for every s_i. Now, to make sure that setting this variable to true sets ϕ to true at all the "places" that s_i would have set to true, we adjust the formula ϕ to ψ by replacing every occurence of a variable x_i by the subformula

$$\bigvee_{j \in S[x_i]} v_j$$

where $S[x_i]$ denotes the set of the indices j of those instantiations s_j from S that set the variable x_i to true. The remaining formula then has the variables $v_1, \ldots, v_n$, and any truth assignment to these variables corresponds to exactly one subset of S, and vice-versa. Hence, there is a selection of k elements of S whose union is a satisfying assignment of ϕ if, and only if, there is a instantiation of $\psi?^n$ that sets k of the variables to true. Finally, we output the new template $\psi?^n$, where $?^n$ is reserved for the assignment. Again, let us have a look at an example: Let S be the input where

$$S = \{\phi000, \phi101, \phi010, \phi011\} \quad \text{with} \quad \phi = x_1 \wedge (x_2 \to x_1) \wedge \overline{x_3}.$$

The new formula ψ is then

$$\psi = v_2 \wedge \Big((v_3 \vee v_4) \to v_2\Big) \wedge \overline{(v_2 \vee v_4)},$$

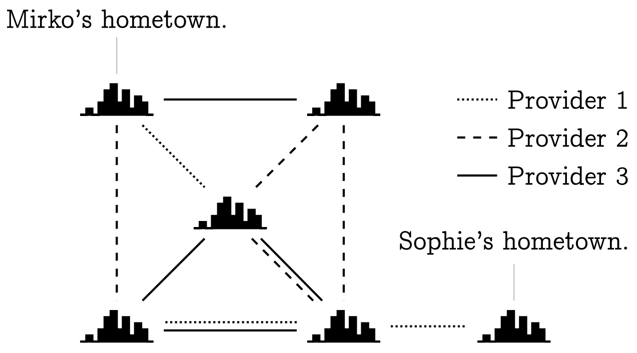
and, therefore, the template computed by the reduction is $\psi?^4$, the parameter stays k. □

We already know that p-WEIGHTED-SAT and p-WEIGHTED-UNION-BFVP are the same, and this fact yields an interesting observation: The famous problem p-WEIGHTED-SAT, which is complete for W[SAT], the class of problems reducible using para-P reductions to p-WEIGHTED-SAT, a superclass of W[t] for every t, is also complete for $\text{para}_W\text{-NC}^1$! It is tempting to conclude that $\text{W[SAT]} = \text{para}_W\text{-NC}^1$ and thus $\text{W[t]} \subseteq \text{para}_W\text{-NC}^1$, but this is not possible at the moment: Completeness of p-WEIGHTED-SAT for W[SAT] has been shown using para-P-reductions, but it is unknown whether $\text{para}_W\text{-NC}^1$ is closed under these reductions. However, we can conclude that W[SAT] is the para-P-reduction closure of $\text{para}_W\text{-NC}^1$, and this yields the following corollaries:

▷ *Corollary 37.* $\text{NC}^1 = \text{P}$ implies $\text{W[SAT]} = \text{para-P}$.

▷ *Corollary 38.* $\text{para}_W\text{-NC}^1 \subseteq \text{para-P}$ if, and only if, $\text{W[SAT]} = \text{para-P}$.

Let us turn to the second kind of problems studied within this section, namely graph problems. The first natural question to ask is: what is a union graph problem? Towards an answer of this question, let us consider an example: Uncle Mirko wants to visit aunt Sophie in her hometown, and he wants to visit her by train. Fortunately, there is a large railway network connecting the cities nearby, but, unfortunately, due to privatization of the network, the lines connecting the cities are operated by different providers. However, uncle Mirko can only afford tickets of two different providers (every provider only sells tickets that are valid for all the day within their whole network). The question is, can he afford to visit aunt Sophie? The situation is illustrated in the graphic below, and it is easy to see that Mirko has to spend money for at least two tickets.



We can formalize and generalize this as a reachability problem in edge-colored graphs where the colors of the edges represent the different providers, denoted by p-COLORED-REACH:

▷ *Problem 39 (Parameterized Colored Reachability).*

Instance: A directed graph G with vertex set V and edge relation E, in which every edge is colored with at least one color. Moreover, two vertices s and t of G and a natural number k are given.

Parameter: k.

Question: Can we choose k colors such that there is a path from s to t in G where every edge along the path is colored with at at least one color from the selected colors?

The problem p-COLORED-UNDIRECTED-REACH is defined in the same way for undirected graphs.

How is this problem related to union problems? If we focus on each color on its own and consider only edges that are colored with the currently considered color, we get a new graph with the same vertex set as before, but the set of edges can be a subset of the original set of edges. In other words, we can decompose the graph into several edge relations over the same vertex set, one edge relation for each color. The task of the colored reachability problem is then to find k of these edge relations such that their union contains a path from s to t. What remains is to encode these graphs and edge relations as templates and instances, but this is easily done: We simply use their adjacency matrices. These are essentially instantiations of the template $?^{n^2}$ where n is the number of vertices. From this point of view, it is easy to formulate the colored reachability problem in terms of templates and instances: It is p-SUBSET-UNION-REACH where REACH is defined by

$$\textsc{Reach} = \{\, \mathrm{code}(G) \mid G \text{ with } G = (\{1, \ldots, n\}, E) \text{ contains a path from } 1 \text{ to } n\}$$

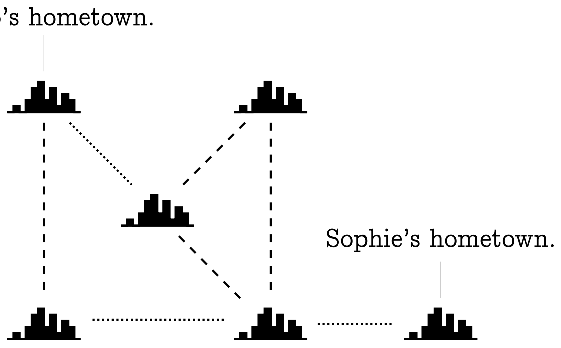
where code maps a directed graph to its adacency matrix written row-wise in one line, and, if we consider undirected graphs, the colored reachability problem becomes p-SUBSET-UNION-UNDIRECTED-REACH. After numbering the vertices of the map to aunt Sophie from top to bottom and left to right, the problem can be seen as an instance $(S, 2)$ of p-SUBSET-UNION-UNDIRECTED-REACH with

$$\begin{aligned} S = \{\, & 001000\,000000\,100000\,000010\,000101\,000010, \\ & 000100\,001010\,010010\,100000\,011000\,000000, \\ & 010000\,100000\,000110\,001010\,001100\,000000\,\}. \end{aligned}$$

We can obtain a possible solution by picking the first two edge relations. Their union yields

$$001100\,001010\,110010\,100010\,011101\,000010,$$

which is, displayed as the city map for visiting Sophie,



What can we say about p-SUBSET-UNION-UNDIRECTED-REACH's complexity? First of all, if we stick to parameterized time classes, it lies in para_{W}-P, a.k.a. W[P]. This is because a Turing machine with $O\big(k \cdot \log(n)\big)$ nondeterministic bits can interprete these bits as the description of the k colors that have to be chosen in order to get from 1 to n, and testing reachability in the remaining graph can easily be done in polynomial time. Can we improve this result? Showing that the problem lies in some level $W[t]$ of the Weft-Hierarchy or even belongs to para-P seems to be a really tough task because the obvious approach of guessing a set of colors and verifying that in the resulting graph 1 and n are connected via a path does not work: Recall that $W[t]$ is defined as the closure of p-WD_{φ} where φ is a first-order formula with a free second-order variable. With such a formula we can easily check whether the colors that are chosen with the set variable are k distinct colors and we can also compute the resulting graph, but we cannot compute whether the resulting graph contains a path from 1 to n, because reachability is not expressible using first-order formulas, see Papadimitriou (1994). But maybe there is an alternative approach for solving this problem? Parameterized space complexity tells us: there is not – if we make use of the reasonable assumptions that $W[\mathrm{SAT}] \not\subseteq W[t]$ for any t. The reason behind this is that p-Subset-Union-Reach and p-Subset-Union-Undirected-Reach are complete for para_W-NL and para_W-L, respectively, as we will see in a moment. Moreover, we have already seen that p-Weighted-SAT $\in \mathrm{para}_W$-NC^1 and, thus, we also have p-Weighted-SAT $\in \mathrm{para}_W$-L. Hence, showing that p-Subset-Union-Reach $\in W[t]$ or p-Subset-Union-Undirected-Reach $\in W[t]$ immediately implies $W[\mathrm{SAT}] \subseteq W[t]$. Let us now show the completeness results whose consequences we just discussed:

▷ *Theorem 40.* With respect to para-AC^0-reductions we have that

- p-Subset-Union-Reach is complete for para_W-NL,
- p-Subset-Union-Undirected-Reach is complete for para_W-L.

Proof. Let us consider the case of directed graphs, we can then prove the case of undirected graphs in a similar way.

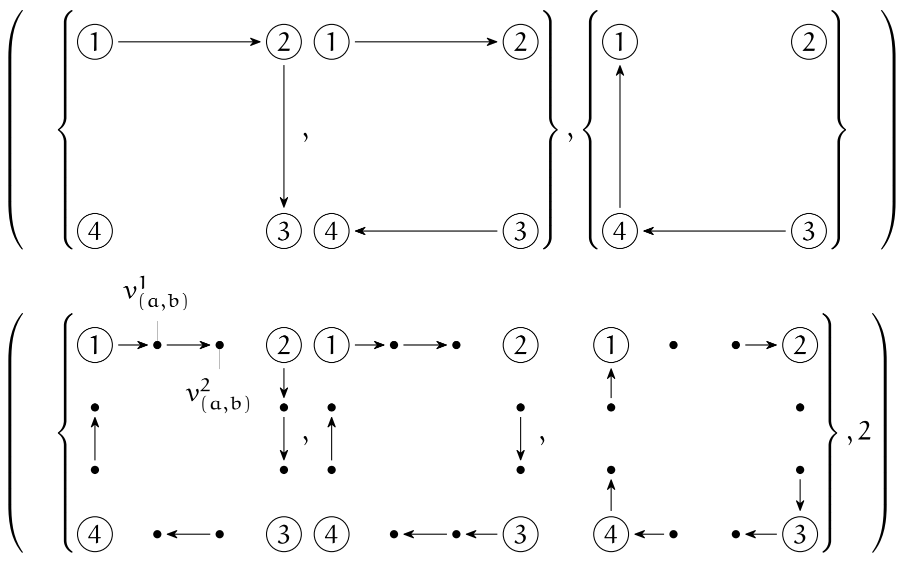
First of all, the classical reachability problem Reach is complete for NL, and, as has been shown by Immerman (1987), this completeness can be obtained using format-preserving AC^0-computable projections. From the union lemma we know that the problem p-Family-Union-Reach is complete for para_W-NL, and, thus, we only have to reduce it to p-Subset-Union-Reach. Hence, let $(S_1, \dots, S_k)$ be a family of sets of instantiations over the same template $?^{n^2}$ for some fixed n, and let us denote the edge relation encoded by an instance s with $E(s)$. Our task is now to create a single set of edge relations over the same vertex set such that there is a union of k of them that contain a path from 1 to n if, and only if, there are k relations, one from each S_i, whose union contain a path from 1 to n. Again, we cannot simply compute the union of the sets, because this might result in picking two relations from the same S_i. The main idea how to avoid this is to split the edges of the graph into chains of edges and distribute the parts of the edges among the edge relations in such a way that the corresponding path can only be formed if we pick one relation from every S_i. Formally, we define a new vertex set V' and edge relations E_s^i with $i \in \{1, \dots, k\}$ and $s \in S_i$ where $E_{\text{all}} = \bigcup_{s \in S_i \wedge i \in \{1,\dots,k\}} E(s)$ as follows:

$$V' = V \cup \big\{ v^i_{(a,b)} \mid (a,b) \in E_{\text{all}} \wedge i \in \{1,\dots,k\} \big\},$$
$$E_s^i = \big\{ (a, v^1_{(a,b)}) \mid (a,b) \in E(s) \big\} \cup \begin{cases} \big\{ (v^i_{(x,y)}, v^{i+1}_{(x,y)}) \mid (x,y) \in E_{\text{all}} \big\} & \text{if } i < k; \\ \big\{ (v^i_{(x,y)}, y) \mid (x,y) \in E_{\text{all}} \big\} & \text{if } i = k. \end{cases}$$

If $k = 1$, then we set $E_s^1 = E(s)$. The new instance for p-Subset-Union-Reach is then

$$\Big(\big\{ I(E_s^i) \mid s \in S_i \wedge i \in \{1,\dots,k\} \big\}, k \Big)$$

where $I(E)$ denotes the instantiation for the edge relation E. Let us consider an example for this construction. The top half of the next figure shows an instance for p-Family-Union-Reach and the bottom half shows the instance for p-Subset-Union-Reach contructed by the reduction from the instance above.



For every instantiation from the original instance the reduction constructs a single instantiation in the new instance, thus yielding a one-to-one correspondence between the instantiations. Let us denote the instantiation that is derived from s by s'. To prove the correctness of the reduction, it suffices to observe that for any selection of instantiations $s'_1, \dots, s'_k$ from the new instance there is a path $(a, v^1_{(a,b)}, \dots, v^k_{(a,b)}, b)$ with $\{a, b\} \subseteq V$ and $a \neq b$ in $\big(V', \bigcup_{i=1}^k E(s'_i)\big)$ if, and only if, the original s_i belong to k different sets in the original instance and $\big(V, \bigcup_{i=1}^k E(s_i)\big)$ contains the edge (a, b). Thus, let us first assume that $s_1, \dots, s_k$ originate from k different sets and that (a, b) is an edge in $\big(V, \bigcup_{i=1}^k E(s_i)\big)$. We can assume that $s_i \in S_i$. Since $(a, b) \in \big(V, \bigcup_{i=1}^k E(s_i)\big)$, there is one $E(s_i)$ with $(a, b) \in E(s_i)$. Thus, we have $(a, v^1_{(a,b)}) \in E(s'_i)$. Furthermore, since the s_i are from different sets, $\big(V', \bigcup_{i=1}^k E(s'_i)\big)$ contains the paths $(v^1_{(x,y)}, \dots, v^k_{(x,y)}, y)$ for every pair (x, y) in E_{all}. Hence, we have $(a, v^1_{(a,b)}, \dots, v^k_{(a,b)}, b) \in \big(V', \bigcup_{i=1}^k E(s'_i)\big)$. Let us now assume that for a selection $s'_1, \dots, s'_k$ from the new instance the graph $\big(V', \bigcup_{i=1}^k E(s'_i)\big)$ contains a path $(a, v^1_{(a,b)}, \dots, v^k_{(a,b)}, b)$ with $(a, b) \in V^2$. Then, one of the s'_i has to contain the edge $(a, v^1_{(a,b)})$ and, therefore, $E(s_i)$ contains (a, b). It remains to show that no two elements of $\{s'_1, \dots, s'_k\}$ originate from the same S_i. Thus, for a contradiction, assume that two elements s'_α and s'_β originate from S_γ. Then, there is a set S_δ such that $S_\delta \cap \{s_1, \dots, s_k\} = \emptyset$. This means that $\big(V', \bigcup_{i=1}^k E(s'_i)\big)$ does not contain a single edge leaving any vertex $v^\delta_{(x,y)}$. Hence, there cannot be a path from a to b in $\big(V', \bigcup_{i=1}^k E(s'_i)\big)$, which is absurd because we assumed that there is a path from a to b, and, thus, we can conclude that for the selection $s'_1, \dots, s'_k$ the elements $s_1, \dots, s_k$ are from different sets in the original instance. Overall, we can conclude that we have $(S_1, \dots, S_k) \in$ p-FAMILY-UNION-REACH if, and only if, we have $\big(\{\, s' \mid s \in S_i \wedge i \in \{1, \dots, k\}\}, k\big) \in$ p-SUBSET-UNION-REACH. □

▷ *Corollary 41.* With respect to para-AC^0-reductions we have that

- p-COLORED-REACH is complete for para$_{\text{W}}$-NL,
- p-COLORED-UNDIRECTED-REACH is complete for para$_{\text{W}}$-L.

The problems REACH and UNDIRECTED-REACH are probably the most well-known NL-complete and L-complete problems, respectively, and, overall, the completeness of the family-union and subset-union problems for the accompanying classes of parameterized bounded nondeterminism is a fairly natural result. However, these results raise the question of which other problems are complete for these classes. The following theorem gives a first answer to this question.

▷ *Theorem 42.*

1. The following problems are complete for para_W-NL with respect to para-AC^0-reductions:
   - p-Subset-Union-DAG-Reach,
   - p-Subset-Union-Cycle.
2. The following problems are complete for para_W-L with respect to para-AC^0-reductions:
   - p-Subset-Union-Tree,
   - p-Subset-Union-Forrest,
   - p-Subset-Union-Undirected-Cycle.

For all these problems their accompanying family-union problems are also complete for the corresponding class. Here, the underlying problems are

- DAG-Reach, the reachability problem for directed acyclic graphs,
- Cycle, the problem of deciding whether a given directed graph contains a cycle,
- Undirected-Cycle, the same as Cycle but for undirected graphs,
- Tree, the problem of deciding whether a given undirected graph is a tree, i. e., connected and cycle-free,
- Forrest, the problem of deciding whether a given undirected graphs is a forrest, i. e., cycle-free.

Proof. We can prove these results basically in the same way as in Theorem 40 where we showed completeness of p-Subset-Union-Reach for para_W-NL. With the fact that the above problems are all complete with respect to AC^0-computable format-preserving projections, see Immerman (1999) for more details, what remains is to give a reduction from the corresponding family-union problem to the subset-union version. For nearly all of of these problems, we can use the reduction presented above, the only exception is p-Subset-Union-Forest. The problem here is, that choosing two instantiations that stem from the same S_α of the family-union instance, the resulting graph becomes a collection of paths, i. e., a forrest, and this is exactly what may not happen. Hence, we use a different reduction: For every s_i and s_j with $s_i \neq s_j$ from the same S_α we add three vertices to the graph. Let us call these vertices a, b, and c. Then, we add the edges (a, b) and (b, c) to $E(s_i')$ and the edge (c, a) to $E(s_j')$. This way, when we chose s_i' and s_j', the resulting graph contains a cycle and, therefore, is not a forrest. On the other hand, if we do not pick such a pair, then the new vertices are only engaged into paths of length 1 or 2 and have no influence on whether the graph is a forrest or not. $\square$

Still, problems like TREE or CYCLE can be seen as special kinds of reachability problems, and it is therefore not surprising that they have the same complexity as the underlying reachability problems for directed and undirected graphs. Hence, let us turn to a problem that, on first sight, has nothing to do with reachability in graphs: the problem of *associative generability*, a restricted version of the more general generability problem GEN. In GEN we are given a finite set U together with a mapping $\circ\colon U^2 \to U$, an element x with $x \in U$, the *target element*, and a set G with $G \subseteq U$, the set of *generators*. The question is whether we can generate x from the generators, i. e., whether the smallest superset of G that is closed under $\circ$ contains x. If we restrict our attention to functions $\circ$ that are associative, we get the associative generability problem AGEN.

Both problems, GEN and AGEN have been fully classified with respect to their classical computational complexity: Jones and Laaser (1976) showed that GEN is complete for P, and Jones et al. (1976) showed that the AGEN is complete for NL. In the parameterized setting, we study the problems p-GEN and p-AGEN.

▷ *Problem 43 (Parameterized Generability and Associative Generability).*

Instance: A finite set U, a *target element* x with $x \in U$, a set of *generator candidates* C with $C \subseteq U$, a binary operator $\circ\colon U^2 \to U$ given as a table, and a natural number k.

Parameter: k

Question: Is there a set G with $G \subseteq C$ and $|G| = k$ such that the smallest superset of G that is closed with respect to $\circ$ contains x?

We denote this problem with p-GEN and the variant where we restrict the operator $\circ$ to be associative with p-AGEN.

Flum and Grohe (2006) studied the problem p-GENERATORS, a variant of p-GEN where no generator candidates are given, but the task is to find a set of k generators that generates all of U. They showed that this problem is complete for para_W-P, and it is not hard to see that, even though the problem definition slightly changes, this essentially also proves that p-GEN is complete for para_W-P. If we turn to p-AGEN, it is not surprising that this problem is, similarly to the classical case, complete for para_W-NL.[1]

▷ *Theorem 44.* p-AGEN is para_W-NL-complete under para-AC^0-reductions.

[1]However, it is still unclear whether p-AGENERATORS, the associative version of p-GENERATORS, is complete for para_W-NL.

Proof. Given an instance of p-AGEN, a para_W-NL-machine interprets the content of its choice tape as a description of the set of generators. Due to the associativity of $\circ$ the machine can check the generability of the target element by successively guessing the elements of a sequence $g_1, g_2, \ldots$ of generators and computing the values

$$(g_1), (g_1 \circ g_2), ((g_1 \circ g_2) \circ g_3), (((g_1 \circ g_2) \circ g_3) \circ g_4), \ldots.$$

For this, it suffices to only store the lastly computed element, which can be done in logarithmic space. Note that it is not necessary to guess a sequence of more than $|U|$ elements to generate x, because otherwise the resulting sequence of values $(g_1), (g_1 \circ g_2), ((g_1 \circ g_2) \circ g_3), \ldots$ contains an element twice, i. e., there is a shorter sequence that generates x.

Concerning hardness, we will proceed like in the proofs above: First, we show that p-FAMILY-UNION-AGEN is complete for para_W-NL using the union lemma. Afterwards we describe the reduction chain

$$\begin{aligned}\text{p-FAMILY-UNION-AGEN} &\leq \text{p-SUBSET-UNION-AGEN}\\ &\leq \text{p-WEIGHTED-UNION-AGEN}.\end{aligned}$$

Finally, we argue that p-WEIGHTED-UNION-AGEN is equivalent to p-AGEN.

Before we discuss p-FAMILY-UNION-AGEN, we have to fix a suitable encoding of AGEN that enables us to apply the union operation. The crucial part is how to encode the set of generators because it is the only part that will be affected by the union operation. Hence, we encode U, $\circ$, and x using Σ in a reasonable way, and add a ? for each element of U that may be part of the generators. If such a symbol is replaced by 1, the corresponding element of U is considered to be in G, if it is replaced by 0, the element is considered to be not in G. Note that by using this encoding it is possible to specify which elements of the universe can be chosen to be a member of the generators because we allow that not every element has a corresponding ? in the template.

Let us start with p-FAMILY-UNION-AGEN. Jones et al. (1976) showed that AGEN is complete for NL, and it is not difficult to see that their reduction is indeed a format-preserving AC^0-computable projection: They give a reduction from the reachability problem for directed graphs where the universe of the resulting instance of AGEN essentially encodes all possible edges between the vertex set $\{1, \ldots, n\}$, the target element is the edge from the start vertex to the target vertex, the associative operation maps pairs of edges to its transitive edges or, if no such transitive edge exists, to a certain error element, and the set of generators is the set of edges that are actually contained in the given graph. Hence, different input graphs with the same encoding length, i. e., with the same number of vertices, are reduced to instantiations of the same template where only the bits selecting the available generator elements change from instance to instance. From the union-lemma we thus get that p-FAMILY-UNION-AGEN is complete for para_{W}-NL with respect to para-AC^0 reductions.

We now reduce p-FAMILY-UNION-AGEN to p-SUBSET-UNION-AGEN. For this, let the input be a family of sets $S_1, \dots, S_k$ of instantiations of a common template that encodes a universe U, a target element x, an associative operation $\circ$, and a set of generator candidates C. Then, every instantiation additionally encodes a set of generators. Our aim is to construct an instance of p-SUBSET-UNION-AGEN, i. e., a new set S' of instantiations of a common template that encodes a universe U', a target element x', an associative operation $\circ'$, and a set of generator candidates C' such that there are k elements $s_1, \dots, s_k$ with $s_i \in S_i$ whose union is contained in AGEN if, and only if, there are k elements in S' and their union lies in AGEN. The crucial part here is – once more – that we cannot simply set $S' = \bigcup_i S_i$, because this would allow picking two instantiations that stem from the same S_i. To fix this, we first set $U' = U \cup \{e_1, \dots, e_k\}$ and $C' = C \cup \{e_1, \dots, e_k\}$ for new elements e_i. We then augment the operator $\circ$ to $\circ'$ by defining it on these new elements such that none of the new elements can be generated from any two other elements. Moreover, we add a new target element x' to the universe that can only be generated via the expression $x \circ' e_1 \circ' e_2 \circ' \cdots \circ' e_k$. Finally, we add a new element `error` to the new universe that we will use to catch the evaluation of undesired expressions by augmenting $\circ'$ such that any expression that is not desired or already contains `error` is evaluated to `error`. The new set S' then contains a string $s'_{i,j}$ for every $s_{i,j}$ with $s_{i,j} \in S_i$ that is essentially $s_{i,j}$ adjusted to U', x', $\circ'$, C'. Moreover, we require that $s'_{i,j}$ additionally selects e_i as a generator and no other of the newly introduced elements $e_{i'}$ with $i \neq i'$. This construction enforces that we, in order to generate x', have to pick k strings $s'_{i_1,j_1}, s'_{i_2,j_2}, \dots, s'_{i_k,j_k}$ with $i_a \neq i_b$ for $a \neq b$, i. e., strings that stem from pairwise different sets of the original instance. Hence, there is a selection of k elements, one from each $S_1, \dots, S_k$, such that their union lies in AGEN if, and only if, there is a selection of k elements from S' such that their union lies in AGEN. However, we are not done yet.

Unfortunately, the new operator $\circ'$ is not a binary operator, and, therefore, we cannot evaluate expressions like $x \circ' e_1 \circ' \cdots \circ' e_k$ in one step. To make it even worse, $\circ'$ has to be associative and, hence, it has to be possible to evaluate every subexpression of a larger expression. In order to achieve these properties, we conceptually use strings as elements of our universe that represent expressions that are evaluated "as far as possible". For example, let us consider the expression $a \circ' b \circ' c \circ' e_1 \circ' e_2$ where $a \circ b \circ c$ does not evaluate to the old target element x and we have $k > 2$. The "furthest possible evaluation" is then $d \circ' e_1 \circ' e_2$ for some element d. Hence, we want a representation of the expression $d \circ' e_1 \circ' e_2$ to be part of the universe. To be exact, we formalize this approach using *replacement systems*.

Given an alphabet Γ, we call a set R of rules $w \to w'$ with both $w \in \Gamma^*$ and $w' \in \Gamma^*$ a *replacement system*. An *application* of a rule $w \to w'$ to a word uwv yields the word $uw'v$, and we write $uwv \Rightarrow_R uw'v$. We call a word *irreducible* if it is not possible to apply a rule to it. Let us call $\equiv_R$ the reflexive, symmetric, transitive closure of the relation $\Rightarrow_R$, and define the equivalence classes of $\equiv_R$ by $[u]_R = \{v \mid v \equiv_R u\}$ for any word u over Γ. Finally, an *irreducible representative system* of R is a set of irreducible words that contains exactly one word from each equivalence class of Γ^*. Our aim is now to replace the universe U' of our instance for p-SUBSET-UNION-AGEN by such an irreducible representative system and let the corresponding operator represent the replacement rules defined on this system.

For the reduction, we set Γ to U'. Then, R contains the following rules:

$$\begin{aligned}
ab &\to c \quad \text{for elements } a, b, c \text{ with } a, b, c \in U \text{ and } a \circ b = c,\\
xe_1e_2\ldots e_k &\to x',\\
\mathtt{error}u &\to \mathtt{error} \quad \text{for every } u,\\
u\mathtt{error} &\to \mathtt{error} \quad \text{for every } u,\\
e_iu &\to \mathtt{error} \quad \text{for every } u \text{ with } u \in (U' - \{e_{i+1}\}),\\
x'u &\to \mathtt{error} \quad \text{for every } u.
\end{aligned}$$

With these rules we can finally define the output $(U'', \circ'', x'', G'')$ of our reduction from p-FAMILY-UNION-AGEN to p-SUBSET-UNION-AGEN: The universe U'' is an irreducible representative system of R, the operation $\circ''$ maps a pair (u, v) to the representative of $[u \circ v]_R$ in U'', the target element x'' is the representative of $[x']_R$ in U'', and C'' contains one representative of $[c]_R$ in U'' for each $c \in C'$. We have already argued that the reduction underlying the resulting representative system is correct, thus what remains is to show that the reduction is indeed a para-AC^0 reduction.

To see that our reduction is in fact a para-AC^0 reduction, let us first consider the size of the universe U''. The size of the universe, i. e., the number of equivalence classes is at most $1 + |U'|(k^2 + 1)$. To see this, let us consider any equivalence class $[w]_R$ and let w be irreducible. Then, w can be `error`. If w is not `error`, then it cannot contain `error` together with other symbols, because it would not be irreducible in this case. Moreover, in w there cannot be any element of U to the right of any e_i or x', because this would be evaluated to `error` and, hence, w would, again, not be irreducible. Therefore, w has to be of the form $w_1 w_2$ where, due to its irreducibility, w_1 must be a single letter and w_2 has to be x' or a sequence $e_i e_{i+1} \dots e_j$ with $i \leq j$. This shows the polynomial bound claimed above. Moreover, computing the new generators e_i, the error element, the new target element x', the table defining the operator $\circ'$, and the resulting representative system is tedious but possible using parameterized AC^0 circuit families. Overall, we can conclude that p-FAMILY-UNION-AGEN reduces to p-SUBSET-UNION-AGEN.

Let us now turn to the second part of the reduction chain, namely the reduction from p-SUBSET-UNION-AGEN to p-WEIGHTED-UNION-AGEN. We are given a set S of instantiations of a common template that encodes a universe U, a binary associative operator $\circ\colon U^2 \to U$, a target element x, and a set of generator candidates C. Our task is now to compute the template that encodes a new universe U', a new binary operator $\circ'$, a new target element x', and a new set of generator candidates C', such that there is selection of k instantiations $s_1, \dots, s_k$ from S whose union lies in p-AGEN if, and only if, there is an instantiation of the new template that lies in p-AGEN and that has weight $f(k)$ for some function f. The problem here is, that in the instance of p-SUBSET-UNION-AGEN a single instantiation can select an arbitrary number of generators, but in an instance of p-WEIGHTED-UNION-AGEN the number of elements that we may select is fixed to the parameter. The overall idea for our reduction thus is to essentially use the old universe, operator, and target element, and add new elements to the universe that enable us to enumerate the generator elements that a single instantiation of the input instance can select. In other words, if an instantiation s selects the generators a, b, c, we add a single new element to the universe that enables us to enumerate a, b, c by using appropriate expressions. Then we only have to select k of these enumerating elements, but we are still able to enumerate an arbitrary number of generators. Moreover, like in the previous reduction, we make use of replacement rules with an appropriately chosen set of irreducible representatives as our universe. Let us get into the details and discuss the new elements of the universe:

1. We have an *error element* `error` with similar rules as above to catch unintended expressions.

2. We have a new *end element* $\lhd$ that we require for technical reasons and that we will use to mark the end of expressions. The element $\lhd$ is not generated by any expression, i. e., it is not on the right-hand side of a replacement rule, if it is not already on the left-hand side. Hence, $\lhd$ has to be element of any generating set. Moreover, we have the rules $\lhd u \to \texttt{error}$ for every u with $u \in U'$.
3. We have a new *counter element* $|$. Like $\lhd$, this element cannot be generated and, therefore, has to be an element of any generating set.
4. We have new *selector elements* σ_i, one for each s_i with $s_i \in S$. Together with the counter element $|$ we will use these selector elements to enumerate the elements $u_1, \ldots, u_l$ that the corresponding instantiation s_i selects. Our aim is, that an expression like $\sigma_i|||$ evaluates to u_3, the third generator element selected by s_i.

In the resulting template, the set of generator candidates is then the set of representatives of error, $\lhd$, $|$, and σ_i for every i. Hence, there will be a selection of $k + 3$ generators that suffice to generate the target element if, and only if, there is a selection of k elements of S such that their union lies in AGEN. Let us now discuss the replacement rules and the definition of the binary operator on the new elements.

Consider the two expressions

$$\sigma_1||\sigma_3|||\sigma_2|| \qquad\text{and}\qquad \sigma_1||\sigma_3|||\sigma_2|||$$

and let the values of these expressions be well-defined elements of the universe. Note that the first expression is a subexpression of the second one because the second has an additional counter element at the right end. Since the operator $\circ'$ has to be associative, it must be possible to completely evaluate any subexpression of these expression. Hence, if $\sigma_1||\sigma_3|||\sigma_2||$ evaluates to the representative of an element e of the original universe, it must be possible to evaluate the expression $e|$, but, unfortunately, the value of this expression is not well-defined by $\circ$ or by our replacement rules. Moreover, since e can be selected by different selector elements, say we have $\sigma_1| = e$ and $\sigma_2|| = 3$, we cannot fix an e' that $e|$ evaluates to, because we may have $\sigma_1|| \neq \sigma_2|||$. At this time, all we know is that $\sigma_1||$ and $\sigma_3|||$ are well-defined generators, but this is just because we already know that there are no counter elements directly on the right next to them. If we are presented a pure subexpression like $\sigma_2||$, we cannot fully evaluate it to an element of the universe because we do not “know” whether there will be a symbol on the right of this expression or what this symbol is. To fix this issue, we make use of the end symbol $\lhd$. This symbol has to be placed at the end of every expression, and enables us to unambiguously evaluate it. In terms of replacement rules, if $\sigma_i|||$ should select u_3, we have the rules

$$\sigma_i|||u \to u_3 u \quad \text{if } u \neq |.$$

Hence, strings like $\sigma_i|||$ are irreducible. Moreover, to ensure that we do not get overly long sequences of counter elements, we add the rule

$$|^n \to \texttt{error} \quad \text{with } n = |U|.$$

Finally, we set the target element to $x\lhd$.

Let us argue that the number of equivalence classes is polynomially in the size of the universe U. For this, note that irreducible strings start and end with at most n counter elements and, in the middle, have a single element of the universe that is possibly followed by a selector element. In toto, the size of the resulting irreducible representative system is at most polynomial. Again, it is not hard to see that this reduction is AC^0-computable, thus giving us the theorem. □

Before we start using classes of bounded nondeterminism to study problems within para-P in the next section, let us revisit a problem from Section 2.2, namely the feedback vertex set problem. We have already seen in Theorem 12 that p-DFVS $\in$ para-L implies that L $=$ NL, and p-FVS $\in$ para-NC^1 implies $\mathrm{NC}^1 = \mathrm{L}$. Using classes of bounded nondeterminism we can improve and strengthen these result:

▷ *Theorem 45.*

1. p-FVS $\in$ $\mathrm{para}_{\mathrm{W}}$-L.
2. p-DFVS $\in$ $\mathrm{para}_{\mathrm{W}}$-NL.
3. p-FVS $\in$ $\mathrm{para}_{\mathrm{W}}$-$\mathrm{NC}^1$ if, and only if, $\mathrm{NC}^1 = \mathrm{L}$.
4. p-DFVS $\in$ $\mathrm{para}_{\mathrm{W}}$-L if, and only if, L $=$ NL.

Proof. To see that p-FVS $\in$ $\mathrm{para}_{\mathrm{W}}$-L, let (G, k) with $G = (V, E)$ be an instance of p-FVS. A $\mathrm{para}_{\mathrm{W}}$-L-machine can now interpret its $O\big(f_k \cdot \log(n)\big)$ nondeterministic bits on the choice tape as the description of a possible feedback vertex set of size k of G. Hence, the machine only has to check whether G without these vertices is a forest, which can be done in logarithmic space, see Cook and McKenzie (1987). With essentially the same arguments we can show that p-DFVS $\in$ para_{W}-NL. Here we only have to argue that testing whether a directed graph is a forest can be done in nondeterministic logarithmic space. This, however, directly follows from Immerman (1988).

The proofs of items 3 and 4 are essentially the same as the proofs of Theorem 12: If we had p-FVS $\in$ para_{W}-NC^1, then we can use the corresponding circuit family to solve p_0-CYCLE-FREE, the problem of deciding whether a given undirected graph is cycle-free with the trivial parameterization. In contrast to the proof of Theorem 12 our circuit now also has access to $O\big(f_k \cdot \log(n)\big)$ nondeterministic bits. However, since we use the trivial parameterization, the number of nondeterministic bits is effectively in $O\big(\log(n)\big)$. This means that we can get rid of the requirement for nondeterministic bits by creating polynomially many copies of the circuit, one for each possible choice of the nondeterministic bits, and testing in parallel whether one of them accepts. Overall, the resulting circuit family is effectively a NC^1 circuit family, thus giving us CYCLE-FREE $\in \text{NC}^1$. On the other hand, if $\text{NC}^1 = \text{L}$, then we also have para_{W}-$\text{NC}^1 = \text{para}_{W}$-L and together with item 1, that p-FVS $\in$ para_{W}-NC^1. Item 4 can be shown in a similar way. $\square$

Let us conclude this section with a discussion on the relations between the classes of bounded two-way nondeterminism studied in this section and the Weft-Hierarchy. This discussion is of particular interest because both the classes of this chapter and the Weft-Hierarchy share two-way bounded nondeterminism as a central and fundamental aspect of them: While this is obvious for the classes of this chapter, recall that a problem is conceptually a “Weft problem” if we can solve it using a preprocessing phase, followed by a guessing phase using a bounded amount of nondeterminism, followed by a verification phase. Thus, the classes of the “Weft-Hierarchy” are inherently connected to the concept of bounded nondeterminism which immediately leads to the question of how all these classes are related to each other and what hardness or completeness results of problems for classes like para_{W}-L tell us about their relations to Weft classes and vice versa. Let us start our investigation with the lower classes of the Weft-Hierarchy, i. e., with the classes $W[t]$. We defined these classes as the para-P-reduction closures of problems p-WD_{φ} with $\varphi \in \Pi_t$, a definition that directly exhibits the pattern of Weft problems. Note now that the preprocessing phase corresponds to the reduction closure and that the verification phase corresponds to the evaluation of the underlying formula φ, i. e., for every fixed problem the evaluation of a fixed first-order formula. With the well-known fact that for the class of first-order definable sets FO we have $\text{FO} = \text{AC}^0$, see for example Vollmer (1999) or Immerman (1999), it follows immediately that $W[t]^{\text{para-AC}^0} \subseteq \text{para}_W\text{-AC}^0$. Hence, if we show that a problem is hard for $\text{para}_W\text{-AC}^0$, it is also hard for $W[t]$ for every t. However, we cannot conclude that $W[t] \subseteq \text{para}_W\text{-AC}^0$ because $\text{para}_W\text{-AC}^0$ is not closed under para-P-reductions. Note that we, implicitly, have already seen a result of this kind before when we discussed that $W[\text{SAT}]$ is the para-P-closure of $\text{para}_W\text{-NC}^1$: If problem is hard for $\text{para}_W\text{-NC}^1$, then it is also hard for $W[\text{SAT}]$ – and in this latter case this statement also holds if we substitute "hard" with "complete".

In both of the previously studied cases, namely that $\text{para}_W\text{-AC}^0$-hardness implies $W[t]$-hardness and that $\text{para}_W\text{-NC}^1$-hardness or completeness implies $W[\text{SAT}]$-hardness or completeness, respectively, the para-P-reduction in the definition of the Weft-Hierarchy proves to be a very powerful component. On the other hand, the verification phase of the classes of the Weft-Hierarchy is relatively weak compared to classes like $\text{para}_W\text{-L}$. In toto, these two aspects make it hard to compare the Weft-Hierarchy and the hierarchy of classes of bounded two-way nondeterminism. For example, $W[\text{SAT}] \subseteq \text{para}_W\text{-NL}$ implies that $\text{para-P} \subseteq \text{para-NL}$ and, thus, $\text{NL} = \text{P}$. On the other hand, it is completely unclear whether $\text{para}_W\text{-L} \subseteq W[\text{SAT}]$ is true or not.

While the classes of the Weft-Hierarchy and the classes of bounded two-way nondeterminism share a common foundation, they appear to be very different and largely incomparable. On first sight this might seem to be a shame, but, on the other hand, this actually enriches the theory because their (presumably) orthogonal structure allows us a more refined view on the complexity of parameterized problems than a linear structure of complexity classes.

3.3. Natural Problems for para_β-Classes

While classes like $\text{para}_W\text{-L}$ or $\text{para}_W\text{-NL}$ perfectly capture the complexity of natural problems like the colored reachability problem p-COLORED-REACH or the associative generability problem p-AGEN, these classes have the drawback that they presumably lie orthogonal to the fundamental class para-P. Hence, if we want to use these classes of bounded nondeterminism to study problems within para-P, we have to study the intersections of para-P with these classes. However, this is somehow unsatisfying because, under reasonable assumptions, this implies that we will not be able to show completeness of problems within those intersections for any of the classes involved in the definition of the intersection. If we want to study the complexity of problems within para-P, it is therefore preferable to use subclasses like para-L. Unfortunately, many natural problem do not seem to fit to classes like para-L or para-NL. One of these problems is the parameterized distance problem p-DISTANCE which asks whether two given vertices of a given graph have at most a given distance. In this section, we show that this problem is complete for para_β-L, a parameterized space class using a bounded amount of nondeterminism, and we study natural variants of p-DISTANCE, showing that subtle differences in the problem definition yield problems of many different complexities that, however, can all be characterized using parameterized space classes. Let us start our investigation with a formal definition of the parameterized distance problem p-DISTANCE:

▷ *Problem 46 (Parameterized Distance).*

Instance: A directed graph G, two vertices s and t of G, and a natural number k.

Parameter: k.

Question: Is there a path from s to t in G of length at most k?

▷ *Theorem 47.* p-DISTANCE is para_β-L-complete under para-L-reductions.

Proof. To solve p-DISTANCE using a para_β-L-machine, the machine successively guesses a sequence $v_1, v_2, \dots, v_{k+1}$ of vertices with $s = v_1$ and $v_{k+1} = t$, and checks whether the graph contains edges (v_i, v_{i+1}) for $i \in \{1, \dots, k\}$. For an instance of n vertices, the machine then requires $O\big(k \cdot \log(n)\big)$ nondeterministic bits and space $O\big(\log(k) + \log(n)\big)$ to store the number of already guessed vertices and a description of the current vertex of the sequence. We thus have p-DISTANCE $\in \text{para}_\beta$-L.

Let us now prove hardness. For this, we show how to reduce any problem in para_β-L to p-DISTANCE. Hence, let (Q, κ) be such a problem and M be a para_β-L-machine that decides Q. Using standard techniques, we may assume that M has exactly one accepting configuration and that its configuration graph is always acyclic. The main idea for the reduction is now, given an input x, to compute the configuration graph of M on x and ask whether the distance from the single initial configuration to the single accepting configuration in the configuration graph is at most a certain value l. The question now is: what is this value?

Since the space bound of M lies in $O\big(f_k + \log(n)\big)$, the size of its configuration graph lies in $O(f'_k \cdot n^c)$ for a constant c and a function f'. Hence, l can be at most this large. However, there is a problem here: Since l is the parameter, we have to ensure that l only depends on the old parameter and not on the size of the input instance – which is not the case here. To tackle this problem, we do some processing on the configuration graph: Because the underlying machine M may use at most $O\big(f_k \cdot \log(n)\big)$ nondeterministic bits, on every path starting at the node of the initial configuration in the acyclic configuration graph there are at most $O\big(f_k \cdot \log(n)\big)$ nodes that have an out-degree larger than 1. Let us call these nodes together with the nodes corresponding to the initial and the accepting configuration the *red nodes* and the remaining nodes the *black nodes*. In a first step, we contract paths of black vertices. For this, we iterate over the nodes of the configuration graph and, for every red vertex u_r, we replace every outgoing edge (u_r, v) with an edge (u_r, v_r) to the first red node v_r reachable from v. Now, we drop the black vertices. In the resulting graph, the length of every path starting from the node of the initial configuration lies in $O\big(f_k \cdot \log(n)\big)$, which still is not exclusively bounded in the parameter. Hence, in a second step, we get rid of the $O\big(\log(n)\big)$ by augmenting the graph with shortcuts that allow "jumps" of logarithmic length: For every vertex v, we iterate over all paths of length $O\big(\log(n)\big)$ starting at v and insert an edge to every vertex w that we reach during this process. Since the maximal out-degree of the vertices is determined by the machine M, namely by the maximum number of different states reachable in a nondeterministic step, the maximal out-degree of the graph is constant. This allows us to describe every path of logarithmic length using $O\big(\log(n)\big)$ space. In the resulting graph, there then is a path from the vertex of the initial configuration to the vertex of the accepting configuration of length $O(f_k)$ if, and only if, there is a path from the vertex of the initial configuration to the vertex of the accepting configuration in the original configuration graph. Because each of the steps above can be performed in parametric logarithmic space, we obtain a para-L-reduction. □

So far, every time we considered graph problems, we could observe that the complexity of the problem drops if we switch from directed to undirected graphs. An interesting problem that shows a special behaviour and, thus, does not fit to this pattern is the distance problem. It is well-known that the unparameterized distance problem is complete for directed as well as undirected graphs, see Tantau (2005). This carries over to the parameterized setting:

▷ *Theorem 48.* p-UNDIRECTED-DISTANCE is para_β-L-complete with respect to para-L-reductions.

Proof. To show that p-UNDIRECTED-DISTANCE lies in para_β-L, we proceed like in the previous theorem: We let our machine nondeterministically guess a path of at most the given length. To show hardness, we reduce from the distance problem in directed graphs, namely p-DISTANCE: For a given directed graph G with $G = (V, E)$, two vertices s and t, and a given number l, we construct a new undirected graph G' with vertices s' and t' such that there is a path from

s to t in G of length at most l if, and only if, there is a path of length l from s' to t in G'. We construct l copies of the vertices of the original input graph and interpret them as layers. Let us denote the i-th copy of vertex v by v_i. We then insert an edge between v_i and w_{i+1} if, and only if, there is an edge from v to w in the original graph, thus making the connections from one layer to the next layer reflect the original edge relation. Moreover, we insert edges between v_i and v_{i+1} for every v and every i that allow us to move down the layers without switching the currently considered vertex. Now, since "going back" one layer and, therefore, using an edge in the "wrong direction" prolongs the resulting path by at least 2 steps, there is a path from s_1 to t_l of length l in G' if, and only if, there is a path from s to t in G of length at most l. $\square$

What happens if, instead of asking whether the distance is at most l, we ask whether the distance between two nodes is *at least* l, i. e., what is the complexity of the problem p-Distance-At-Least:

▷ *Problem 49 (Parameterized Least Distance).*

Instance: A directed graph G, two vertices s and t of G, and a natural number k.

Parameter: k.

Question: Is the distance of t from s in G at least k, i. e., there is no path from s to t of length less than k?

Clearly, the problem p-Distance-At-Least is essentially the complement of p-Distance, and, therefore, lies in co-para_β-L. Moreover, since p-Distance is complete for para_β-L, we can also conclude that p-Distance-At-Least is also complete for co-para_β-L. But what exactly is co-para_β-L? For a problem (Q, κ) in para_β-L we require that, for any instance x, we have $x \in Q$ if, and only if, there is a binary string b of length $O(f_k \cdot \log(n))$ such that the para-L machine M deciding (Q, κ) accepts with b on its choice tape. Equivalently, we can require that, for any instance x, we have that $x \notin Q$ if, and only if, our machine rejects for every such binary string b on its choice tape. Hence, and because of the well-known fact that L is closed under complement, the class co-para_β-L can equivalently be defined as the class $\text{para}\forall^{\rightarrow}_{\text{flog}}$-L, which we abbreviate in the following with $\text{para}_{\beta\forall}$-L. This notion also gives us a much more intuitive understanding of why p-Distance-At-Least is contained in $\text{para}_{\beta\forall}$-L: A corresponding machine verifies for every content of the choice tape that if it is a valid description of a path in the input graph, then this path does not lead to the target vertex. Note that with essentially the same arguments and the fact that both the directed as well as the undirected version of the distance problem are complete for para$_\beta$-L, we can also conclude that p-UNDIRECTED-DISTANCE-AT-LEAST is complete for para$_{\beta\forall}$-L.

p-DISTANCE-AT-LEAST asks whether the distance of the target vertex from the start vertex is at least a certain value where we consider the distance to be infinite if the target vertex is not reachable from the start vertex. In many cases, however, this is not what we are looking for. Typically, asking whether the distance between two vertices is at least a certain value implies that we still require that there is path from the start to the target vertex. Between these two closely related problems of asking whether the graph contains no short path and the graph contains is no short but still a long path there is, however, a notable gap in complexity. To study this gap, let us first formalize p-LARGE-DISTANCE, the variant of the distance problem we are discussing:

▷ *Problem 50 (Parameterized Large Distance).*

Instance: A directed graph G, two vertices s and t of G, and a natural number k.

Parameter: k.

Question: Is there a path from s to t in G, but the length of every such path is at least k?

▷ *Theorem 51.* p-LARGE-DISTANCE is complete for para-NL with respect to para-L-reductions.

Proof. To prove this, we first show that p-LARGE-DISTANCE lies in para-NL, and then that there is a NL-complete slice of p-LARGE-DISTANCE which, together with a useful technical lemma, suffices to show para-NL-completeness. Hence, let us start with membership.

The central idea of how to show membership is based on a technique of Immerman (1988): Using double-inductive counting, we successively enumerate the vertices of the input graph with larger and larger distance from the start vertex. During this enumeration, we make sure that the target vertex is not reachable within $k-1$ steps from the start vertex, but is reachable at last, and we accept and reject accordingly.

For hardness, we make use of a lemma that is based on a theorem of Flum and Grohe (2006):

▷ *Lemma 52.* Let (Q, κ) be a nontrivial parameterized problem in para-NL. Then (Q, κ) is complete for para-NL (with respect to para-L-reductions) if, and only if, there are natural numbers $m_1, \dots, m_l$ such that the language

$$\bigl\{\, x \mid x \in Q \wedge \kappa(x) \in \{m_1, \dots, m_l\}\bigr\}$$

is complete for NL with respect to L-computable reductions.

Proof. For the first direction, let (Q, κ) be a nontrivial para-NL-complete problem over the alphabet Σ, and let Q' be an NL-complete problem over the alphabet Σ'. Since Q' lies in NL, we have $(Q', \kappa_{\text{one}}) \in$ para-NL for the trivial parameterization κ_{one} that maps everything to the fixed value 1, and, since (Q, κ) is complete for para-NL, there is a para-L-computable reduction $r\colon (\Sigma')^* \to \Sigma^*$ from $(Q', \kappa_{\text{one}})$ to (Q, κ). Due to the trivial parameterization, we can, for any x, compute $r(x)$ in space $f(1) + O\bigl(\log |x|\bigr)$ for some function f, and we have $\kappa(r(x)) \le g(1)$ for some function g. Hence, r is in fact a reduction from Q' to $\bigl\{\, x \mid x \in Q \wedge \kappa(x) \in \{1, \dots, g(1)\}\bigr\}$ that is computable in logarithmic space. Since Q' is complete for NL, this implies NL-completeness of the language $\bigl\{\, x \mid x \in Q \wedge \kappa(x) \in \{1, \dots, g(1)\}\bigr\}$.

For the other direction, assume that $\bigl\{\, x \mid x \in Q \wedge \kappa(x) \in \{m_1, \dots, m_l\}\bigr\}$ with $Q \subseteq \Sigma$ for an alphabet Σ is NL-complete. Now, let (Q', κ') be a problem in para-NL over the alphabet Σ'. From the definition of para-NL we then have that there is a function $\pi\colon \mathbb{N} \to \Pi^*$ for some alphabet Π and a language X in NL such that for every x with $x \in \Sigma'$ we have $x \in Q'$ if, and only if, $\bigl(x, \pi(\kappa'(x))\bigr) \in X$. From the assumption that $\bigl\{\, x \mid x \in Q \wedge \kappa(x) \in \{m_1, \dots, m_l\}\bigr\}$ is complete for NL we can conclude that there is a reduction $r\colon (\Sigma')^* \times \Pi^* \to \Sigma^*$ from X to this language that is computable in logarithmic space. Based on r, we define a new reduction $s\colon (\Sigma')^* \to \Sigma^*$ from (Q', κ') to (Q, κ) by

$$s(x) = \begin{cases} r(x) & \text{if } \kappa(r(x)) \in \{m_1, \dots, m_l\}; \\ x_0 & \text{otherwise} \end{cases}$$

where x_0 is a fixed element with $x_0 \in (\Sigma^* - Q)$, which exists because we assumed Q to be nontrivial. Since we have $\kappa(s(x)) \in \{m_1, \dots, m_l, \kappa(x_0)\}$, the function s is indeed a parameterized reduction, and, moreover, it is a para-L-computable reduction because checking whether we have $\kappa(r(x)) \in \{m_1, \dots, m_l\}$ can easily be done in parameterized logarithmic space due to the fact that r is computable in logarithmic space. $\square$

Now, consider the language

$$\{x \mid x \in \textsc{Large-Distance} \wedge \kappa(x) = 0\}.$$

This clearly is the NL-complete language REACH. Hence, with the lemma above and the fact that p-LARGE-DISTANCE $\in$ para-NL, we have proven the theorem. □

While p-DISTANCE-AT-LEAST and p-UNDIRECTED-DISTANCE-AT-LEAST are both complete for the same class, namely $\text{para}_{\beta\forall}$-L, this behaviour does not carry over to p-LARGE-DISTANCE and p-UNDIRECTED-LARGE-DISTANCE, the undirected version of p-LARGE-DISTANCE:

▷ *Theorem 53.* p-UNDIRECTED-LARGE-DISTANCE is complete for $\text{para}_{\beta\forall}$-L with respect to para-L-reductions.

Proof. Instead of "directly" proving that p-UNDIRECTED-LARGE-DISTANCE is complete for $\text{para}_{\beta\forall}$-L, we just show that its complement is complete for para_{β}-L. From the definition of p-UNDIRECTED-LARGE-DISTANCE we get that the complement is p-UNDIRECTED-ONLY-SHORT-DISTANCE

▷ *Problem 54 (Parameterized Undirected Only Short Distance).*

Instance: An undirected graph G, two vertices s and t of G, and a natural number k.

Parameter: k.

Question: Is there a path from s to t in G of length at most k or no path from s to t at all?

We can decide this problem using a para_{β}-L-machine by first checking whether there is a path from s to t of arbitrary length using the famous theorem of Reingold (2008). If there is no path, we accept. Otherwise we have to test whether there is a short path, which we do in the same manner as before when we decided p-DISTANCE: We nondeterministically guess a path from s to t and accept if we find such a path.

For hardness, we reduce from p-DISTANCE. Let (G, s, t, k) be an instance of p-DISTANCE. We have to face three possible situations:

1. There is a path from s to t of length at most k in G.
2. There are paths from s to t in G, but any of them has length at least $k+1$.
3. There is no path from s to t in G.

Case 1 and case 2 are no problem because any machine deciding the problem p-UNDIRECTED-ONLY-SHORT-DISTANCE gives us the correct answer for p-DISTANCE on input of a graph matching any of these cases. Case 3 is a problem: While a machine deciding p-UNDIRECTED-ONLY-SHORT-DISTANCE accepts an instance of this case, any machine deciding p-DISTANCE does not.

However, we can solve this problem by adding a fresh vertices and edges to the graph that form path of length $k + 1$ from s to t. Now, if we had a short path before, we still have a short path in the new graph. If there were only long paths, we still have only long paths. However, if there has been no path from s to t before, there is such a path now. Hence, on input of such a graph, a machine deciding p-UNDIRECTED-ONLY-SHORT-DISTANCE now rejects this input. Morever, adding a path of length $k + 1$ can easily be done using para-L-reductions. Hence, we get the theorem. □

Up to now we discussed several variants of distance problem, questioning whether the distance between two vertices is at most a certain value or whether it is at least a certain value. The single case that remains is the natural question what happens if we want both, namely that the distance between two vertices is *exactly* a certain value. Let us formalize this as the problem p-EXACT-DISTANCE:

▷ *Problem 55 (Parameterized Exact Distance).*

Instance: A directed graph G, two vertices s and t of G, and a natural number k.

Parameter: k.

Question: Is there a path from s to t in G of length k but no shorter path?

Such "exact optimization problems" have already been studied in the setting of classical computational complexity, most notably is the work of Papadimitriou and Yannakakis (1984): They introduced the class DP, which is defined by $\mathrm{DP} = \{ L_1 \cap L_2 \mid L_1 \in \mathrm{NP} \wedge L_2 \in \text{co-NP}\}$ and that captures the complexity of many important exact optimization problems like, for example, EXACT-TSP, the traveling salesperson problem where we ask whether the shortest tour has precisely a certain length. The main idea of DP is that if we ask for a solution to a problem that has *exactly* a given value c, we might equivalently ask for a solution that has a value *less or equal* to c and, at the same time, a value *greater or equal* to c. This translates into complexity classes by using the intersection of languages in NP and co-NP: The NP-part ensures that there is at least one solution of a given value, the co-NP-part ensures that there are no solutions with a value below the given one. Let us now transfer this into our world of parameterized space complexity.

Equivalently to the definition above, we can define p-Exact-Distance by

$$\begin{aligned}\text{p-Exact-Distance} &= (Q_1 \cap Q_2, \kappa) \quad \text{with} \\ (Q_1, \kappa) &= \text{p-Distance} \quad \text{and} \\ (Q_2, \kappa) &= \text{p-Distance-At-Least}\end{aligned}$$

which, together with the concept underlying DP, directly leads us to the definition of a new class capturing exact optimization problems in the world of parameterized space complexity:

▷ *Definition 56 (para$_{D\beta}$-L).* The class para$_{D\beta}$-L is defined by

$$\text{para}_{D\beta}\text{-L} = \big\{\, (Q_1 \cap Q_2, \kappa) \mid (Q_1, \kappa) \in \text{para}_{\beta}\text{-L} \wedge (Q_2, \kappa) \in \text{para}_{\beta\forall}\text{-L}\big\}.$$

From the definition we immediately get p-Exact-Distance $\in$ para$_{D\beta}$-L, but we can even show that para$_{D\beta}$-L exactly captures the complexity of this problem:

▷ *Theorem 57.* p-Exact-Distance is complete for para$_{D\beta}$-L with respect to para-L-reductions.

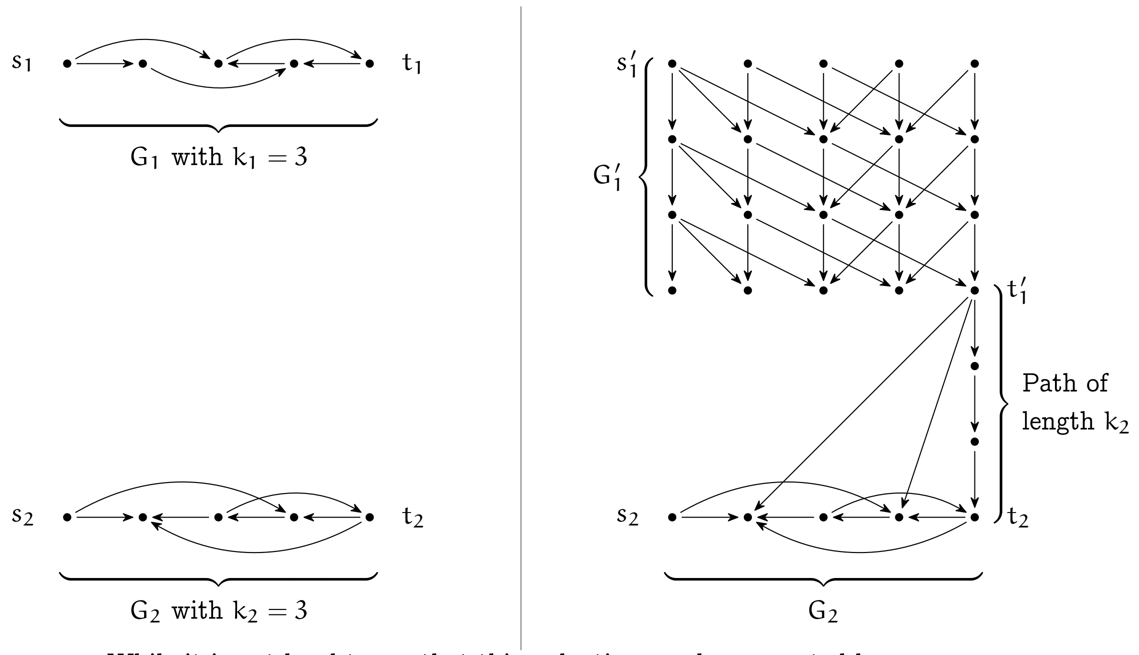
Proof. We have already seen above that p-Exact-Distance lies in para$_{D\beta}$-L, thus the only thing that remains to show is the hardness of p-Exact-Distance. Hence, let (Q, κ) be a problem in para$_{D\beta}$-L. Then we know that there are two problems (Q_1, κ) and (Q_2, κ) such that $Q = Q_1 \cap Q_2$. Moreover, since p-Distance and p-Distance-At-Least are complete for para$_\beta$-L and para$_{\beta\forall}$-L, respectively, there exists a reduction r_1 from (Q_1, κ) to p-Distance and a reduction r_2 from (Q_2, κ) to p-Distance-At-Least. We will now combine these two reductions to form a reduction from (Q, κ) to p-Exact-Distance.

A first idea may be to, for an instance x of (Q, κ), compute $r_1(x)$ and $r_2(x)$, i. e., instances (G_1, s_1, t_1, k_1) and (G_2, s_2, t_2, k_2) of p-Distance and p-Distance-At-Least, respectively, and to combine them to an instance $(G, s_1, t_2, k_1 + k_2)$ where G is made up from G_1 and G_2 by identifying t_1 and s_2. This, however, does not work for various reasons:

- We cannot conclude that if (G_1, s_1, t_1, k_1) and (G_2, s_2, t_2, k_2) are positive instances of p-Distance and p-Distance-At-Least, respectively, that there is a path in G from s_1 to t_2 of length exactly $k_1 + k_2$ because k_1 and k_2 are only upper and lower bounds, respectively.
- Since (G_2, s_2, t_2, k_2) is also a positive instance of p-Distance-At-Least if there is no path from s_2 to t_2 at all, there might also be no path from s_1 to t_2 in G at all, even though (G_1, s_1, t_1, k_1) and (G_2, s_2, t_2, k_2) are positive instances of p-Distance and p-Distance-At-Least.

To avoid these problems, we roughly proceed as sketched above, but we have to do a little more work: First, we apply a layering trick similar to the one in the proof of Theorem 48 but now with careful attention to the edge direction to (G_1, s_1, t_1, k_1) which yields (G'_1, s'_1, t'_1, k_1), a graph that contains a path from s'_1 to t'_1 of length exactly k_1 if, and only if, there is a path from s_1 to t_1 in G_1 of length at most k_1. Moreover, this trick ensures that there is no shorter path from s'_1 to t'_1. Now, we create the new graph $(G, s'_1, t_2, k_1 + k_2)$ from (G'_1, s'_1, t'_1, k_1) and (G_2, s_2, t_2, k_2) by first taking the disjoint union of the vertices and edges of G'_1 and G_2, then inserting edges from t'_1 to every vertex that is reachable from s_2 via a direct edge, and finally adding fresh vertices and edges that form a path of length k_2 from t'_1 to t_2. Altogether, from this construction we obtain a graph G that has a path from s'_1 to t_2 of length $k_1 + k_2$ and no shorter such path if, and only if, $x \in (Q, \kappa)$.

The following graphic shows an example of this construction. The left hand side shows two instances (G_1, s_1, t_1, k_1) and (G_2, s_2, t_2, k_2) of p-DISTANCE and p-DISTANCE-AT-LEAST, respectively. The right hand side shows the result of the reduction that combines these instances to $(G, s'_1, t_2, k_1 + k_2)$.



While it is not hard to see that this reduction can be computed by a para-L machine if r_1 and r_2 can be computed by para-L machines, the correctness of the reduction may not be obvious. Hence, let us briefly discuss the possible cases:

1. Suppose that we have $x \in Q$. This means that $r_1(x)$ is a positive instance of p-DISTANCE and $r_2(x)$ is a positive instance of p-DISTANCE-AT-LEAST. Let $r_1(x) = (G_1, s_1, t_1, k_1)$ and $r_2(x) = (G_2, s_2, t_2, k_2)$. We then know that there is a path of length at most k_1 from s_1 to t_1 in G_1 and no path of length k_2 or shorter from s_2 to t_2 in G_2. Now let (G_1', s_1', t_1', k_1) and $(G, s_1', t_2, k_1 + k_2)$ be constructed like discussed above. Then, in G, there is a path from s_1' to t_1' of length exactly k_1 and no shorter path between these two vertices. A shortest path from t_1' to t_2 is the newly constructed path of length k_2. Hence, the shortest path from s_1' to t_2 in G has length $k_1 + k_2$, thus we have that $(G, s_1', t_2, k_1 + k_2)$ is a positive instance of p-EXACT-DISTANCE.
2. Suppose now that $x \notin Q$. Hence, we have that $r_1(x)$ is a negative instance of p-DISTANCE, $r_2(x)$ is a negative instance of p-DISTANCE-AT-LEAST, or both $r_1(x)$ and $r_2(x)$ are negative instances. If $r_1(x)$ is a negative instance of p-DISTANCE, then there is no path of length at most k_1 from s_1 to t_1 in G_1 and, hence, no path from s_1' to t_1' in G at all, which implies that $(G, s_1', t_2, k_1 + k_2)$ is a negative instance of p-EXACT-DISTANCE. On the other hand, if $r_2(x)$ is a negative instance of p-DISTANCE-AT-LEAST, then there is path from s_2 to t_2 in G_2 of length less than k_2. This means that, depending on whether $r_1(x)$ is a positive or negative instance of p-DISTANCE, there is either a path of length shorter than $k_1 + k_2$ in G from s_1' to t_2 or no path between these vertices. Thus, $(G, s_1', t_2, k_1 + k_2)$ is then a negative instance of p-EXACT-DISTANCE.

We can conclude that p-EXACT-DISTANCE is complete for $\text{para}_{D\beta}$-L. □

The results that we have studied so far show that distance problems are strongly related to parameterized space complexity. Let us now, for the rest of this section, do a little twist: Instead of searching short paths, let us search long paths! Formally, let us study the *parameterized longest path problem*:

▷ *Problem 58 (Parameterized Longest Path).*

Instance: A directed graph G and a natural number k.

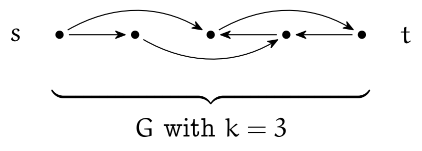
Parameter: k.

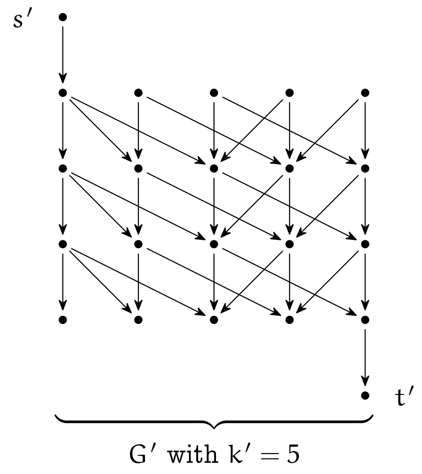
Question: Is there a simple path of length k in G, i. e., a path that contains no vertex twice?

The unparameterized LONGEST-PATH is well known to be NP-complete, see Garey and Johnson (1979) for details, and has been studied mostly from the perspective of approximation algorithms. After Bodlaender (1993) showed that p-LONGEST-PATH lies in para-P, the focus from the parameterized point of view laid on improving the dependency of the runtime on the parameter for different graph classes. However, Chen and Müller (2015) studied p-LONGEST-PATH from the perspective of conjunctive queries as well as embedding and homomorphism problems, and they showed that p-LONGEST-PATH is complete for para_β-L. In their proof, they make use of a useful technique that allows a para_β-L machine to make sure that her nondeterministic guesses are "injective", i. e., they show how a para_β-L machine can make sure that no two of the $O(f_k)$ blocks of $O\big(\log(n)\big)$ nondeterministic bits are equal. Since their proof is based on embedding problems and, moreover, uses a special machine model named *jump machines*, we will reprove their result now, but with our focus on the mentioned technique and in the context of the distance problems discussed so far.

▷ *Theorem 59.* p-LONGEST-PATH is complete for para_β-L with respect to para-L-reductions.

Proof. Let us start with the proof of hardness because this is straight-forward. We reduce from p-DISTANCE: Similar to the proof of Theorem 57, we first transform the graph G of the given instance (G, s, t, k) into a layered graph with $k + 1$ layers such that there is path from the new start vertex to the new target vertex if, and only if, there is a path of length at most k from s to t in G, and such that every path from the new start vertex to the new target vertex has length exactly k. Then we add two additional vertices to the graph. We insert an edge from the first fresh node to the start vertex, and a second edge from the target vertex to the other fresh node. Finally, we set the new parameter to $k + 2$. The follow graphic shows an example of the construction: On the left hand side the instance for p-DISTANCE is depicted, on the right hand side the result of the reduction on this instance.





Our claim is that this new graph contains a simple path of length $k + 2$ if, and only if, the distance of t from s in G is at most k. Hence, assume that there is a path from s to t in G of length at most k. Then there is a path $v_s^1, \dots, v_t^{k+1}$ that is basically using the edges of the path in G, but on every step this path advances to the next layer. Together with the two additional vertices s' and t' and their accompanying edges (s', v_s^1) and (v_t^{k+1}, t'), this makes up a path of length $k + 2$. On the other hand, if there is a simple path of length $k + 2$ in G', then this path has to start at s' and end at t' because, due to the layering, this is the only possibility of a path of length $k + 2$ in G'. This implies that there is a path from v_s^1 to v_t^{k+1} in G'. Since the layering only reflects the edges in G (together with an imaginary loop (t, t)), we can conclude that there is also a path from s to t in G. Finally, it is not hard to see that this reduction can easily be computed using para-AC^0 circuits.

Let us now discuss why p-LONGEST-PATH lies in para_β-L. A first and admittedly natural idea for an algorithm deciding p-LONGEST-PATH is to use the nondeterminism of the para_β-L-machine to simply guess the vertices of a path of length k. However, we run into a problem here: Because we cannot store the already guessed vertices, we might accidentally follow a loop in the graph without noticing, but what we really want is a simple path. Hence, we somehow have to keep track of the already visited vertices, but we may only use parameterized logarithmic space. The key idea to solve this problem is to use *hashing*.

To ensure that we do not visit a vertex twice, we maintain a list of small hash values for every vertex we already visited. If we never have a hash collision during the process of nondeterministically guessing a path, we found a simple path. However, we now have the problem that two different vertices on a path may accidentally lead to a collision and that we have to compute and store the hash values space-efficiently. To solve these problems, we make use of certain *families of* k*-perfect hash functions*:

▷ *Definition 60.* For two sets M and N, a k*-perfect family of hash functions* is a family H of functions mapping from M to N such that for every set K with $K \subseteq M$ and $|K| = k$ there is a function h in H such that for every x and y in K with $x \neq y$ we have $h(x) \neq h(y)$, i. e., h is injective on K.

Hence, we now only have to find such families of hash functions that are computable in parameterized logarithmic space and that allow us to store k hash values within our limited space. Appropriate functions that meet these requirements are known, see for instance Flum and Grohe (2006): They show that the set

$$\Big\{ h_{p,a} \mid p \text{ is prime and } p < k^2 \cdot \log(n) \text{ and } a \in \{1, \dots, p-1\} \Big\} \quad \text{with}$$
$$h_{p,a}(x) = (a \cdot x \bmod p) \bmod k^2$$

is a family of k-perfect hash functions mapping from $\{1, \dots, n\}$ to $\{0 \dots, k^2 - 1\}$, and, indeed, we can use these functions within our limited space: storing the prime number p and the factor a requires space at most $O\big(\log(k) + \log(\log(n))\big)$ and storing the at most k hash values requires space at most $O\big(k \cdot \log(k)\big)$. Moreover, computing the hash values can easily be done within our space bounds. The last question remaining is how to choose the correct function of these families on input of G and k? One way is to simply iterate over all these functions, another possibility is to use the available nondeterminism to guess the function, which requires $O\big(\log(k) + \log(\log(n))\big)$ and is therefore doable within the available bounded nondeterminism of $O\big(f_k \cdot \log(n)\big)$ bits. □

If we turn to the undirected version of the parameterized longest path problem, the situation becomes less clear. It is tempting to conclude that, in analogy to the directed and the undirected versions of the distance problem, the undirected longest path problem is also complete for para_β-L, but this does not seem to be true:

▷ *Theorem 61.* p-UNDIRECTED-LONGEST-PATH $\in$ para-L

Proof. For a given instance (G, k) we iterate over every pair of vertices of G, and for each such pair (s, t) we iterate over the functions of a $(k+1)$-perfect family of hash functions that map the vertices of G to the set $\{1, \dots, k +$

$1\}$. For each such function h we consider every function $p\colon \{1,\dots,k+1\} \to \{1,\dots,k+1\}$ of the symmetric group S_{k+1} and construct a new graph $G_{h,p}$ from G by removing all edges from G that do not connect vertices of "adjacent hash values", i. e., for every remaining edge $\{u,v\}$ in $G_{h,p}$ we have $|p(h(u)) - p(h(v))| = 1$. In $G_{h,p}$ we then test whether $p(h(s)) = 1$, $p(h(t)) = k+1$, and whether we can reach t from s using the Theorem of Reingold (2008). If we find a pair (s,t) of vertices, a hash function h, and a permutation p such that there is a path from s to t in $G_{h,p}$, we know that this path has to pass at least $k+1$ vertices, thus we have found a path of length k in G. In this case, we accept, otherwise we reject, because if there is a path $(v_1,\dots,v_{k+1})$ of length k with $s = v_1$ and $t = v_{k+1}$, then there is also a hash function h such that $h(v_i) \neq h(v_j)$ if, and only if, $i \neq j$, and a permutation p for h such that $p(h(v_i)) = i$. In other words: By iterating over all pairs of vertices, all hash functions and all permutations, we consider every possibility of paths of length $k+1$.

In the proof of Theorem 59 we have already seen that iterating over a $(k+1)$-perfect family of hash functions can easily be done by a para-L machine. Now note that both additionally testing for reachability and iterating over all functions of S_{k+1} can easily be done within the space bounds of para-L. □

Whether p-UNDIRECTED-LONGEST-PATH is also complete for para-L is a part of current research, but recent results from Bannach et al. (2015) suggest that this is not the case.

Let us conclude this chapter. Figure 3.1 summarizes its core results: We have seen that the combination of parameterized space and circuit complexity with bounded nondeterminism is a fruitful basis for the exact classification of many natural and important computational problems that could not be classified so far using parameterized time classes like colored reachability problems or the associative generability problem. These problems share a common property: they are union problems. Many parameterized problems exhibit this property, and we have seen that this property is linked to classes of read-again bounded nondeterminism by the union lemma. Moreover, classes like para_W-NC^1, para_W-L, and para_W-NL provide additional upper and lower bounds (under reasonable assumptions) for both the directed and undirected feedback vertex set problem, thus extending the results from the previous chapter where we studied classes of parameterized space and circuits without nondeterminism. Another important observation we have made in this chapter is that space and circuit classes of bounded nondeterminism are linked to the Weft-Hierarchy: Showing that a problem is hard for para_W-AC^0 suffices as a proof for the hard-

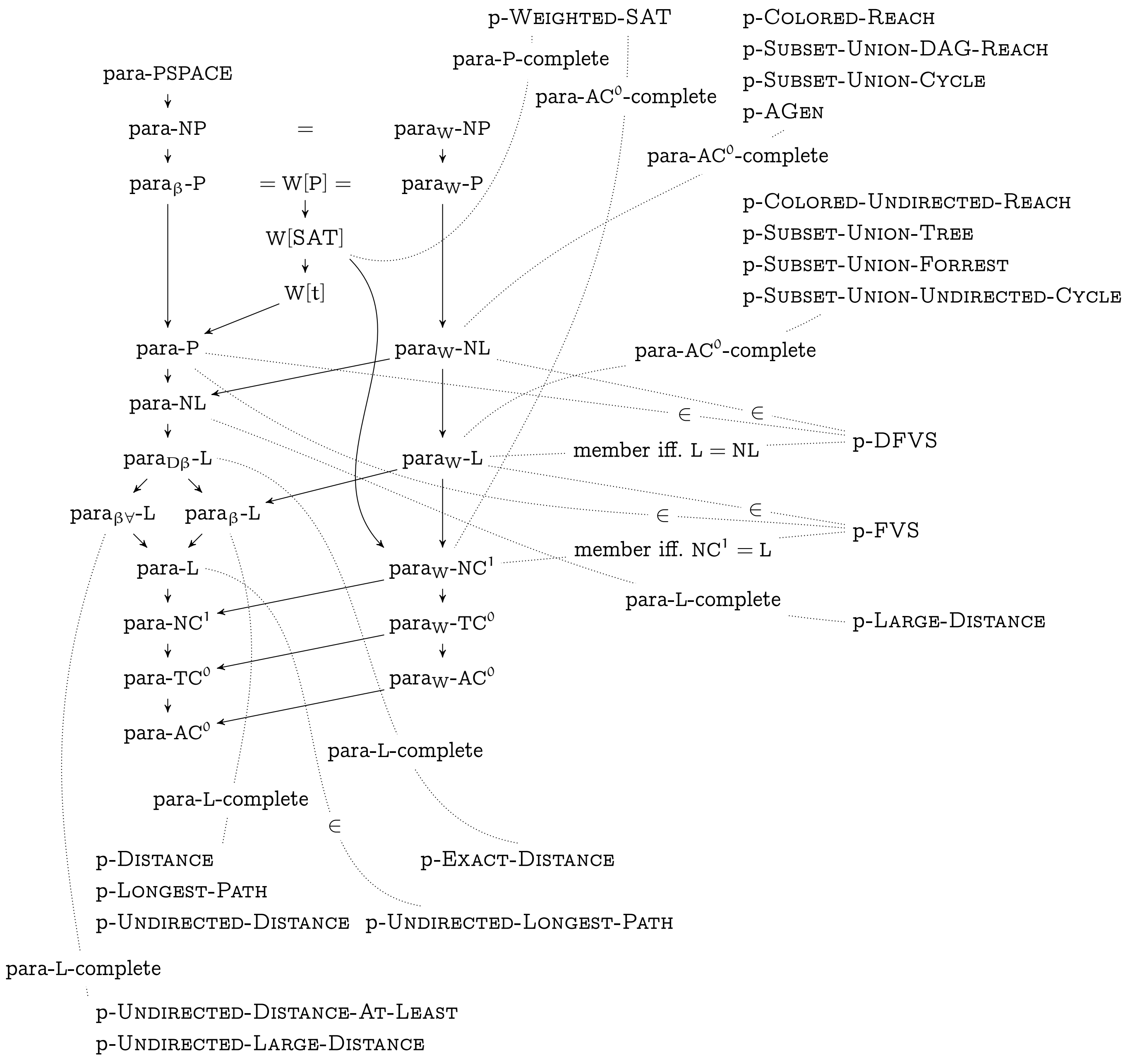


Figure 3.1: Diagram of classes of bounded nondeterminism together with some of their surrounding classes and inclusions where $A \rightarrow B$ denotes the inclusion $A \supseteq B$. Moreover, the relations between the problems and classes discussed in this chapter are illustrated.

ness of the problem for W[t], showing that a problem is hard or complete for para_W-NC^1 suffices as a proof for the hardness or completeness for W[SAT], respectively. Concerning read-once bounded nondeterminism, we have seen that many natural distance problems are complete for the classes para_β-L, $\text{para}_{\beta\forall}$-L, and $\text{para}_{D\beta}$-L.

While Flum and Grohe (2006) may be right when judging that classes like para-NP (and thus also para-L and para-NL) as uninteresting, their statement clearly does not hold when we switch to classes of bounded nondeterminism. Bounded nondeterminism proves to be the right way of connecting the concept of nondeterminism with parameterized complexity theory.

However, parameterized space and circuit complexity theory alone, even when extended using bounded nondeterminism, is not a panacea. It is often very useful to combine time and and space complexity in order to get deep insights into the computational complexity of problems with high practical relevance, as we will see in the next chapter.

4. Simultaneous Time-Space Classes

Time and space are the two fundamental resources when talking about the computational complexity of algorithms. Starting with the works of Hartmanis and Stearns (1965) and Stearns et al. (1965), the study of the time and space required to solve a problem is the central momentum of computational complexity. One of the most important and at the same time hardest questions to answer is: How are time and space related to each other? An incomplete answer to this question is the well-known inclusion chain

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP},$$

which shows that time and space seem to be interleaved.

While this chain presents the simple and beautiful essence of the classical theory of computational complexity, it has its disadvantages: It does not allow the study of computations with simultaneous time and space bounds. Bounding either time or space always limits the other resource. For example, it does not make sense to study computations that require a polynomial amount of time and a logarithmic amount of space, because this is already captured by limiting the allowed use of space to a logarithmic amount alone. This changes if we switch to the setting of parameterized complexity.

In parameterized complexity theory we deal with several kinds of complexity classes: para-classes, X-classes, and classes of bounded nondeterminism like para_{W}-classes and para_{β}-classes. In Chapter 2 we have already seen that many of these classes presumably lie orthogonal to each other and are therefore incomparable. While this makes the world of parameterized complexity appear somehow less "easy" in comparison to classical computational complexity, it is in fact an advantage: (Presumably) orthogonal classes like para-P and XL allow us to study their intersection para-P $\cap$ XL or even the class of problems that are decidable by a single machine simultaneously obeying the resource bounds imposed by classes like para-P and XL, i. e., in time $O(f_k \cdot n^c)$ and space $O\big(f_k \cdot \log(n) + f_k\big)$. This section is devoted to the study of such problems.

In Section 4.1 we will introduce the most important classes with simultaneous time-space bounds and study their structural properties. This includes the investigation of some admittedly artificial complete problems for these classes. However, we require these for our results in Section 4.2 where we will then use the previously introduced classes and problems to study natural problems like the longest common subsequence problem or the feedback vertex set problem.

4.1. Classes and Structural Properties

Before we start with our study of problems, we have to prepare our working gear, which consists of a reasonable way to denote simultaneous time-space classes and classes to deal with. Let us start with the notation:

▷ *Definition 62 (Simultaneous Time-Space Classes).* Let t and s be a time bound and a space bound, respectively, that both depend on a parameter k and the input length n. Then,

$$D[t, s] \qquad \text{and} \qquad N[t, s]$$

denote the class of parameterized problems that are solvable by a deterministic and nondeterministic Turing machine, respectively, within time $t(\kappa(x), n)$ and space $s(\kappa(x), n)$.

If we now plug in our previously discussed time and space bounds for para-classes and X-classes, we get a whole bunch of complexity classes, many of them unreasonable, but four of them are of interest:

$$D[f_k \cdot n^c, f_k \cdot \log(n)], \qquad N[f_k \cdot n^c, f_k \cdot \log(n)],$$
$$D[n^{f_k} + f_k, f_k \cdot n^c], \qquad N[n^{f_k} + f_k, f_k \cdot n^c].$$

In a more intuitive way, we can denote these classes by

para-P/XL, para-NP/XNL,
XP/para-PSPACE, XNP/para-PSPACE,

where, for example, para-P/XL denotes the class of problems decidable by a para-P-machine that may use at most the space of an XL-machine. Figure 2.1 on page 33 shows where these classes are related to the previously discussed classes.

The intuition behind para-P/XL is that it is a space-efficient variant of para-P and, thus, a subclass of para-P allowing us to study the problems within para-P. para-NP/XNL then is the nondeterministic variant of para-P/XL. Similarly, the classes XP/para-PSPACE and XNP/para-PSPACE can be seen as space-efficient variants of XP and XNP. Alternatively, if we focus on space as our central resource, the classes above can be seen as time-efficient space classes. However, even though we argue that space and circuit complexity is extremely important, we will stick to time as the central resource of interest in this chapter.

From the definition of these classes it is immediately clear that they are subclasses of the underlying classes that define them, i. e., we have that, for instance, para-P/XL is a subclass of both para-P and XL. Moreover, we have the following relations to other parameterized complexity classes we previously studied:

$$\begin{aligned}\text{para}_{\text{W}}\text{-NP} &\subseteq \text{XNP/para-PSPACE},\\ \text{para}_{\text{W}}\text{-P} &\subseteq \text{XP/para-PSPACE},\\ \text{para}_{\text{W}}\text{-NL} &\subseteq \text{para-NP/XNL},\\ \text{para-L} &\subseteq \text{para-P/XL}.\end{aligned}$$

The first inclusion follows from the trivial fact that $\text{para}_{\text{W}}\text{-NP} = \text{para-NP}$ and that a para-NP-machine always obeys the simultaneous time-space bounds imposed by XNP/para-PSPACE. For the second inclusion note that we can iterate over the possible contents of the choice tape of a para_{W}-P-machine and simulate the machine on each of them within the bounds of XP/para-PSPACE. For the third inclusion note that we can nondeterministically guess the certificate of a para_{W}-NL-machine and simulate the machine afterwards within the bounds of para-NP/XNL. To see the last inclusion just note that a para-L clearly obeys the time-space bounds of a para-P/XL machine. For an overview of how the above classes relate to the previously discussed classes see Figure 4.1 on page 107

Before we start investigating natural complete problems for the classes introduced above, let us first study a bunch of artificial complete problems that we will use as a starting point. Typically, machine simulation problems are a good basis for showing completeness of problems, and this is also the case here. Thus, we consider the space-bounded acceptance problem, i. e., deciding whether a given machine accepts the empty string using at most a certain number of tape cells:

▷ *Problem 63 (Parameterized Deterministic Space-Bounded Acceptance).*

Instance: The code of a single-tape Turing machine M together with a natural number s in unary.

Parameter: s.

Question: Does M halt in an accepting state on an initially empty tape using at most s tape cells?

Let us denote this problem with p-DSBA and the variant of this problem where we consider nondeterministic machines by p-NSBA. Moreover, let us define two variants of p-DSBA and p-NSBA with an additional time constraint, namely p-TIMED-DSBA and p-TIMED-NSBA for deterministic and nondeterministic Turing machines, respectively:

▷ *Problem 64 (Parameterized Timed Deterministic Space-Bounded Acceptance).*

Instance: The code of a single-tape Turing machine M together with two natural numbers s and t in unary.

Parameter: s.

Question: Does M halt in an accepting state on an initially empty tape using at most s tape cells and making at most t steps?

▷ *Theorem 65.*

1. p-DSBA is para-AC^0-complete for XL.
2. p-NSBA is para-AC^0-complete for XNL.
3. p-TIMED-DSBA is para-AC^0-complete for para-P/XL.
4. p-TIMED-NSBA is para-AC^0-complete for para-NP/XNL.

Proof. Let us start with containment and item 1. To see that p-DSBA lies in XL, let the input be a tuple $(M, 1^s)$. To check whether M accepts the empty string using at most s tape cells, we simply simulate M. This requires s blocks of $O(\log(n))$ space, each encoding a tape cell of M, i. e., a symbol of M's alphabet. To show item 2, i. e., the nondeterministic case, we can proceed similarly, but we have to additionally use nondeterminism to simulate the nondeterministic behaviour of the input machine. For the timed variants, i. e., item 3 and item 4, we argue in a same way, but now we get a tuple $(M, 1^t, 1^s)$ as input and also have to keep track of the the number of steps made by the machine, which can easily be done in polynomial time. Hence, the overall runtime does not exceed the para-P limit.

To prove hardness, we again start with item 1. Let (Q, κ) be a problem in XL that is decided by a machine M in space $s_M(k, n)$ with $s_M \in O\big(f_k \cdot \log(n)\big)$. The main idea of the reduction is to compute on input x a tuple $(M', 1^s)$ such that M' simulates M on x and s is a space bound sufficient for this simulation but that exclusively depends on the parameter $\kappa(x)$, because in parameterized reductions the new parameter may only depend on the the old parameter. To achieve this, we proceed in two steps. First, we hard-wire x to M yielding M_x. For this, we augment M's set of states to $O(|x|)$ copies of its old state set and connect them in such a way that the input x and the movement of the head on x is represented by these states. Using this technique, we get rid of the requirement of passing x to M separately. In a second step, we have to make sure that our new machine only requires f_k tape cells for some function f. To achieve this, we apply a classical compression trick: We augment M_x's alphabet such that every symbol encodes a block of at most $O\big(\log(|x|)\big)$ many symbols together with one of the possible head position within this block, yielding M'. This enlarges the old alphabet Σ to $O\big(|\Sigma^{\log(|x|)} \times \{0, 1, \dots, \log(|x|)\}|\big)$, where the first component encodes a block of $O\big(\log(|x|)\big)$ symbols and second component encodes the head position, i. e., 0 means that the head is not in this section, 1 means that it points to the first symbol, 2 to the second symbol, and so on. Finally, we have to adjust the state transitions of M' such that it correctly simulates the computation of M_x. While this may be tedious, it is not complicated. The new instance is then the code of M' together with the parameter f_k. In the nondeterministic case of item 2 we proceed in exactly the same way.

To show hardness of the timed variants, i. e., items 3 and 4, we proceed in a similar way, but now we start with a problem (Q, κ) that is solvable within space $s_M(k, n)$ and time $t_M(k, n)$, and we have to additionally construct a unary encoded time bound. To do so, we simply set the time bound to $t_M(\kappa(x), |x|)$. Note, that the time bound is not a parameter and, therefore, may depend on the input length. $\square$

Now that we have our first problems that are complete for classes of simultaneous time-space bounds, we can start investigating natural complete problems for these classes in the next section.

4.2. Natural Problems for Time-Space Classes

Despite the overwhelming success of parameterized complexity theory, numerous parameterized problems could not be classified satisfactorily for decades.

The textbooks of Downey and Fellows (1997) and Flum and Grohe (2006) contain long lists of problems that are annotated with "member of" and "hard for" but that could not be shown to be complete for any complexity class. One of these problems is based on the classical problem of computing longest common subsequences that we already mentioned in Section 2.1 when we discussed the concept of parameterized problems. Maier (1978) studied LCS in the context of classical computational complexity and showed that LCS is NP-complete. In the parameterized world we have to deal with several versions of this problem, depending on the chosen parameter. Bodlaender et al. (1995) showed that $p_{|S|}$-LCS is hard for W[t] for all t, p_{ι}-LCS lies in W[P] and is hard for W[2], and $p_{|S|,\iota}$-LCS is complete for W[1]. Moreover, they showed that if the underlying problem is restricted to a fixed alphabet, then p_{ι}-LCS and $p_{|S|,\iota}$-LCS are complete for W[1], while they conjectured that $p_{|S|}$-LCS remains W[t]-hard. Pietrzak (2003) proved that this conjecture is right. Guillemot (2011) refined this analysis by showing that $p_{|S|}$-LCS is equivalent to p-TIMED-NSBA with respect to para-P-reductions, and, consequently, introduced the parameterized class WNL as the closure of p-TIMED-NSBA under para-P-reductions in order to capture the complexity of $p_{|S|}$-LCS. Apart from the fact that this result underlined that the $p_{|S|}$-LCS does not fit into the well-established complexity classes and seems to require a new complexity class, Guillemot's result has two blemishes: First, since WNL is based on a problem, it lacks a machine-based interpretation. This is somehow fixed by our result above that p-TIMED-NSBA is complete for para-NP/XNL. However, this reveals the second problem: para-NP/XNL is presumably not closed with respect to para-P-reductions that were used to show the equivalence of $p_{|S|}$-LCS and p-TIMED-NSBA by Guillemot (2011). We fix this in this section by showing that $p_{|S|}$-LCS is complete for para-NP/XNL with respect to para-AC^0-reductions, i. e., a very weak reduction. For this we make us of a reduction chain that also shows the completeness of several other reasonable problems for this class. In toto, this result underlines the importance of simultaneous time-space classes for the classification of natural problems like $p_{|S|}$-LCS.

The second problem that we study with respect to its simultaneous time-space complexity is the problem of computing feedback vertex sets. In Section 2.2 we already discussed that p-DFVS and p-FVS both lie in para-P, and, moreover, in Theorem 12 we saw that they are separated within para-P by parameterized space and circuit classes. In this section, we will go a step further by additionally showing an upper bound for p-FVS and an additional lower bound for p-DFVS using simultaneous time-space classes.

Let us begin our journey towards $p_{|S|}$-LCS by studying a first intermediate problem that is based on the computational model of *cellular automatons*, see Wolfram (2002) for the wide range of their applications. The central parts of a cellular automaton are its possibly infinitely many *cells*, which are arranged in a certain manner such that every cell has a well-defined neighborhood of other cells. Typical arrangements are the euclidian plane such that every cell has exactly nine neighbors, or the line such that every cell has exactly two neighbors. These cells perform a synchronized computation: Initially, every cell is in one of a finite set Q of states. A state transition function that defines for each cell its successor state depending on its own state and the states of its neighbors is then applied repeatedly and simultaneously to the cells. There are numerous applications of cellular automatons, most notably is probably *Conway's Game of Life*, presented by Gardner (1970).

The cellular automaton has proven to be a universal computational model, i. e., it is equivalent to the Turing machine regarding its expressive power. Hence, many problems concerning cellular automatons are undecidable, making it reasonable to modify the model in order to obtain decidability. We consider the natural variant where we restrict the number of cells of the automaton to be finite. Consequently, the transition function has to cover the corner cases where the cells do not have a "full" neighborhood. Moreover, we consider automatons that have a set of accepting states, and we say that a automaton accepts its initial configuration, which can be seen as its input, if one of its cells arrives at one of the accepting states during the automaton's computation. This yields the natural problem p-DCA:

▷ *Problem 66 (Parameterized Deterministic Cellular Acceptance).*

Instance: A deterministic cellular automaton with k cells and an initial configuraton.

Parameter: k.

Question: Does the automaton accept?

Naturally, we can define this problem for transition relations, i. e., nondeterministic automatons, and we can even define timed versions where we want to know whether the automaton accepts within a certain time (encoded as a unary string along with the input). Let us denote these problems by p-NCA, p-TIMED-DCA, and p-TIMED-NCA, respectively.

▷ *Theorem 67.*

1. p-DCA is para-AC^0-complete for XL.

2. p-NCA is para-AC^0-complete for XNL.
3. p-TIMED-DCA is para-AC^0-complete for para-P/XL.
4. p-TIMED-NCA is para-AC^0-complete for para-NP/XNL.

Proof. We start with containment. In order to simulate the computation of a given automaton, we have to store the current configuration of the automaton's cells, which we can do in $O\big(k \cdot \log(n)\big)$ space. Hence, we have p-DCA $\in$ XL and p-NCA $\in$ XNL. Moreover, we have p-TIMED-DCA $\in$ para-P/XL and p-TIMED-NCA $\in$ para-NP/XNL because the simulation of the automatons for a number of steps that is given in unary can easily be done in time polynomial in the number of cells and the input length.

For hardness, let us start with case 1, where we reduce from p-DSBA. We are thus given the encoding of a Turing machine M together with a natural number s in unary that is our parameter, and we must map this to a cellular automaton C with at most $f(s)$ cells for some function f that accepts if, and only if, M accepts the empty string using at most s tape cells. The natural idea here is to simulate M's computation by C and use the cells of C to store the contents of M's tape cells and the current head position. We thus let C have s cells that are arranged in a sequence $c_1, \dots, c_s$. Cell c_i represents the content of the i-th tape cell of M together with the information whether M's head currently points to the i-th cell or not, and, if so, also in what state the machine currently is. For this, we let the state set of C be $(Q \cup \{\bot\}) \times \Sigma$ where Q is M's state set, $\bot$ does not occur in Q and Σ is M's tape alphabet. If c_i is in state (q, γ), this means that the i-th cell of M contains the symbol γ, M is in state q, and the head points to this cell. If c_i is in state $(\bot, \gamma)$, this means that the i-th cell contains symbol γ and the head is on some other cell. For example, the sequence

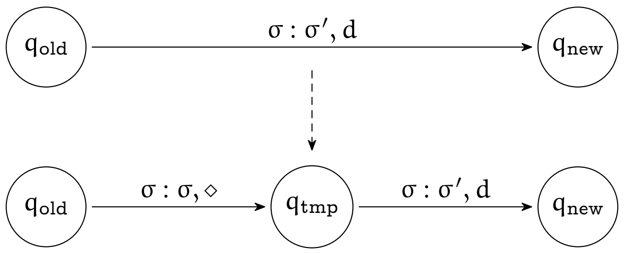
$(\bot, a)$	$(\bot, b)$	(q_1, b)	$(\bot, a)$

encodes that M's tape contains the string abba, M is in state q_1, and M's head points to the third tape cell. Using this encoding, C's state transition function can then simulate M's behaviour based on M's transition function. The following table shows how the transitions of the Turing machine define the transitions of the cellular automaton, showing the Turing machine transitions on the left hand side and the corresponding transitions of the cellular automaton on the right hand side. The symbols $\triangleright$, $\triangleleft$, $\diamond$ denote movements of the head of the Turing machine to the left, the right, and staying on the current cell, respectively. On the right, $(a, b, c) \mapsto d$ means that "a cell in state b whose left neighbor is in state a and right neighbor is in state c transitions into state d". Moreover, the placeholders x and y may also be interpreted as the "empty space" on the left or the right of the arrangement of cells to also cover the corner cases.

$$\begin{aligned}
(q,\sigma) \mapsto (q',\sigma',\triangleright) &: \big(x, (q,\sigma), y\big) \mapsto (\bot, \sigma') \\
& \big((q,\sigma), (\bot,\gamma), y\big) \mapsto (q', \gamma) \\
(q,\sigma) \mapsto (q',\sigma',\triangleleft) &: \big(x, (q,\sigma), y\big) \mapsto (\bot, \sigma') \\
& \big(x, (\bot,\gamma), (q,\sigma)\big) \mapsto (q', \gamma) \\
(q,\sigma) \mapsto (q',\sigma',\diamond) &: \big(x, (q,\sigma), y\big) \mapsto (q', \sigma')
\end{aligned}$$

The remaining part of the automaton's transition function can be defined arbitrarily, since these transitions correspond to cases that cannot occur, for instance the case when two heads of the Turing machine points to two neighboring cells.

The nondeterministic case now seems only a stone's throw away, but we have to be careful: Only generalizing from state transition functions to state transition relations does not suffice! The problem is that while in the deterministic case above we can be sure that at any time the automaton only simulates a single head, multiple heads can appear in the nondeterministic case, which we definitely not want. For example, consider the case where the Turing machine may nondeterministically choose between moving its head to the left and moving its head to the right. This translates into transitions for the automaton that allow both of the cells surrounding the cell currently having the head to take over the head. This results in the situation that the automaton simulates two heads at the same time, and, hence, to undesired computations that may erroneously accept the input. To overcome this, we make a preprocessing on the Turing machine before we derive the transition rules of the cellular automaton: We replace every transition $(q_{\text{old}}, \sigma) \to (q_{\text{new}}, \sigma', d)$ of the machine by two transitions $(q_{\text{old}}, \sigma) \to (q_{\text{tmp}}, \sigma, \diamond)$ and $(q_{\text{tmp}}, \sigma) \to (q_{\text{new}}, \sigma', d)$ using a fresh state q_{tmp}, such that the first transition effectively does nothing but change the state of the machine into a fresh intermediate state, and the second transition yields the old desired state together with the tape modification and the head movement done by the old transition. Here, d denotes the direction the machine's head moves to and $\diamond$ denotes that the head will not move. The following picture shows informally how we modify the transition relation of the machine:



The new machine now has the property that during a nondeterministic transition only the state of the machine changes, and, hence, movements of the head only occur at deterministic transitions. This fixes the issue of having transitions that "produce several heads".

In the timed cases our reductions essentially work in the same way as in the non-timed versions, but we additionally have to provide a time-bound. In the deterministic case it suffices to simply copy the already given time-bound. In the nondeterministic case we also have to double the length of the time-bound because the preprocessing on the Turing machine doubles the length of its computations.

For all these operations it is not hard to see that we can compute them efficiently, i. e., we obtain para-AC^0-reductions. □

We are now just a small step away from our target, namely the longest common subsequence problem. A last problem to solve is that the computation of a cellular automaton is a parallel computation while the sequences are inherently sequential. Hence, we have to somehow "sequentialize" the computations of cellular automatons if we want to reduce them to the longest common subsequence problem. For this, let us define *sequential cellular automatons*. We obtain them by modifying the computational model of cellular automatons such that, instead of making parallel steps, initially only the first cell makes a transition, then the second cell makes a transitions, and so on. If the last cell made its transition, this process is repeated, starting with the first cell. During this computation, if a cell has to make a transition, the cell can already see the new state of the previous cell. Based on this model we obtain the problems p-SEQUENTIAL-DCA, p-SEQUENTIAL-NCA, p-TIMED-SEQUENTIAL-DCA, and p-TIMED-SEQUENTIAL-NCA that are defined similar to the previously studied problems but now for sequential cellular automatons.

▷ *Theorem 68.*

1. p-SEQUENTIAL-DCA is para-AC^0-complete for XL.

2. p-Sequential-NCA is para-AC^0-complete for XNL.
3. p-Timed-Sequential-DCA is para-AC^0-complete for para-P/XL.
4. p-Timed-Sequential-NCA is para-AC^0-complete for para-NP/XNL.

Proof. We only show how to reduce p-DCA to p-Sequential-DCA because the other reductions work similarly and membership of the problems can be shown in the same way as in Theorem 67 where we showed that the non-sequential problems are complete for the corresponding classes. Hence, suppose that we are given a cellular automaton. What would happen if, instead of letting its cells work in a parallel fashion, the automaton would proceed in the sequential way described above? We cannot be sure that the result of the two computations are the same because the cells "see" different states at their neighbors during the computation: While in the first model every cell after the i-th transition sees the states of the neighboring cells after the i-th transition, in the second model every cell sees the state of its left cell after the $(i+1)$-th transition and the state of the right cell after the i-th transition. In order to be still able to simulate a purely parallel computation on the sequential cellular automaton, we construct from the given automaton C a new automaton C' whose cells are able to remember their previous state besides the current state. Formally, if the state set of C is Q, the state set of C' is $Q \times Q$, which intuitively can be seen as tuples whose first component is the previous state and the second component is the current state. For every transition $(q_{\text{left}}, q_{\text{old}}, q_{\text{right}}) \mapsto q_{\text{new}}$, in C the automaton C' thus has transitions

$$\Big((q_{\text{left}}, x), (y, q_{\text{old}}), (z, q_{\text{right}})\Big) \mapsto (q_{\text{old}}, q_{\text{new}})$$

where x, y, and z are arbitrary states. Now, the cells make their transitions based on the previous state of the left and the current state of the right cell, which is exactly what we want. What remains is to adapt this construction to the corner cases, i. e., the cells at the left and the right end of the automaton, but this can be done in the obvious manner. Now, if C started on the initial configuration $(q_1, \dots, q_k)$ arrives in configuration $(q'_1, \dots, q'_k)$ after t steps, C' started on the configuration $((x_1, q_1), \dots, (x_k, q_k))$ will arrive in the configuration $((x'_1, q'_1), \dots, (x'_k, q'_k))$ after $t \cdot k$ steps for arbitrarily chosen states $x_1, \dots, x_k$ and appropriate states $x'_1, \dots, x'_k$.

The other items can be proven in a similar way. In the nondeterministic cases we have to deal with relations instead of functions, in the timed cases we have to prolong the time bounds by the factor k, but this does not change the essentials of the construction. Moreover, all of these reductions can be implemented in para-AC^0. □

We are now ready to show the central result of this section:

▷ *Theorem 69.* p-LCS is complete for para-NP/XNL under para-AC^0-reductions.

Proof. To show that p-LCS $\in$ para-NP/XNL, we construct a nondeterministic machine that scans the first of the given strings and nondeterministically guesses the common subsequence during this scan, verifying on the fly that the other strings contain the guessed subsequence. For this, the machine requires $|S|-1$ pointers $p_2, \dots, p_{|S|}$ that are initially set to 0. The machine starts scanning the first string, and if it nondeterministically guesses for a scanned symbol σ that it is contained in the desired subsequence, the pointers p_i are increased in order to point to the position of the next occurence of σ in the i-th string. If it is possible to guess a sequence of length l in the first string and increase the pointers to the other strings accordingly, the machine accepts, otherwise it rejects. Since every such pointer has at most the value n, the overall space requirement is $O\big(|S| \cdot \log(n)\big)$. Moreover, the time requirement is clearly polynomial.

To prove hardness we reduce from p-TIMED-SEQUENTIAL-NCA. Hence, let $(C, q_1, \dots, q_k)$ together with a time bound be the input for the reduction. By Theorem 68 we may assume that the time bound is of the form $t \cdot k$. If necessary, we modify C such that it makes exactly $t \cdot k$ steps if it accepts and strictly less steps otherwise. We now construct $4 \cdot k$ strings $s_1^1, s_2^1, s_3^1, s_4^1, \dots, s_1^k, s_2^k, s_3^k, s_4^k$ and ask for a common subsequence of these strings of length $t \cdot k$. The idea is that each block of 4 strings $s_1^i, s_2^i, s_3^i, s_4^i$ encodes the behaviour of the i-th cell of the automaton and every symbol of the subsequence encodes a transition of the automaton.

Let us now construct the strings. To make the construction easier to explain, we first add some special symbols conceptually to the strings, i. e., we do not really add these symbols to the strings, but we treat them as if they were there. This way, we can use these symbols as markers, allowing us to specify certain positions exactly. The set of these *marker symbols* is defined by

$$\big\{ \langle q, i \rangle \mid q \in Q \wedge i \in \{1, \dots, tk\} \big\}$$

where Q with $Q = \{q_1, \dots, q_{|Q|}\}$ is the automaton's set of states. Intuitively, a symbol $\langle q, i \rangle$ on one of the strings $s_1^j, \dots, s_4^j$ represents that the j-th cell is in state q at the i-th step of the automaton's computation. Initially, we construct every block of 4 strings $s_1^i, s_2^i, s_3^i, s_4^i$ to be as follows:

$$\begin{array}{ll}
s_1^i & \langle q_1,1\rangle\langle q_2,1\rangle\ldots\langle q_{|Q|},1\rangle\langle q_1,3\rangle\langle q_2,3\rangle\ldots\langle q_{|Q|},3\rangle\langle q_1,5\rangle\langle q_2,5\rangle\ldots\langle q_{|Q|},5\rangle\ldots \\
s_2^i & \langle q_{|Q|},1\rangle\langle q_{|Q|-1},1\rangle\ldots\langle q_1,1\rangle\langle q_{|Q|},3\rangle\langle q_{|Q|-1},3\rangle\ldots\langle q_1,3\rangle\langle q_{|Q|},5\rangle\langle q_{|Q|-1},5\rangle\ldots\langle q_1,5\rangle\ldots \\
s_3^i & \langle q_1,2\rangle\langle q_2,2\rangle\ldots\langle q_{|Q|},2\rangle\langle q_1,4\rangle\langle q_2,4\rangle\ldots\langle q_{|Q|},4\rangle\langle q_1,6\rangle\langle q_2,6\rangle\ldots\langle q_{|Q|},6\rangle\ldots \\
s_4^i & \langle q_{|Q|},2\rangle\langle q_{|Q|-1},2\rangle\ldots\langle q_1,2\rangle\langle q_{|Q|},4\rangle\langle q_{|Q|-1},4\rangle\ldots\langle q_1,4\rangle\langle q_{|Q|},6\rangle\langle q_{|Q|-1},6\rangle\ldots\langle q_1,6\rangle\ldots
\end{array}$$

More formally, the above strings are defined by

$$s_1^i = \prod_{\substack{i=1 \\ i\equiv 1 \bmod 2}}^{t\cdot k} \prod_{j=1}^{|Q|} \langle q_j, i\rangle$$
$$s_2^i = \prod_{\substack{i=1 \\ i\equiv 1 \bmod 2}}^{t\cdot k} \prod_{j=|Q|}^{1} \langle q_j, i\rangle$$
$$s_3^i = \prod_{\substack{i=2 \\ i\equiv 0 \bmod 2}}^{t\cdot k} \prod_{j=1}^{|Q|} \langle q_j, i\rangle$$
$$s_4^i = \prod_{\substack{i=2 \\ i\equiv 0 \bmod 2}}^{t\cdot k} \prod_{j=|Q|}^{1} \langle q_j, i\rangle$$

where we denote the concatenation of symbols using the product symbol. Note that some of the concatenations are in decreasing order! Due to this opposite ordering, we will later have that no cell can be in two different states at the same step of the computation.

Using the marker symbols, we now insert the real symbols of the string. The real symbols will be tuples (q, s, i) where $q = (q_{\text{left}}, q_{\text{old}}, q_{\text{right}}, q_{\text{new}})$ is an element of the transition relation, representing a possible transition of a cell from q_{old} to q_{new} if the left neighboring cell is in state q_{left} and the right neighboring cell is in state q_{right}, s is a step number, and i is a cell number. Let us call these elements *transition symbols*. These elements are inserted into the strings according to the following rules:

1. Iterate over all (q, s, i) in some fixed order and insert (q, s, i) directly after $\langle q_{\text{old}}, s\rangle$ in s_1^i if s is odd and in s_3^i if s is even. This ensures that the transition q of cell i at step number s can only be made if the cell i was in state q_{old} directly before step number s is made.
2. Iterate over all (q, s, i) but now in reverse order, and insert (q, s, i) after $\langle q_{\text{old}}, s\rangle$ in s_2^i if s is odd and in s_4^i if s is even. Due to the two opposite orderings by this and the previous rule, it is now only possible to select at most one of the many transitions of cell i at step s if the previous state

of the cell was q_{old}. It is not possible to select two symbols (q_1, s, i) and (q_2, s, i) with $q_1 \neq q_2$ into a common subsequence, because they appear in different orderings in s_1^i and s_2^i or s_3^i and s_4^i.

3. Iterate over all (q, s, i) in some order and insert (q, s, i) directly before $\langle q_{\text{new}}, s+1 \rangle$ in s_3^i if s is odd and s_1^i if s is even. This ensures that the state of cell i directly before step $s+1$ is at least q_{new} (with respect to the ordering of the states).
4. Iterate over all (q, s, i) but now in reverse order, and insert (q, s, i) directly before $\langle q_{\text{new}}, s+1 \rangle$ in s_4^i if s is odd and s_2^i if s is even, in order to achieve a similar effect as the combination of rules 1 and 2. Furthermore, all the rules we have seen so far together with the opposite ordering of the marker symbols ensure that (conceptually) for every step at most one marker symbol can be chosen into a common subsequence and, moreover, that the corresponding states of the marker symbols are consistent with the chosen transition symbols.
5. Iterate over all (q, s, i) and insert (q, s, i) directly after $\langle q_{\text{left}}, s+1 \rangle$ in s_3^{i-1} and s_4^{i-1} if s is odd and in s_1^{i-1} and s_2^{i-1} if s is even. If $i = 1$, then no symbols are added to the strings. This ensures that the transition q of cell i at step s can only be made if the left cell previously switched to state q_{left}.
6. Iterate over all (q, s, i) and insert (q, s, i) directly after $\langle q_{\text{right}}, s \rangle$ in s_1^{i+1} and s_2^{i+1} if s is odd and in s_3^{i+1} and s_4^{i+1} if s is even. If $i = k$, then no symbols are added. This ensures that the transition q of cell i at step s can only be made if the right cell previously switched to state s_{right}.
7. Iterate over all the $4 \cdot k$ strings, and, for each such string s_j^i, consider the set of all (q, s, l) that are not present in s_j^i. Add all of these symbols in some fixed order $t \cdot k$ times after each symbol of s_j^i. Since the previous rules only added symbols to specific subsets of strings, this rule allows us to select a subsequence of symbols that occurs in *every* string, without influencing the restrictions on the subsequences introduced by the previous rules.
8. Finally, in order to reflect the initial configuration $p_1, \ldots, p_k$, iterate over all strings s_j^i and remove all symbols before $\langle q_i, 1 \rangle$.

Note that the order of the rules above is important: Since rules 5 and 6 are applied later than rules 3 and 4, the added symbols are closer to the associated marker symbols, thus enforcing that the neighboring cells are in the correct state before the currently considered cell can make its transition.

To see that the above construction is correct, first assume that the automaton does accept its input $q_1, \dots, q_k$. By assumption, we then know that the automaton makes $t \cdot k$ steps. Let the transitions of these steps be

$$f^{1,1}, f^{1,2}, \dots, f^{1,k}, f^{2,1}, f^{2,2}, \dots, f^{t,k},$$

where $f^{i,j}$ with $f^{i,j} = (f^{i,j}_{\text{left}}, f^{i,j}_{\text{old}}, f^{i,j}_{\text{right}}, f^{i,j}_{\text{new}})$ is the element of the transition relation used for the i-th transition of cell j. Let us now show that

$$(f^{1,1}, 1, 1)(f^{1,2}, 1, 2) \dots (f^{2,1}, 2, 1) \dots (f^{t,k}, t, k)$$

is a common subsequence of the constructed set of strings, and, thus, the strings contain a common subsequence of length $t \cdot k$ as desired. Consider the first symbol, $(f^{1,1}, 1, 1)$, and let $f^{1,1}$ be the transition of the first cell from state $f^{1,1}_{\text{old}}$ to $f^{1,1}_{\text{new}}$. By rules 1 and 2, $(f^{1,1}, 1, 1)$ is present in s^1_1 and s^1_2 after $\langle f^{1,1}_{\text{old}}, 1\rangle$ since $f^{1,1}_{\text{old}} = q_1$ and it has not been removed by rule 8. Moreover, the symbol is present in s^1_3 and s^1_4, namely right before $\langle f^{1,1}_{\text{new}}, 2\rangle$. By rule 6, the symbol is also contained in s^2_1 and s^2_2 directly after $\langle f^{1,1}_{\text{right}}, 1\rangle$. In all other strings, the symbol is contained due to rule 7. Now consider the symbol $(f^{1,2}, 1, 2)$, which represents the first transition of cell 2, going from $f^{1,2}_{\text{old}}$ to $f^{1,2}_{\text{new}}$ with $f^{1,2}_{\text{old}} = q_2$, $f^{1,2}_{\text{left}} = f^{1,1}_{\text{new}}$ and $f^{1,2}_{\text{right}} = q_3$. In all of the constructed strings $(f^{1,2}, 1, 2)$ indeed comes after $(f^{1,1}, 1, 1)$: In the strings s^2_1 to s^2_4 this is due to the rules 1, 2, 3, 4. In strings s^1_3 and s^1_4 this is due to rule 5, and in strings s^3_1 and $^3_2$ this is due to rule 6. In all other strings, the symbol is contained due to rule 7. This continues in a similar fashion for the remaining symbols of the subsequence. Moreover, the sequence above clearly has the desired length $t \cdot k$.

Let us now argue that if there is a common subsequence of the constructed strings that has length $t \cdot k$, then the automaton accepts the input. First note that in order to have a sequence of length $t \cdot k$, the sequence must be of the form

$$(f^{1,1}, 1, 1)(f^{1,2}, 1, 2) \dots (f^{1,k}, 1, k) \dots (f^{2,1}, 2, 1) \dots (f^{t,k}, t, k).$$

This is because for any two symbols (f^{s_1,i_1}, s_1, i_1) and (f^{s_2,i_2}, s_2, i_2) we have that if $s_1 < s_2$, then the first symbol comes before the second symbol in every of the constructed strings because we inserted them directly next to the marker symbols for steps s_1 and s_2 and the marker symbols are placed in the strings in increasing order with respect to the step number. If we have $s_1 = s_2$ and $i_1 < i_2$, then the first symbol again comes before the second symbol because rules 5 and 6 are applied after rules 1, 2, 3, 4, placing the symbols closer to the marker symbols and thus ensuring the ordering. If $s = s'$ and $i = i'$, then the opposite orderings in rules 1, 2 and 3, 4 guarantee, that at most one of these two symbols may occur in the subsequence. Overall, the symbols of the subsequence must strictly increase, and since the length of the subsequence has to be $t \cdot k$, the sequence has to be of the form presented above. We now have to argue that the transitions of the sequence make the automaton accept. The crucial observation here is that for every i with $i < k$ the only symbol $(f^{s,i+1}, s, i+1)$ that can follow $(f^{s,i}, s, i)$ is a symbol that encodes a transition $f^{s,i+1}$ that makes cell $i+1$ change its state according to this transition. If $i = k$, then we similarly have that the only symbol $(f^{s+1,1}, s+1, 1)$ that can follow encodes a transition that makes cell 1 switch its state according to this transition. Since rule 8 ensures that the first transitions $(f^{1,i}, 1, i)$ start from the initial configuration of the automaton, the overall sequence is therefore a valid computation. Together with the assumption that the automaton accepts its input if, and only if, it makes $t \cdot k$ steps, we have proven the claim of this direction and, thus, of the whole theorem. □

The above result shows that parameterized time-space classes serve as the right tool for the exact classification of previously unclassifiable parameterized problems like the longest common subsequence problem.

Let us conclude this section with revisiting two problems we already discussed several times in this thesis: The feedback vertex set problem for directed and undirected graphs, i. e., the question whether there is a subset of the vertices of a given graph such that removing the vertices of this subset from the graph makes it acyclic. For a formal definition of these problems see page 28. We already saw that while the theory of parameterized time complexity only reveals that both the directed and the undirected version are fixed-parameter tractable, the theory of parameterized space and circuit complexity discussed in this thesis so far gives deeper insights into the complexity of these two problems: In Section 2.2 we showed that if p-FVS $\in$ para-NC^1, then $\mathrm{NC}^1 = \mathrm{L}$, and if p-DFVS $\in$ para-L, then $\mathrm{L} = \mathrm{NL}$. In Section 3.2 we extended this by showing that p-FVS $\in$ $\mathrm{para}_{\mathrm{W}}$-$\mathrm{NC}^1$ implies $\mathrm{NC}^1 = \mathrm{L}$, and p-DFVS $\in$ $\mathrm{para}_{\mathrm{W}}$-L implies $\mathrm{L} = \mathrm{NL}$. Parameterized time-space classes, however, reveal even more of the nature of these problems:

▷ *Theorem 70.*

1. p-FVS $\in$ para-P/XL.
2. If p-DFVS $\in$ para-P/XL, then $\mathrm{L} = \mathrm{NL}$.

Proof. Let us start with the first item. Our proof is based on an algorithm of Downey and Fellows (1997) showing that the undirected feedback vertex set problem is fixed-parameter tractable. The crucial part here is to modify their algorithm such that it only requires space at most $O\big(f_k \cdot \log(n)\big)$ while it remains fast, i. e., requires time at most $O\big(f_k \cdot p(n)\big)$ for a polynomial p.

In the following, we will repeatedly use phrases like "remove the vertex from the graph". This may be misleading. Since we consider machines with very small space, we cannot store the resulting graph and, hence, do such an operation like it would be done in the domain of parameterized time. Instead, we make use of the fact that logarithmic space-bounded computations are closed under composition. For a more detailed elaboration on this see the textbook of Papadimitriou (1994). In short: Instead of removing the vertices once and for all from the graph and storing the result, we recompute the bits of the result that are required by the subsequent parts of the computation. These recomputations only add a polynomial factor to the running time, which is okay because we only require to stay within the time bounds of para-P.

Let G with $G = (V, E)$ be the given undirected graph and k be the parameter. Since the vertices of a feedback vertex set only affect the cycles within the connected component of the vertex, we consider the connected components of the graph individually. Hence, let us assume that G is a connected component. First, we test whether G admits a feedback vertex set of small size, i. e., size 0 or 1. To check if there is a feedback vertex set of size 0, we simply test whether G is acyclic, to check if there is a feedback vertex set of size 1, we test whether G gets acyclic after removing one vertex with its incident edges. Checking whether a graph is acyclic or, equivalently, whether a graph is a forest can be done in logarithmic space, see Cook and McKenzie (1987). At this point, we accept if

- G admits a feedback vertex set of size 0 and $k \geq 0$, or
- G admits a feedback vertex set of size 1 and $k \geq 1$,

and we reject if

- G does not admit a feedback vertex set of size 0 and $k \leq 0$, or
- G does not admit a feedback vertex set of size 1 and $k \leq 1$.

For the remaining case where the feedback vertex set has size at least 2 and $k \geq 2$, we have to do more work. Our plan is as follows: First, we apply some preprocessing to G. This preprocessing then enables us to apply a useful lemma which essentially states that we only have to detect short cycles in a graph in order to find a feedback vertex set. Finally, we show how to find these cycles using logarithmic space.

First, we want to remove vertices of degree 1 from G because it does not make sense to choose them for the feedback vertex set, and we want to do this iteratively because removing vertices of degree 1 from the graph may yield new vertices of degree 1. However, we cannot perform this iteration in the straightforward way because of the restricted amount of available space. Instead, we remove larger components at once: We remove a vertex v if its connected component is a tree, or if it is possible to remove an edge e from G such that G decomposes into two non-empty components where v's connected component is a tree and contains at most one vertex from e. Since testing whether a graph is a tree is possible in logarithmic space, see Cook and McKenzie (1987), we can remove all these vertices independently of each other from G. Let us from now on denote the resulting graph by G.

Next, we shrink paths within the graph. Let $v_1, v_2, \dots, v_{l-1}, v_l$ be a path from v_1 to v_l where v_1 and v_l have degree larger than 2 and $v_2, \dots, v_{l-1}$ each have degree 2. Note that if removing any of the vertices $v_2, \dots, v_{l-1}$ from the graph would destroy one of the graphs cycles, then we can instead remove v_1 or v_l because they also lie on the circle. Hence, we can compress these paths by, for every vertex v of G and degree at least 3, following its outgoing paths of vertices of degree 2 to a lexicographically larger vertex v' of degree at least 3. We then insert an edge between v and v', and remove the inner vertices of the path together with its incident edges. In the resulting graph, every vertex has degree at least 3, and there may be self-loops and multiple edges between pairs of vertices. Let us again denote the resulting graph by G.

If G contains a vertex v with a self-loop, we know that v has to be in the feedback vertex set because the original input graph contained a cycle from v to v via vertices of degree 2. We hence store v as one of the vertices of the solution. Then, we restart the whole algorithm but with v removed from the original graph. A similar situation arises if we find two vertices u and v between which there are two edges. In this case, we know that at least one of them has to be in the feedback vertex set. Hence, we do a branching over these two possibilities, leading to a search tree with branching width 2 and depth at most k.

Let us now consider the case that G does neither contain self-loops nor multiple edges between two vertices and every vertex has degree at least 3. We can now make use of a lemma by Downey and Fellows (1997), saying that if an undirected graph has minimum degree 3 and a feedback vertex set of size k, then the shortest cycle in the graph has length at most $2k$. Using this lemma, it remains to find these cycles of short length and, since at least one of the vertices of such a cycle has to be contained in a feedback vertex set, do a branching over the vertices of such a cycle, considering each as a possible member of the solution. This branching will lead to a search tree of size $(2k)^k$ and in toto yield a running time of $O\big((2k)^k \cdot n^c\big)$ for some constant c. What remains is to show how to identify these cycles quickly using little space. For this, we make use of two nested depth-first search loops. Starting at a vertex v, the outer loop visits all vertices reachable from v for increasing depth d with $d = 1, 2, \dots, k$. For this, the outer loop requires space $O\big(d \cdot \log(n)\big)$ to maintain a stack storing the currently considered path starting at v. The inner loop does the same using another stack. We stop both loops if their paths lead to the same vertex w on different paths because then we have found a cycle. Note that if v lies on a cycle of length at most $2k$, then this method will find this cycle. While this algorithm requires only $O\big(k \cdot \log(n)\big)$ space, we still have to argue that it only requires little time. The important observation here is that the loops stop if they have identified two ways to reach a vertex w from a common starting vertex v, and since they visit vertices with increasing distance, this means that until they have reached a common vertex, there is a unique path to every vertex visited so far. Hence, both the outer loop and the inner loop can, for each distance, only visit at most n vertices, yielding at most n^3 steps. Since we have to start this algorithm on any vertex of the graph, the search requires at most n^4 steps.

To show the second item, i. e., if p-DFVS $\in$ para-P/XL, then we have L $=$ NL, we can reuse the construction used in Theorems 12 and 45, that gives a reduction from the NL-complete problem UNREACHABILITY using layerings and a trivial parameterization. The important observation now is that a para-P/XL-machine working on a constant parameter value is in fact an L-machine. □

Once more, parameterized space complexity has proven to be a fruitful concept – this time in combination with parameterized time in form of parameterized simultaneous time-space classes. Figure 4.1 gives an overview of the results from this chapter. We have seen that the previously unclassifiable parameterized longest common subsequence problem is complete for para-P/XL under para-AC^0-reductions, finally settling this long-standing open problem. This alone justifies the study of parameterized simultaneous time-space classes. But even if we do not talk about completeness, we can profit from the concept of simultaneous time-space classes: Regarding the feedback vertex set problems, we have seen that both p-FVS and p-DFVS are members of para-P and that using parameterized space and circuit classes provide additional upper and lower bounds that are directly related to important open questions of classical computational complexity theory. In this chapter we have seen that p-FVS lies in para-P/XL, but p-DFVS does presumably not, thus giving us another aspect that separates these problems: While both problems can be solved efficiently in the relaxed notion of fixed-parameter tractability, p-FVS admits a space-efficient para-P-algorithm but p-DFVS does presumably not – a fact that is clearly not exhibited by parameterized time classes alone but in combination with parameterized space and circuit classes.

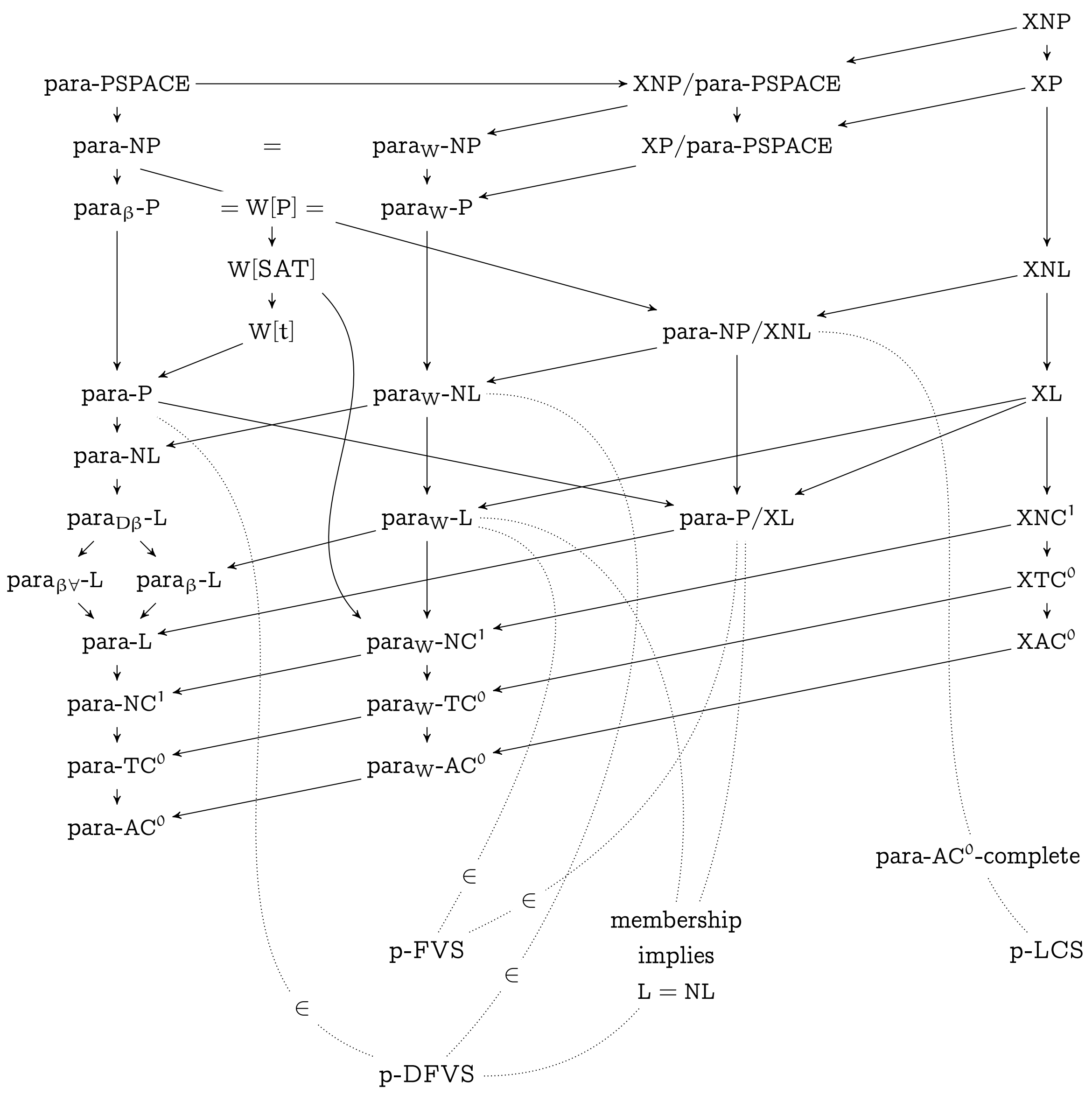


Figure 4.1: Diagram of classes of simultaneous space and time together with some of their surrounding classes and inclusions where $A \rightarrow B$ denotes the inclusion $A \supseteq B$. Moreover, the relations between the problems p-LCS, p-FVS, and p-DFVS and the classes discussed in this chapter are illustrated.

5. Conclusion

In this thesis we discussed the space and circuit complexity of parameterized problems. I tried to convince you that, in order to fully understand the computational complexity of parameterized problems, the study of their space and circuit complexity is unavoidable. For this, I presented to you a wide variety of complexity classes and showed for numerous natural problems how they are related to these classes. In this conclusion, I will first give you a brief summary of what we have seen and achieved in this thesis, and then I will discuss some future research directions.

5.1. Summary

This thesis consisted of three main parts: The first gave an introduction to parameterized space and circuits, and discussed parameterized problems as well as para-classes and X-classes from the perspective of space-bounded computations and computations done by circuits. In the second part we augmented the classes we have seen so far by studying classes of bounded nondeterminism. In the third part we then combined time and space by studying classes that are defined by computations that are simultaneously bounded with respect to both time and space.

5.1.1. Parameterized Space and Circuits

After a brief introduction of the concept of parameterized problems, we discussed the idea of fixed-parameter tractability, and made use of its generalization, namely the computation after a precomputation, to define parameterized variants of (nondeterministic) logarithmic space and circuit classes. Using these para-classes, we showed several upper and lower bounds for problems like p-VERTEX-COVER, p-FVS, and p-DFVS, giving us a more detailed view of the complexities of these problems that are much more detailed than the insights that we obtained from the perspective of parameterized time alone: p-VERTEX-COVER lies in para-AC^0, a class lying deeply within para-P;

p-FVS $\in$ para-NC^1 implies $NC^1 = L$, and p-DFVS $\in$ para-L implies $L = NL$. Besides the para-classes, we also defined X-classes based on classical space and circuit classes, and we used them to show further upper and lower bounds: Both p-VERTEX-COVER and p-CLIQUE lie in XAC^0 but p-DFVS and p-FVS do not. Finally, we discussed several reduction notions that are appropriate for the classes we are dealing with, and had a review on the famous Weft-Hierarchy from the perspective of parameterized space and circuit theory we just obtained.

5.1.2. Bounded Nondeterminism

Motivated by the Weft Hierarchy, we defined the notion of bounded nondeterminism where we allow the access to $O(f_k \cdot \log(n))$ nondeterministic bits. Depending on whether we allow these bits to be read only once or read repeatedly, we obtained several parameterized classes that capture the complexity of several interesting problems.

For the case that we allowed the nondeterministic bits to be read repeatedly, i. e., if we considered classes like para_W-L or para_W-NC^1, we first showed that many natural problems like the colored reachability problem or the weighted satisfiability problem can be expressed as union problems. Using with the concept of format-preserving projections, we proved the union lemma, a useful ingredient for showing completeness of problems for classes of bounded nondeterminism that may be read several times. We applied this lemma to show that, for example, p-WEIGHTED-SAT is complete for para_W-NC^1 and that p-AGEN is complete for para_W-NL. Moreover, we investigated both the directed and the undirected feedback vertex set problem from the perspective of bounded nondeterminism which yielded additional upper and lower bounds that are linked to open questions from classical computational complexity: p-FVS $\in$ para_W-L, p-DFVS $\in$ para_W-NL, p-FVS $\in$ para_W-NC^1 holds if, and only if, $NC^1 = L$, and p-DFVS $\in$ para_W-L holds if, and only if, $L = NL$.

For the case that the nondeterministic bits may only be read once, i. e., in the case that we consider classes like para_β-L or $\text{para}_{\beta\forall}$-L, we showed that p-DISTANCE is complete for para_β-L and that many variants of this problem like p-DISTANCE-AT-LEAST or p-LONGEST-PATH are related to the distance problem are complete for related classes, giving us a much more detailed picture on para-P.

5.1.3. Simultaneous Time-Space Classes

Combining time and space, we studied machines that respect time and space bounds simultaneously. While impossible for reasonable classes in the case of classical computational complexity, this has proven a fruitful task in the case of parameterized complexity theory. After definitions of classes and the study of admittedly artificial problems, we were able to show completeness of p-LCS for para-NP/XNL using para-AC^0-reductions, answering a long-standing open question on the classification of this problem for reasonable and robust complexity classes. Moreover, we also showed that the undirected feedback vertex set problem lies in para-P/XL while this is not the case for the directed version of this problem under reasonable assumptions, a result underlining that the difference of the directed and the undirected version of the feedback vertex set problem lies in their space complexities.

5.2. Outlook

The study of parameterized space and circuits has proven as a fruitful complementation of the concept of parameterized time. Parameterized space and circuits gave us deep insights into the nature of parameterized problems and raised several new problems. First of all, the classes discussed in this thesis allow us to continue the hunt for new upper bounds because only showing that a problem is fixed-parameter tractable is not enough any more. In order to obtain a full view on the complexity of a parameterized problem, parameterized space and circuit classes enable us to continue the study of problems inside of para-P. Furthermore, also lower bounds are getting more possible to reach: While proving that a problem does not lie in para-P can be hard or even impossible, it is now possible to start showing smaller lower bounds, making it easier or possible at all to attack a problem by starting with a small lower space or circuit bound and raising it. Finally, the study of parameterized time, space, and circuits yield, instead of a hierarchy of classes, a multi-dimensional grid of classes that allows us to look at problems from many perspectives. Studying the relations between these classes, separating them or showing that some of them coincide is a promising task in order to find answers for the relations between time and space in computations.

Bibliography

Max Bannach, Christoph Stockhusen, and Till Tantau. Fast Parallel Fixed-Parameter Algorithms via Color Coding. In Thore Husfeldt and Iyad Kanj, editors, *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, volume 43, pages 224–235. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi: 10.4230/LIPIcs.IPEC.2015.224.

Hans L. Bodlaender. On Linear Time Minor Tests with Depth-First Search. *Journal of Algorithms*, 14:1–23, 1993.

Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Harold T. Wareham. The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science*, 147:31–54, 1995. doi: 10.1016/0304-3975(94)00251-D.

J. Richard Büchi. Weak Second-Order Arithmetic and Finite Automata. *Mathematical Logic Quarterly*, 6:66–92, 1960. doi: 10.1002/malq.19600060105. Originally appeared in "Zeitschrift für mathematische Logik und Grundlagen der Mathematik".

Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22(3):560–572, 1993.

Jonathan F. Buss and Tarique Islam. Simplifying the weft hierarchy. *Theoretical Computer Science*, 351:303–313, 2006. doi: 10.1016/j.tcs.2005.10.002.

Jonathan F. Buss and Tarique Islam. Algorithms in the W-Hierarchy. *Theory of Computer Systems*, 41:445–457, 2007. doi: 10.1007/s00224-007-1325-3.

S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An Optimal Parallel Algorithm for Formula Evaluation. *SIAM Journal on Computing*, 21(4):755–780, 1992. doi: 10.1137/0221046.

S. R. Buss. The Boolean Formula Value Problem is in ALOGTIME. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 123–131. ACM, 1987. doi: 10.1145/28395.28409.

Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84:119–138, 1997. doi: 10.1016/S0168-0072(95)00020-8.

Yixin Cao, Jianer Chen, and Yang Liu. On Feedback Vertex Set: New Measure and New Structures. In Haim Kaplan, editor, *Algorithm Theory – SWAT 2010*, volume 6139, pages 93–104. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-13731-0_10.

Hubie Chen and Moritz Müller. The Fine Classification of Conjunctive Queries and Parameterized Logarithmic Space. *ACM Transactions of Computation Theory*, 7(2):1–27, 2015. doi: 10.1145/2751316.

Jianer Chen and Jie Meng. On Parameterized Intractability: Hardness and Completeness. *The Computer Journal*, 51(1):39–59, 2008. doi: 10.1093/comjnl/bxm036.

Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved Parameterized Upper Bounds for Vertex Cover. In *Mathematical Foundations of Computer Science 2006*, volume 4162, pages 238–249. Springer Berlin Heidelberg, 2006. doi: 10.1007/11821069_21.

Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A Fixed-Parameter Algorithm for the Directed Feedback Vertex Set Problem. *Journal of the ACM*, 55(5):1–19, 2008. doi: 10.1145/1411509.1411511.

Yijia Chen and Jörg Flum. Machine Characterizations of the Classes of the W-Hierarchy. In Matthias Baaz and Johann A. Makowsky, editors, *Computer Science Logic*, volume 2803, pages 114–127. Springer-Verlag Berlin Heidelberg, 2003. doi: 10.1007/978-3-540-45220-1_12.

Yijia Chen, Jörg Flum, and Martin Grohe. Bounded Nondeterminism and Alternation in Parameterized Complexity Theory. In *Proceedings of 18th IEEE Annual Conference on Computational Complexity*, pages 13–29. IEEE, 2003. doi: 10.1109/CCC.2003.1214407.

Yijia Chen, Jörg Flum, and Martin Grohe. Machine-based methods in parameterized complexity theory. *Theoretical Computer Science*, 339:167–199, 2005. doi: 10.1016/j.tcs.2005.02.003.

Stephen A. Cook and Pierre McKenzie. Problems Complete for Deterministic Logarithmic Space. *Journal of Algorithms*, 8:385–394, 1987. doi: 10.1016/0196-6774(87)90018-6.

Steven A. Cook. The Complexity of Theorem-Proving Procedures. In *STOC '71, Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971. doi: 10.1145/800157.805047.

R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer New York, 1997. doi: 10.1007/978-1-4612-0515-9.

Rod G. Downey and Michael R. Fellows. Fixed-Parameter Tractability and Completeness I: Basic Results. *SIAM Journal on Computing*, 24(4): 873–921, 1995a. doi: 10.1137/S0097539792228228.

Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141:109–131, 1995b. doi: 10.1016/0304-3975(94)00097-3.

Rodney G. Downey and Michael R. Fellows. Parameterized Computational Feasibility. In Peter Clote and Jeffrey B. Remmel, editors, *Feasible Mathematics II*, 13, page 219–244. Birkhäuser Boston, 1995c. doi: 10.1007/978-1-4612-2566-9_7.

Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace Versions of the Theorems of Bodlaender and Courcelle. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 143–152. IEEE, 2010. doi: 10.1109/FOCS.2010.21.

Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the Space and Circuit Complexity of Parameterized Problems. In Dimitrios M. Thilikos and Gerhard J. Woeginger, editors, *Parameterized and Exact Computation – 7th International Symposium, IPEC 2012, Ljubljana, Slovenia, September 2012, Proceedings*, volume 7535 of *Lecture Notes in Computer Science*, pages 206–217. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-33293-7_20.

Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and Circuit Complexity of Parameterized Problems: Classes and Completeness. *Algorithmica*, 71:661–701, 2015. doi: 10.1007/s00453-014-9944-y.

Jörg Flum and Martin Grohe. Describing parameterized complexity classes. *Information and Computation*, 187(2):291–319, 2003. doi: 10.1016/S0890-5401(03)00161-5.

Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer-Verlag Berlin Heidelberg New York, 2006. doi: 10.1007/3-540-29953-X.

Merrick Furst, James B. Saxe, and Michael Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984. doi: 10.1007/BF01744431.

Michael Gardner. The Fantastic Combinations of John Conways New Solitaire Games. *Scientific American*, 223:120–123, 1970.

Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

S. Guillemot. Parameterized complexity and approximability of the Longest Compatible Sequence problem. *Discrete Optimization*, 8(1):50–60, 2011. doi: 10.1016/j.disopt.2010.08.003.

Juris Hartmanis and Richard E. Stearns. On the Computational Complexity of Algorithms. *Transactions of the American Mathematical Society*, 117: 285–306, 1965.

Neil Immerman. Languages that Capture Complexity Classes. *SIAM Journal on Computing*, 16(4):760–778, 1987. doi: 10.1137/0216051.

Neil Immerman. Nondeterministic Space is Closed under Complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988. doi: 10.1137/0217058.

Neil Immerman. *Descriptive Complexity*. Graduate Text in Computer Science. Springer New York, 1999. doi: 10.1007/978-1-4612-0539-5.

Neil D. Jones. Space-Bounded Reducibility among Combinatorial Problems. *Journal of Computer and System Sciences*, 11:68–85, 1975. doi: 10.1016/S0022-0000(75)80050-X.

Neil D. Jones and William T. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3:105–117, 1976. doi: 10.1016/0304-3975(76)90068-2.

Neil D. Jones, Edmund Lien, and William T. Laaser. New Problems Complete for Nondeterministic Log Space. *Mathematical Systems Theory*, 10:1–17, 1976. doi: 10.1007/BF01683259.

Richard M. Karp. Reducibility among Combinatorial Problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, page 85–103. Springer US, 1972. doi: 10.1007/978-1-4684-2001-2_9.

David Maier. The Complexity of Some Problems on Subsequences and Supersequences. *Journal of the ACM*, 25(2):322–336, 1978. doi: 10.1145/322063.322075.

Ilan Newman, Prabhakar Ragde, and Avi Wigderson. Perfect Hashing, Graph Entropy, and Circuit Complexity. In *Proceedings of the 5th Structure in Complexity Theory Conference*, pages 91–99. IEEE, 1990. doi: 10.1109/SCT.1990.113958.

Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Habilitation thesis, Universität Tübingen, 2002.

C. H. Papadimitriou and M. Yannakakis. The Complexity of Facets (and Some Facets of Complexity). *Journal of Computer and System Sciences*, 28: 244–259, 1984. doi: 10.1016/0022-0000(84)90068-0.

Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shorest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67:757–771, 2003. doi: 10.1016/S0022-0000(03)00078-3.

M. O. Rabin and D. Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3:114–125, 1959. doi: 10.1147/rd.32.0114.

Omer Reingold. Undirected Connectivity in Log-Space. *Journal of the ACM*, 55(17):1–24, 2008. doi: 10.1145/1391289.1391291.

R. E. Stearns, J. Hartmanis, and P. M. Lewis II. Hierarchies of Memory Limited Computations. In *Proceedings of the Sixth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 179–190. IEEE Computer Society, 1965. doi: 10.1109/FOCS.1965.11.

Christoph Stockhusen and Till Tantau. Completeness Results for Parameterized Space Classes. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation – 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4–6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 335–347. Springer International Publishing, 2013. doi: 10.1007/978-3-319-03898-8_28.

Till Tantau. Logspace Optimisation Problems and Their Approximability Properties. In Maciej Liśkiewicz and Rüdiger Reischuk, editors, *Fundamentals of Computation Theory, Proceedings of the 15th International Symposium, FCT 2005, Lübeck, Germany, August 17-20, 2005*, pages 103–114. Springer Berlin Heidelberg, 2005. doi: 10.1007/11537311_10.

Heribert Vollmer. *Introduction to Circuit Complexity*. Texts in Theoretical Computer Science, An EATCS Series. Springer-Verlag Berlin Heidelberg, 1999. doi: 10.1007/978-3-662-03927-4.

Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.