Parameterized analysis of complexity

Witteveen, J.E.

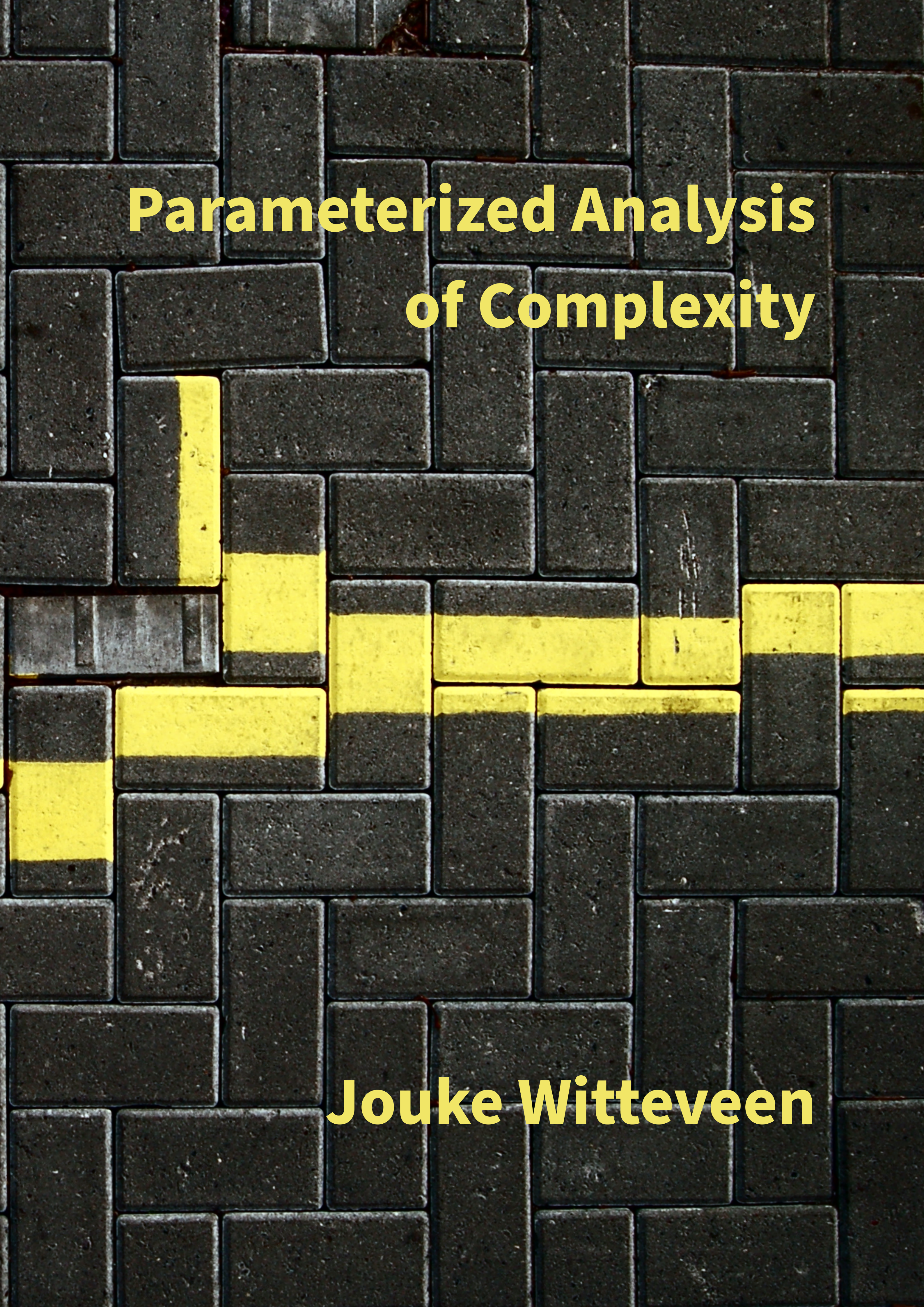
Publication date
2020

Document Version
Final published version

License
Other

Link to publication

Citation for published version (APA):
Witteveen, J. E. (2020). *Parameterized analysis of complexity*. [Thesis, fully internal, Universiteit van Amsterdam]. Institute for Logic, Language and Computation.



Parameterized Analysis of Complexity

Jouke Witteveen

Parameterized Analysis of Complexity

ILLC Dissertation Series DS-2020-03

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 107
1098 XG Amsterdam
phone: +31-20-525 6051
e-mail: illc@uva.nl
homepage: http://www.illc.uva.nl/

Printed by Drukkerij Mostert, Leiden

ISBN: 978–94–90858–64–3

Parameterized Analysis of Complexity

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K. I. J. Maex
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen in de Aula der Universiteit
op woensdag 29 januari 2020, te 11.00 uur

door

Jouke Erik Witteveen

geboren te Leeuwarden

Promotiecommisie

Promotores:	Prof. dr. R. M. de Wolf	Universiteit van Amsterdam
	Dr. L. Torenvliet	Universiteit van Amsterdam
Co-promotor:	Prof. dr. S. J. L. Smets	Universiteit van Amsterdam
Overige leden:	Prof. dr. H. L. Bodlaender	Universiteit Utrecht
	Prof. dr. P. van Emde Boas	Universiteit van Amsterdam
	Prof. dr. ir. C. T. A. M. de Laat	Universiteit van Amsterdam
	Prof. dr. Y. Venema	Universiteit van Amsterdam
	Dr. W. M. Koolen	CWI

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

01001 01110 00110 01111 10010 01101 00001 10100 01001 01111 01110

Contents

List of Slogans

Acknowledgments

For the past few years, I have studied a topic in computer science in more than a reasonable amount of detail. This inevitably meant that I isolated myself somewhat from *normal* people. Luckily, a few people were willing to follow me in my mental endeavors. Not only that, some even supplied valuable guidance, for which I am very grateful.

First and foremost, I want to thank one of the most sincere academics that I know, my supervisor, Leen Torenvliet. I have never caught him thinking of The University as a producer of results instead of as a place where humans cultivate and share knowledge. Because of that mentality, I was allowed to use my curiosity as the most important guidance in the creation of this thesis.

Next, I am much obliged to Ronald de Wolf and Peter van Emde Boas. Both went out of their way to see the production of this thesis through to its completion. If it was not for Ronald, there would be so little redundancy in this thesis that its information content would resemble random noise. His guidance made all the difference. As for Peter, an earlier version of this thesis was, in many places, only "correct up to pesky little details". The fact that the current version is much improved in this regard is the result of his many useful and pleasantly specific comments.

Also outside the ILLC community, I have met people that were willing to help out where they could. In Ralph Bottesch, I found someone equally quirky in his approach to parameterized complexity theory as myself. I much appreciate our collaboration. At this point, I will also mention the awesome teachers I had at the Piter Jelles gymnasium. One of them, Kobus Sijbrandi, has even been involved with this thesis specifically, by being the first to attempt to translate the abstract into Frisian.

On a more personal note, a special *thank you* goes to Jenny Batson. When my thesis was getting the better of me and I started questioning my own sanity, she noticed. For me, having a cup of tea with her in her office will mark the point from which things progressed positively again.

Speaking of offices and sanity, I hope my office mates, wherever they are, have still renounced office chairs in favor of exercise balls. Giovanni Cina, Malvin Gattinger, and Dieuwke Hupkes, I am very happy to have had you as office mates. Likewise, I am happy to have had Frederik Lauridsen in an office nearby. His PhD program was just ahead of mine, so by watching Frederik closely, I often knew what awaited me.

Outside the academic environment, Martijn de Niet has been a great help by reminding me that the modern world is shaped by human desire and perceived need. Applying this political view to research may not be optimistic, but it did make it easier for me to untangle myself from my research emotionally.

Much of my life outside the academic environment revolved around kayaking and I am grateful for my wonderful friends at the Rotterdamsche Cano Club. Particular thanks go to Jaap Kraaijenga for his remarkable ability to sense what is going on in peoples lives.

With regards to kayaking, I am deeply indebted to my K2 partners, Koen van Ginkel, Arthur Hamel, Alexandra van den Elsen, and Erwin Verkleij. Training and racing with them has given each of my years as a PhD student its own character. The most memorable races were team efforts, and would not have been possible without the amazing supporting job done by Ron Hoogwater and Hans Borsboom. I am delighted to have them as my paranymphs for the defense of this thesis.

Jouke Witteveen
Leiden, December 2019

Chapter 1
Introduction

No matter how powerful a machine, it may not be powerful enough. This is true of machines that do physical work, but also of machines that compute, as every computer will struggle on some workloads. When a computational task gets challenging, it is said to be complex. This is not very precise and indeed, the word *complexity* is used to indicate a wide variety of phenomena.

This thesis introduces a unified framework for the analysis of a multitude of forms of complexity. A unifying theory need not be better than any of the specific theories it tries to include. However, it may open the door to new ways of thinking about old concepts and thus enable future developments.

We begin this introductory chapter with an informal section, arguing that there is really no single form of complexity. This section, Section 1.1, is intended to be readable by a broad audience and is centered around intuition more than around mathematics. In Section 1.2, we take a more in-depth look at some of the ways complexity pops up in mathematics and computer science. The notions of complexity that emerge shall each be subject to an analysis in our unified framework in Chapter 3. Our unified framework offers a mathematical model of complexity rooted in a branch of computer science called *parameterized complexity theory*. We shall review the existing formalisms in parameterized complexity theory in Section 1.3. A short overview of the contributions presented in this thesis is given in Section 1.4.

Following this introductory chapter is a chapter that contains the background theory required for our analysis of multiple forms of complexity. This background chapter contains two parts. The first part, Section 2.1, deals with established theory and serves to make this thesis self-contained. In the second part, Section 2.2, our new framework for the analysis of complexity is laid out.

The body of this thesis is formed by Chapter 3, which contains all of our results. It is split into five parts, each dealing with a different form of complexity. A high-level overview of our results in a shared context is given in Chapter 4.

1.1 The Size of a Cube

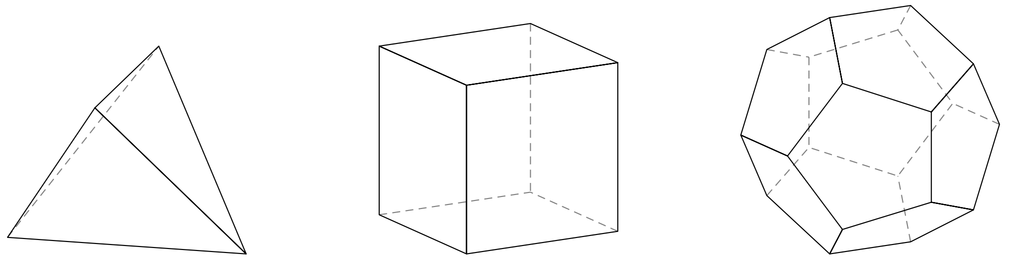
Shown in Figure 1.1 are three different shapes. When we ask which of these three shapes is the largest, we start a journey down a deep rabbit hole. In this thesis, that rabbit hole is explored. As an indication of what is to come, we briefly consider a comparable question: What is the largest species of snakes? The heaviest snakes are anacondas, but the longest snakes are pythons. Thus, the answer to this question depends on what is meant by “the largest”.

For our shapes, the situation is no better. Especially since the figure only shows the abstract essence of a tetrahedron, a cube, and a dodecahedron. Were the figure to show any particular physical instances of these shapes, then at least we could have compared masses and lengths. Luckily, abstract shapes can be measured too. The number of corners, the number of edges, and the number of faces are a few of the metrics we can look at. In all of these measures, a cube, Figure 1.1b, is larger than a tetrahedron, Figure 1.1a, and smaller than a dodecahedron, Figure 1.1c.

	tetrahedron	*cube*	*dodecahedron*
corners	4	8	20
edges	6	12	30
faces	4	6	12

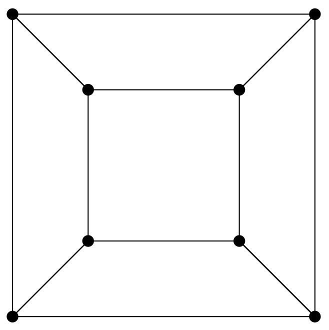
None of these ways of measuring a shape, by the number of corners, edges, or faces, is clearly more fundamental than the others. Besides, these ways of measuring are by no means all ways there are of measuring a shape. Another metric looks at the number of edges that need to be traversed in order to get from one corner to another. Let us consider this metric for the cube. From each corner of the cube, only 3 others are directly reachable. Sometimes, traversing 2 or even 3 edges may be necessary. Since we never need to go over more than 3 edges, we say that the *diameter* of a cube, in an abstract sense, is 3 edges. The diameter of a shape is another metric that may be used to measure a shape.

Yet another metric is defined by the minimum number of corners we need to mark so that every edge connects to a marked corner at one or both of its ends.



(a) A tetrahedron (b) A cube (c) A dodecahedron

Figure 1.1: Three shapes


(a) The abstract cube visualized as a graph. Corners of the cube are shown as dots and dots are connected by a line whenever the corners they represent are connected by an edge of the cube.

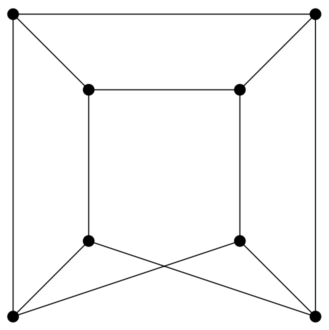

(b) The Wagner graph. Note that this graph can be obtained from the cube graph, Figure 1.2a, by exchanging two endpoints of two edges.

Figure 1.2: Structures of possibly connected objects can be represented as graphs. The dots in the above graphs represent the objects and are called *vertices*. The lines connecting vertices are called *edges*.

Such a collection of marked vertices is said to *cover* the edges. For a cube, it is possible to cover the edges using 4 corners. It is not possible to cover all the edges with fewer corners, thus 4 is the minimum number of corners that is required to cover the edges. Like our previous metrics, this minimum number of corners that is required to cover the edges of a shape may be used to measure a shape.

Which metric is the most relevant depends on what uses of our shape we are most interested in. Our previous two metrics looked specifically at corners and edges. In such cases, we are really only interested in the connection structure of a shape. Its three-dimensional nature is of lesser interest and the cube may as well be represented as in Figure 1.2a.

Note that for all simple graphs, there is a maximum to the number of edges as a function of the number of vertices. Once all vertices in a graph are connected to each other, we cannot add any new edges to the graph. On the other hand, if we allow for vertices that are not connected to any other vertex, then the number of vertices can be far greater than the number of edges. In particular, there is then no maximum to the number of vertices as a function of the number of edges. Moreover, there are only finitely many graphs with any given number of vertices, whereas there are infinitely many for any given number of edges. For this reason, we may consider the number of vertices to be a more fundamental metric than the number of edges. However, calling a graph with very many vertices but hardly any edges “large” may conflict with our intuitive notion of size.

Again, what graphs are large depends on what is meant by “large”. In each context the appropriate notion of size may be different. Roughly speaking, the size of an object relates to how difficult it is to work with. The more extreme the dimensions or masses of objects are, the more inconvenient processing them will likely be. This is true for the physical processing of physical objects, but also for processing abstract objects computationally. It is the latter kind of processing that this thesis is concerned with.

Returning to graphs, some actions on graphs may conceivably be easier on graphs with a smaller diameter. In the context of such actions, the diameter of a graph may be a good measure of the size of a graph. Other contexts may favor graphs in which the edges can be covered by a smaller number of vertices. A set of vertices that cover the edges of a graph is called a *vertex cover*. In contexts that favor small vertex covers, the minimum size of a vertex cover may be a good measure of the size of a graph.

As a demonstration of the possible disagreement between all these measures of the size of a graph, consider the graph in Figure 1.2b. This graph is known as the *Wagner graph*. Because it cannot be drawn without some edges crossing each other, there is no meaningful way we can assign a value to the number of faces of this graph. Of course, there are also no corners in the way that our shapes had them, but the Wagner graph does have vertices, which can be counted just as well. As far as the number of vertices or the number of edges are concerned, the Wagner graph, Figure 1.2b, has the same size as the graph of the cube, Figure 1.2a.

	cube graph	*Wagner graph*
vertices	8	8
edges	12	12
diameter	3	2
vertex cover	4	5

On our other two metrics, the graphs differ. The Wagner graph has a smaller diameter than the cube graph. By contrast, the minimum size of a vertex cover, listed simply as "*vertex cover*" in the table above, is smaller for the cube graph. Thus, we see that which of the two graphs is larger depends on the metric that is used and hence on the context in which we compare the two graphs.

In the next section of this chapter, Section 1.2, several more examples of the multifaceted nature of complexity are discussed more formally. The remainder of this thesis then works towards a unified analysis of complexity that works for each of the forms of complexity we identify in Section 1.2. Finally, in Section 4.1, we shall return briefly to measuring a cube, and summarize our findings.

1.2 Historical Encounters

The word *complexity* is used in various contexts and its meaning is not always made precise. Still, it has often been observed that structural properties of data may influence the notion of complexity at hand. In this thesis, we put forward a mathematical formalization of the notion of complexity, covering as many use cases as possible. First, let us review some of the ways we may encounter complexity and what structural properties these many forms of complexity are involved with. In relation to their notions of complexity, each of these structural properties points at a line of thinking that is essentially parameterized. In Chapter 3, this parameterized nature is made explicit by rephrasing the corresponding results in a unified parameterized framework. A historical overview of the theoretical side of parameterized reasoning can be found in Section 4.2.

1.2.1 Computability

Some sets are intrinsically undecidable. For such sets, there is no effective procedure that is able to distinguish the members of the set from the nonmembers of the set. This is a very extreme form of complexity: We are unable to determine membership in undecidable sets uniformly not because we are perhaps not clever enough, but because it is fundamentally impossible. Undecidable sets, however, are not all equally undecidable. It is possible to organize many sets in a hierarchy of (un)decidability. In fact, there are many ways to do so.

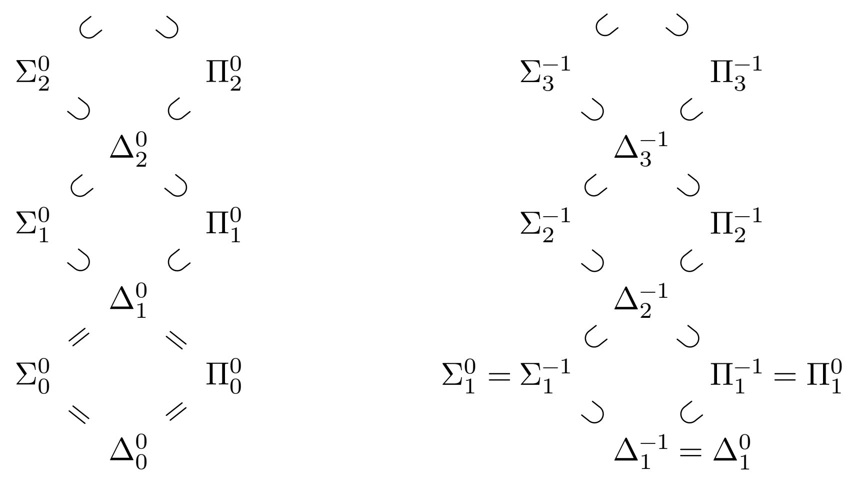
One hierarchy in which sets can be placed based on their decidability is the *arithmetical hierarchy*, due to Kleene [91]. This hierarchy consists of classes of sets that are defined inductively. The classifications are denoted Σ^0_n and Π^0_n, where n is a natural number. A set S is in Σ^0_{n+1} if there is a set P in Π^0_n such that we have

$$S = \{x \mid \exists y\colon (x, y) \in P\}.$$

Observe that the set P is a set of pairs. Conversely, a set P is in Π^0_{n+1} if there is a set of pairs S in Σ^0_n such that we have

$$P = \{x \mid \forall y\colon (x, y) \in S\}.$$

All that remains to define the arithmetical hierarchy is a definition of the base case, the classes Σ^0_0 and Π^0_0. We follow Rogers [131] and Downey and Hirschfeldt [46] and set both classes equal to the class of decidable sets. Originally, Kleene chose a smaller class, namely that of sets definable through “primitive recursive” predicates. Yet another option is to take for Σ^0_0 and Π^0_0 the class of sets definable in first-order logic with only bounded quantifiers [115]. We shall not go into details about these alternatives. Whichever definition we adhere to, we end up with the same classes Σ^0_n and Π^0_n, whenever n is at least 1. To wit, regardless of our choice for Σ^0_0 and Π^0_0, a set is in Σ^0_1 precisely when it is *semidecidable* (also known as *recursively*



(a) The arithmetical hierarchy

(b) The difference hierarchy lies below the Δ^0_2 level of the arithmetical hierarchy.

Figure 1.3: The arithmetical hierarchy and the difference hierarchy are both infinite hierarchies of classes of sets.

enumerable) [91, 115, 131]. Likewise, the complement of a semidecidable set is in Π^0_1, as $\exists$ and $\forall$ are dual to each other in the sense that, for every predicate Q we have

$$\neg\exists x\colon Q(x) \iff \forall x\colon \neg Q(x).$$

Indeed, a set S is in Σ^0_n precisely when there is a decidable set A such that we have

$$S = \{x \mid \exists y_1\colon \forall y_2\colon \exists y_3\colon\ \dots\ \colon ((\cdots((x, y_1), y_2)\cdots, y_{n-1}), y_n) \in A\}, \tag{1.1}$$

and similarly for Π^0_n when we exchange $\exists$ and $\forall$. It follows that, in general, a set is in Σ^0_n if its complement is in Π^0_n. As, whenever n is at least 1, the classes Σ^0_n and Π^0_n are unequal, neither is closed with respect to taking complements. For convenience, we therefore define classes Δ^0_n as the intersection of Σ^0_n and Π^0_n. These classes *are* closed with respect to taking complements. Remark that sets in Δ^0_1 are both semidecidable and have a semidecidable complement, and are therefore decidable. Visually, the arithmetical hierarchy thus looks like depicted in Figure 1.3a.

Returning to undecidability as a form of complexity, the arithmetical hierarchy provides a means of analyzing complexity. For sets that appear in one of the classes of the arithmetical hierarchy, say Δ^0_n, we may call the level at which it occurs, n, "the complexity" of the set. Thus, we can compare how undecidable certain sets are. As demonstrated by (1.1), this notion of complexity ties in with a more or less structural property of the set it applies to. The number of quantifier alternations required for defining a set starting from a decidable set expresses its complexity.

At a conceptual level, one of the focal points of computability theory is the behavior of computations in the limit. Computations that terminate eventually are said to *converge*, and computations that never terminate are said to *diverge*. From this perspective, it is clear how the sets in Σ_1^0, the semidecidable sets, are slightly more complex than the decidable sets. Decidable sets have decision procedures and those procedures converge on all inputs. For semidecidable sets on the other hand, convergence of a procedure that tries to determine membership can only be guaranteed on the members of the set. Ascending further in the arithmetical hierarchy, it becomes near-impossible to give characterizations of the sets in terms of convergence. This is one reason to look for measures of complexity-beyond-decidability that increase complexity more gradually. One such measure was introduced by Ershov [52] some 25 years after the introduction of the arithmetical hierarchy. This measure also revolves around a hierarchy of classes of sets and is in that regard similar in spirit to the arithmetical hierarchy. We have seen that the Σ and Π classes of the arithmetical hierarchy are not closed with respect to taking complements. The starting point of Ershov's hierarchy was the observation that these classes are also not closed with respect to taking relative complements. Given two sets A and B in some class Σ_n^0, with n at least 1, the set $A \setminus B$ need not be in Σ_n^0, and similarly for the classes Π_n^0. However, when A and B are taken from Σ_n^0, we may go about deciding whether a given x is in $A \setminus B$ as follows.

1: We assume x is neither in A nor in B and if our computations are interrupted and we are asked our best guess about x, we answer that it *is not* in $A \setminus B$.

2: We try to find out whether x is in A by running a procedure that halts precisely on the members of A. If this procedure halts, we have learned that x is in A. In case we are then asked our best guess about x, we answer that it *is* in $A \setminus B$.

3: Having found that x is in A, we try to find out whether x is in B by running a procedure that halts precisely on the members of B. If this procedure halts, we have learned that x is also in B. In case we are later asked our best guess about x, we know the correct response and answer that it *is not* in $A \setminus B$.

Only in the final situation, we can answer a membership query with certainty. Therefore, our procedure should be considered a kind of approximation of a decision procedure. Observe that the procedure adjusts its knowledge about membership of x in $A \setminus B$ at most twice. Procedures that are not convergent in the classical sense, but are allowed to "change their mind" a finite number of times were first put forward by Putnam [124] and Gold [66].

The idea of Ershov was to base a hierarchy on the number of times a decision procedure in the sense of Putnam and Gold is allowed to change its mind. The resulting hierarchy is known today as the *difference hierarchy* [46, 140]. The classes of the difference hierarchy are denoted Σ_n^{-1} and Π_n^{-1}. For sets A and B, let $A \triangle B$ denote the *symmetric difference* $(A \setminus B) \cup (B \setminus A)$. A set S is in Σ_n^{-1} if there are sets $A_1, A_2, A_3, \ldots, A_n$ in Σ_1^0 such that we have

$$S = A_1 \triangle A_2 \triangle A_3 \triangle \cdots \triangle A_n. \tag{1.2}$$

Like in the arithmetical hierarchy, a set P is in Π_n^{-1} if its complement, P^{C}, is in Σ_n^{-1}. Note that when n is odd, we have

$$(A_1 \triangle A_2 \triangle A_3 \triangle \cdots \triangle A_n)^{\mathrm{C}} = A_1^{\mathrm{C}} \triangle A_2^{\mathrm{C}} \triangle A_3^{\mathrm{C}} \triangle \cdots \triangle A_n^{\mathrm{C}}.$$

Therefore, for odd n, a set P is in Π_n^{-1} if there are sets $A_1, A_2, A_3, \ldots, A_n$ in Π_1^0 such that we have

$$P = A_1 \triangle A_2 \triangle A_3 \triangle \cdots \triangle A_n.$$

These definitions are equivalent to the original characterization by Ershov [52] that was stated in terms of unions of disjoint relative complements. The equivalence follows from elementary identities. The resulting hierarchy, including the levels Δ_n^{-1} that are the intersection of Σ_n^{-1} and Π_n^{-1}, is depicted in Figure 1.3b.

Because the Δ_2^0 class is closed under taking unions, intersections, and complements, it is also closed under taking symmetric differences. Moreover, it includes both Σ_1^0 and Π_1^0, thus for all natural numbers n, the classes Σ_n^{-1}, Π_n^{-1}, and Δ_n^{-1} of the difference hierarchy are included in Δ_2^0. Consequently, the difference hierarchy is only relevant for sets that are not too complicated according to the arithmetical hierarchy. Still, its core ideas make the ensuing notion of complexity an interesting measure of undecidability. As we have seen, the number of semidecidable sets needed to write a set like in (1.2) relates to a generalized form of convergence of computation.

1.2.2 Computational Tractability

In computational complexity theory, a set is deemed complex if it is intractable. Here, tractability is customarily taken with respect to the running-time behavior of decision procedures. While we can measure the time a decision procedure takes on a specific input, we are primarily interested in how this time compares to other, unknown, inputs. The running time of a decision procedure is therefore traditionally expressed as a function of the size of its input. As we saw in Section 1.1, the notion of input size is heavily reliant on the choice of an encoding scheme for inputs. No computable encoding scheme reflects all possible structural aspects an input may have. This is especially visible in contexts that offer some "standard" encoding. For instance, in graph theory [40], graphs are often thought of as encoded by an adjacency matrix. In that case, the size of a graph is determined by the number of its vertices. However, certain structural properties other than the number of vertices may be exploitable by a decision procedure for a set of graphs. Regarding sets that are intractable because the running time of a decision procedure is at least exponential as a function of the input size, Garey and Johnson remark the following.

> [...] there are a variety of ways in which the time complexity of an algorithm can be "exponential," some of which might be preferable to others. This is especially evident when, as is customary in practice, we consider time complexity expressed in terms of natural problem parameters instead of the artificially constructed "input length." [63, Section 4.3]

When restricting to inputs on which such *natural parameters* assume small values, a set that is initially intractable may indeed become tractable. As inputs with low parameter values may be abundant, it may be that large subsets of the input space are easy to digest for some decision procedure. In addition, Garey and Johnson note that inputs encountered in practice may tend to have low parameter values: "[...] in practice it is often the subproblem, rather than the general problem, that we are called upon to solve."

1.2.1. Example. The standard compilers for the Rust programming language and the Haskell programming language are able to infer data types of variables and functions. The algorithm they use for this has a worst-case running-time that scales exponential as a function of the length of its input. However, such exponential running times are experienced very rarely in practice [89]. Thus, some property of the expressions that this algorithm operates on behaves in a special way for inputs that are encountered in practice. Indeed, the origin of the exponential running time of the algorithm can be traced back to the nesting depth of a certain pattern in the expressions. In practice, this nesting depth is almost always low.

1.2.2. Example. We can model a social network by a graph, representing persons by vertices and connecting two vertices when the persons they represent know each other. Given a graph that models mutual acquaintance, we may ask how large a subset of people there exists that all know each other [a *clique*, see 40]. This is formalized by the set

$$\begin{aligned}\textsc{Clique} = \{(G, l) \mid{} & \text{there is a set of at least } l \text{ vertices of the graph } G \text{ in which} \\ & \text{each pair of vertices is connected by an edge}\},\end{aligned}$$

Let G be a graph with n vertices and let l be at most n. The number of possible subsets of the vertices of G that are of size l cannot be bounded by a polynomial of n alone. In fact, when l is, say, $\frac{n}{2}$, the number of subsets is exponential as a function of n. It is widely believed that there are no decision procedures for Clique that have a subexponential running time. Nonetheless, given a subset of vertices of G, we can check whether its members are pairwise connected within a polynomial running time.

However, this analysis disregards some practicalities that result from the situation we are modeling. It is rare to come across a person with very many mutual acquaintances. Therefore, it is safe to assume that in any graph we that models a social network, almost no vertex is connected to more than, say, 20 others. As a consequence, we find that in any graph encountered in practice, a set of pairwise connected vertices, a *clique*, can contain at most 21 elements. This means that the number of subsets of vertices that needs to be checked is less than n^{21}, which, as Garey and Johnson observe [63, Section 4.1], is polynomial in n. While this already brings the running time down to polynomial, we shall see in Section 1.3 that a far more efficient decision procedure is not hard to come by.

The takeaway is that computational complexity is preferably measured by parameters of the input instead of by the input length alone. Somehow, we need to decide which parameters to take into account. Different decision procedures may favor different parameters. Note, though, that it is possible to combine the benefits that different parameters may provide through aggregating multiple decision procedures.

1.2.3. Example. One way to decide membership in Clique is by generating an exhaustive list of candidate sets of vertices and checking whether any is a clique of the desired size. Suppose we want to decide whether a given graph G has a clique of l elements. The simplest implementation of the aforementioned design pattern starts out by generating all possible sets of l vertices of G. However, other approaches are possible too.

Suppose l is even and we have a clique $(v_1, v_2, v_3, \dots, v_l)$. Because these vertices form a clique there is, for $i \leq \frac{l}{2}$, an edge connecting v_i to $v_{i+\frac{l}{2}}$ in G. Thus an alternative way to generate candidate sets of vertices is via all possible sets of $\frac{l}{2}$ edges, by taking the endpoints of the selected edges. Any clique of size l is guaranteed to be generated this way.

We can compare these two approaches by counting the number of candidate sets they consider. Let n be the number of vertices in G and m the number of edges in G. There are $\binom{n}{l}$ sets of l vertices and $\binom{m}{l/2}$ sets of $\frac{l}{2}$ edges. When either of these numbers is much smaller than the other, we expect the corresponding decision procedure to run much faster than the other. Thus, it pays to construct an aggregate decision procedure. This decision procedure would first compute both numbers and then run the decision procedure that is expected to be the fastest.

The computational complexity of the vertex-centric decision procedure is best expressed as a function of the parameters n and l. On the other hand, the complexity of the edge-centric decision procedure is best expressed as a function of the parameters m and l. Our aggregate decision procedure demonstrates that it pays off to take into account multiple parameters. By taking into account more parameters, we get a more detailed insight into the computational complexity of a set such as CLIQUE. For some instances of CLIQUE, the running-time bound available in terms of n and l is better than that in terms of m and l and vice versa.

The algorithm we presented for our aggregate decision procedure for CLIQUE in the example above combines two other algorithms. Such algorithms are known as *hybrid algorithms* [103]. They offer a best-of-both-worlds alternative in situations where we have two algorithms without one being better than the other. From a parameterized point of view, hybrid algorithms combine the information of multiple input parameters and as such show how these parameters can interact. This is relevant not only when we use parameters to measure computational complexity, but also in the design of ever-faster polynomial-time algorithms.

1.2.4. EXAMPLE. As a showcase of hybrid algorithms that do not face computational intractability, we shall take a look at sorting algorithms. Even naive sorting algorithms, such as repeatedly finding the least value, manage to have a running time bounded by a polynomial of the length of the input list. Because sorting is such a common operation in algorithmics, extensive research has been put into the development of fast sorting algorithms.

The quicksort algorithm [35] makes a number of comparisons that is at most quadratic as a function of the length, n, of the input list. However, worst-case inputs are very rare and on average the algorithm needs roughly $n \log n$ comparisons. With a number of comparisons in the order of $n \log n$ in the worst case, the heapsort algorithm [35] appears at least as good as quicksort. Nevertheless, it has a higher overhead and in practice it is often outperformed by quicksort. By combining both algorithms, we can construct a sorting algorithm that is about as fast as quicksort, yet has a worst-case running time in the order of $n \log n$. This hybrid algorithm is known as *introsort* [111] and is used by some prominent implementations of the C++ standard library.

The way introsort chooses between its constituent algorithms, quicksort and heapsort, is not as up-front as it was in our hybrid decision procedure for CLIQUE. This matters when we want to identify parameters that express structure favored by either of the constituent algorithms. Instead of handing over the input to either quicksort or heapsort, the hybrid introsort hooks into the divide-and-conquer nature of quicksort. The core of the quicksort algorithm is that it splits its input list into a list of low values and a list of high values. These sublists can be sorted independently and for that, quicksort recursively invokes itself. Because the splitting stage requires some n comparisons for a list of n elements, we do not want the recursion depth to exceed a constant multiple of $\log n$. In the worst case, however, quicksort reaches a recursion depth of n. To mitigate this worst case behavior, introsort uses heapsort to sort the sublists when the recursion depth gets too high, say higher than $2 \log n$. Thus, we have found that the recursion depth reached by quicksort functions as a parameter of the input list. This parameter may appear somewhat unnatural and its specific value for a given list depends on the implementation of quicksort. For all implementations, however, input lists that get high parameter values can be constructed [105].

For short lists, insertion sort [35] is faster still than quicksort and heapsort. However, it makes a quadratic number of comparisons in the average case. Therefore it is suboptimal for all inputs of a length exceeding some platform-dependent constant. Many implementations of introsort use insertion sort when the input list is short. Here, the parameter at play is evident. It is the length of the input list.

The architecture of introsort follows quicksort and only for the base case of the recursion it deviates from that. In this way, introsort is based on an algorithm that has a bad worst-case running time, but is fast in practice. Another hybrid sorting algorithm, timsort [121], takes the opposite approach. This algorithm was developed for the Python programming language and is also used in the Java virtual machine and in the standard library of the Rust programming language. It is adapted from mergesort [35], which has a good worst-case running time, but fails to exploit patterns encountered in practice. Like quicksort, mergesort is a divide-and-conquer algorithm. Where introsort followed quicksort top-down and resorted to heapsort at high recursion depths, timsort follows a bottom-up implementation of mergesort. It uses insertion sort on small chunks of the input to quickly prepare a base case that is as accommodating to mergesort as possible. Using insertion sort for the base case is efficient, because insertion sort is fast on short lists. The base case is prepared in such a way that the use of locally sorted parts of the input list is maximized. This makes timsort an *adaptive* sorting algorithm. A proper analysis of timsort is therefore not possible on the basis of only the length of the input list as a parameter. In addition, it requires a parameter related to the locality of the disorder in the input list.

Note that a detailed analysis of the computational complexity of an algorithm may expose many parameters that influence the running time of the algorithm. These parameters need not be independent of one another, and their interactions need not be straightforward. Parameterized computational complexity theorists are tasked with gaining insight in these interactions.

1.2.3 Algorithmic Complexity

Complexity, in whatever incarnation, can often be traced back to a lack of structure. Objects that are evidently highly structured are easy to analyze and an unlikely cause of complexity. But what makes an object structured? One answer could be “if it has many properties”, but this simply makes us wonder what counts as a property of an object. Certainly, we want the properties of an object to tell us something about the object. In other words, it must be special for an object to have a given property. We mean this in a technical sense and assert that structural properties help in giving short descriptions of objects.

1.2.5. Example. Consider graphs on 8 vertices. There are

$$2^{\binom{8}{2}} = 268\,435\,456$$

such graphs. However, if we know that a particular graph has 9 edges, we are left with only

$$\binom{\binom{8}{2}}{9} = 6\,906\,900$$

options. Because this is a substantial reduction, we can give a succinct description of a graph with 8 vertices and 9 edges by first giving its number of edges.

The critical observer will have noticed that we have assumed a number of vertices that was fixed and known. Indeed, having 8 vertices is in itself already a structural property of a graph. Without knowing the number of vertices in a graph that is being described to us, we have an infinite number of candidates to choose from.

Often, we can identify a value inside a structural property that can be generalized over. The property can then be seen as an instantiation of an underlying property schema. This is most apparent when the structural property has an element of counting in it, such as is the case with our "has 9 edges" property in the example above.

1.2.6. Example (continued). Replacing the value 9 in the property "has 9 edges" by the variable e, we obtain the property schema "has e edges".

Seldomly is it necessary to make a distinction between properties and property schemata. However, it is useful here because it clarifies the relation between parameters, as encountered earlier, and properties. Technically, parameters correspond to property schemata.

When describing objects, it is necessary to settle, beforehand, what counts as a reasonable description method. Were we to allow all possible mappings of objects to strings, we run into paradoxes like the Berry paradox [see 100]. This paradox shows that if any mapping goes, any object could be described in arbitrarily few bits. The argument is simple: if it would not be so, then we could take the first among the counterexamples and map an arbitrarily short description to it, violating it being a counterexample. Therefore, we prefer to work with computable total functions as description methods. Each description method is thus assigned a length according to a specification of the performed computation. Doing so, we get, for each object, a lower bound on the sum of the length of the description method used and the length of a description of the object. This bound is the Kolmogorov complexity of the object. Indeed, any structure in an object is picked up by its Kolmogorov complexity. This urges an investigation of the use of the Kolmogorov complexity as a parameter, as the property schema "has a Kolmogorov complexity of at most c". However, the Kolmogorov complexity is not itself a computable function, so such an investigation would take us outside the realm of reasonable description methods.

Next to thinking of Kolmogorov complexity as a parameter itself, we may think of Kolmogorov complexity as emerging from an assembly of all parameters. A property can be thought of as a set of objects, namely the objects that have that property. Any object that has a property can then be specified by its index in this set. This interpretation of properties was already present in our example with the "has 9 edges" property. From this interpretation, it is a small step up to a kind of meta-property, the property schema "has property q". As before, we want our description method to adhere to a form of computability. We therefore assume that a property q is specified in the form of a decision procedure for the set of objects that have property q. These decision procedures are computable total functions. However, there is no decision procedure that recognizes precisely the computable total functions. Therefore, whether any given q represents a reasonable property in the sense that it corresponds to a computable total function cannot be decided uniformly.

Alternatively, the undecidability of our meta-property can be deduced from the observation that this property schema too relates to Kolmogorov complexity. An object represented by some x is the only object that has the property "equals x". It should be clear that the shortest specification of this property has a length equal to the Kolmogorov complexity of x. Moreover, the index of x in the set of objects that adhere to this property is 1. Hence, the length of a description according to this property added to the length of this property is about equal to the Kolmogorov complexity of x. At the same time, no two-part code can give description of x of a length shorter than the Kolmogorov complexity of x. Hence our parameter, the property schema "has property q", is also closely related to Kolmogorov complexity.

The analysis of Kolmogorov complexity as emerging from an assembly of all parameters has brought us one abstraction level up in the analysis of parameters. Instead of working with individual parameters, we now work with sets of, not necessarily related, parameters. When a certain set of parameters has our interest, we may ask how good of an approximation to the Kolmogorov complexity the ensuing two-part code is. Of course, we should also ask whether or not the set of parameters is in any way decidable. Taking the abstraction even further, we may look at ways to define sets of parameters. For instance, given a computationally hard decision problem, we may consider those parameters that are responsible for the computational hardness. This then leads to a notion of complexity in its own right. If two decision problems lead to the same set of parameters, we could say that the computational complexity of the two problems has the same origins. While exciting, this notion of equivalent computational complexity through identical sets of parameters has not seen much attention yet.

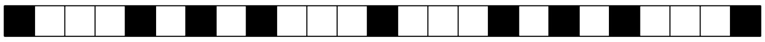
1.2.4 Algorithmic Statistics

A picture is worth a thousand words, or, in more convoluted terms, not all media are equal when it comes to the presentation of information. This sets information apart from computability, which remains unchanged on a large class of machine models. In the landmark paper in which he introduced his machine model, Turing wrote the following about the use of one dimension or two dimensions in *computation.*

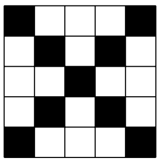
> In elementary arithmetic the two-dimensional character of the paper is sometimes used. But such a use is always avoidable, and I think that it will be agreed that the two-dimensional character of paper is no essential of computation. I assume then that the computation is carried out on one-dimensional paper, i.e. on a tape divided into squares. [144]

A similar statement is arguably not true of *information.* In the presentation of information, using one dimension or two dimensions makes a difference.

1.2.7. Example. Consider the following one-dimensional sequence of squares, some of which are filled.



We can spot pretty quickly that this sequence is a palindrome. However, a far stronger geometrical relationship between the filled squares becomes evident when we present the same sequence as a two-dimensional grid.



This goes to show that certain modes of presentation are preferred for certain patterns. Not all information is equally accessible in all modes of presentation.

For a more thorough treatment of informativeness, we need a formal description of what we mean by information. Typically, we link information to *entropy* [36], thus bringing statistics on board the study of informativeness. Information, then, relates to a property of a data source that expresses how unpredictable the data source is. In other words, information is what breaks the mold, as it defies our expectation. We remark that there is more to information than entropy alone, but refer the interested reader to the philosophical overview by Adriaans and Van Benthem [3].

Different kinds of data come with different expectations. Some patters are more clearly presented in certain modes of presentation, and conversely some modes of presentation make us expect certain patterns. A musical idea, for example, is hard to convey in a painting, yet may be precisely what an audience member in a concert is listening for. The medium is thus part of the message and information is context-dependent.

1.2.8. Example. Let us take a look at the distribution of passwords that are made up of four digits. Four-digit passwords are the de facto norm for a personal identification number, a PIN. A naive analysis would be that when we have no information about someone's PIN and we have to guess it, we stand a one in ten thousand chance of being correct. As we shall see, this is far from the truth when the PIN was chosen freely by its owner.

To estimate the distribution of PINs in use, we filter a collection of over half a billion passwords for four-digit passwords. The collection is made up of passwords obtained in data breaches and was put together by Hunt [85]. This tactic was first employed by Berry [20] and our approach here is much the same as his. In the database we use, 14 830 591 uses of a four-digit password are recorded. Based on the relative frequency of each PIN, we quickly find that they are not distributed uniformly. As can be seen in Figure 1.4, the bulk of the distribution of PINs can be approximated by a straight line in a log–log plot. Therefore, our PINs are an example of a dataset that adheres to Zipf's law. This empirical law states that the frequency of a word in a corpus of natural language is inversely proportional to its rank. Outside natural language, distributions of this kind have been observed in many real-world data encountered in the physical and social sciences [112].

Having found that PINs are not distributed uniformly, we wonder what can be said about the specifics of their distribution. In Figure 1.4, we cannot see which PINs correspond to which data points. Therefore, we turn to a different plot, depicting the rank of all PINs as estimated by our approximation of the empirical probability distribution, Figure 1.5.

In Figure 1.5, we can recognize many categories of PINs that occur frequently. Some of these categories are purely syntactical. The prominent diagonal line, for instance, represents PINs of the form xyxy. Algorithmic complexity theory is well-suited to deal with such patterns. We may therefore expect the universal distribution [100], though uncomputable, to be a good model for the distribution of PINs. For this to be true, all structure in Figure 1.5 must be essentially syntactical. However, there is also a strong vertical line of PINs of the form 19xy, continuing shortly into the 20xy pattern. The prevalence of such PINs can easily be explained

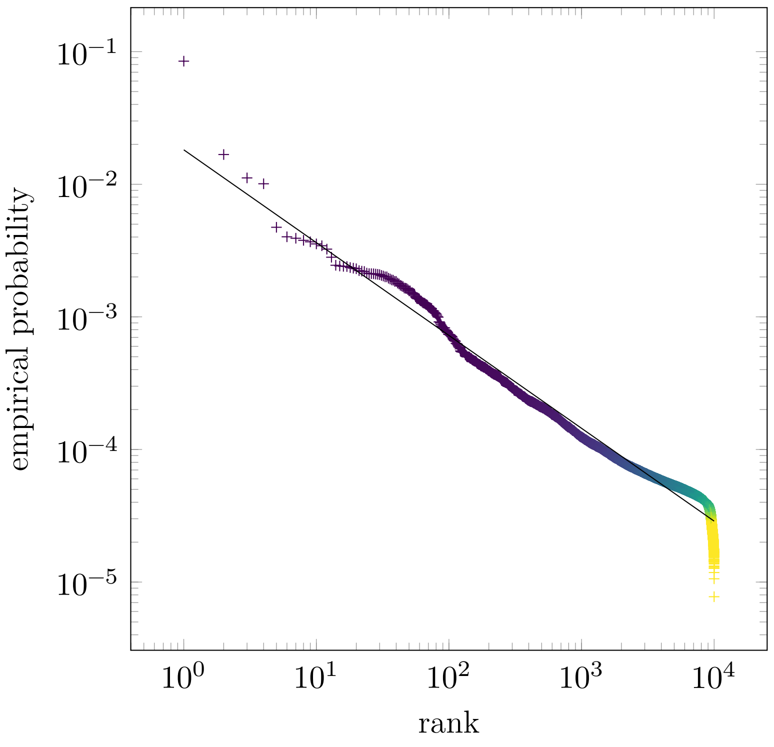


Figure 1.4: A log–log plot of the empirical probability of PINs by rank, in decreasing order of likelihood. Except for one outlier on the top-end, the PIN `1234`, and a thin tail of less than 5% of the PINs, the distribution can be approximated by a power law. Points are colored in accordance with their rank as approximated by the fitted power law. Because the horizontal axis is logarithmic, the colormap appears distorted.

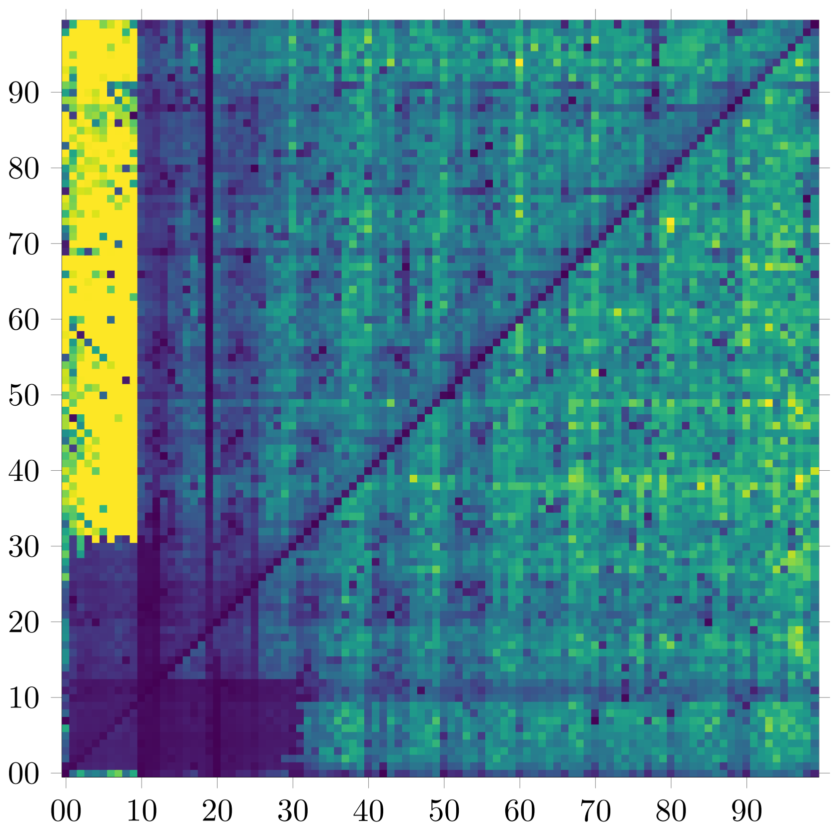


Figure 1.5: A heatmap of PINs, inspired by a similar heatmap created by Berry [20]. The first two digits of the PIN determine its horizontal position, while the last two digits determine the vertical position. This way, the PIN `0099` ends up at the top-left corner and the PIN `9900` ends up at the bottom-right corner. The colors are based on the clamped power law fitted on the bulk of the PINs in Figure 1.4. Darker colors represent more frequent occurrence of a PIN in the database. As witnessed by the area of light-colored PINs on the left side of the plot, many PINs in the tail of the distribution start with a `0`.

as people choosing a year as they PIN and most emotionally meaningful years being in the recent past. This explanation is not a syntactical one. Of course, a model for algorithmic complexity can include this contextual information at the cost of only an additive constant. With little more modeling of context, also the frequent occurrence of PINs formatted as dates, MMDD or DDMM, can be covered. The lengths of the months generate a distinct and recognizable pattern in our heatmap. Yet, we claim that the additive constant relative to a standard abstract definition of Turing machines will quickly become significant. Relative to the limited complexity of four-digit PINs, the cost of modeling all contexts that influence the choice of a PIN is huge. Some properties of the plot in Figure 1.5 can easily be explained culturally, but lack any numerical meaning. For instance, the PIN `5683` occurs particularly often, which can be explained by its meaning as a phoneword. A phoneword is a word spelled on a traditional phone keypad using ITU E.161 assigned letters. The letters of the word "Love" are assigned the digits `5683`, respectively. It is thus not reasonable to expect the distribution of PINs to approximate the universal distribution in any way. We are better off identifying as many meaningful categories of PINs and build our model of PINs from there.

We can try to model a multitude of contexts so that we can deal with the information in each of them. In doing so, however, we should control the amount of detail with which we describe our models. Surely, we do not want a tailor-made model for every conceivable arrangement of data. In such a setting, every data sample would have a context in which the sample contains no information. This is known as *overfitting* [72]. *Algorithmic statistics* addresses overfitting by taking into account the complexity of a description of a model. Interestingly, there are data sets for which no *simple* model is a decent fit. There are complex models in relation to which these data sets contain disproportionately little information. In relation to any simple model, the information in these data sets is considerably higher. From this observation, it follows that the minimum model complexity required to achieve a decent fit is a nontrivial property of a data set. Thus, a parameterized analysis of data sets, where the parameter is this minimum model complexity, emerges.

In traditional algorithmic statistics, Kolmogorov complexity is used in the assignment of complexity to models. By doing so, it is possible to include practically all effective models in the analysis of informativeness. As a result, the central probability distribution of interest in traditional algorithmic statistics is the universal distribution. We have seen in Example 1.2.8 that the universal distribution is not always the best for a particular domain. Fortunately, it is possible to employ the methods of algorithmic statistics in a setting where we restrict our attention to a specific collection of models.

1.2.5 Computational Redundancy

At the end of Moore's law, advances in computer engineering are no longer visible as more powerful computation, but instead as more ubiquitous computation. With this shift, the choice of *where* to perform computation, along with *when* to perform *how much* of it, becomes increasingly influential.

1.2.9. Example. There are many ways to store digital information, ranging from dynamic random access memory to magnetic tape and optical discs. Each storage medium has its own strengths and weaknesses. Storage can be classified according to different metrics, such as cost per bit, availability, or access speed. Typically, embedded volatile memory is fast, but expensive, whereas archived non-volatile memory is cheap, but slow. It is commonplace for applications to make use of different types of memory for different types of data, depending on the anticipated use. For a large database, cost is a primary concern, while the cache of a dynamic algorithm is best served by memory that is as fast as possible. In practice, this means the database may be stored on an off-site storage server, while the dynamic algorithm makes use of the CPU cache.

For computation, similar considerations can be made. Computing on a mobile device may provide very low latency, yet faces bottlenecks in terms of speed and energy consumption. The alternative would be to move computation to an external computer. This architecture, treating computation as a service, is nowadays known as cloud computing. Offloading computation becomes interesting when a computational workload is more demanding than the locally available computational power. One possibility is that the workload is inherently heavy, as is the case for rendering computer-generated imagery. Another possibility is that the local computational power is limited, as is the case when cross-compilation for embedded devices is used.

There is an interplay between the location where data is stored and the location where computation is performed. The two are best kept close together, and modern databases implement increasingly rich query languages [see also 4]. An extreme example is provided by the Apache Hadoop Distributed File System. Its manual phrases it as follows.

> A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. [28]

Note that a server may be highly capable as a data storage facility, yet be unimpressive in its computational abilities. The converse is possible as well. In both cases, we may still want to minimize the cost of communication with the server. For this, it becomes necessary to recognize redundancy in our computational task and leave it out of transmission.

1.2.10. EXAMPLE. Suppose we want to render many frames of an animation on a so-called "render farm". This requires the transmission of a description of the computational task for each frame. The animation may, however, contain duplicate frames. It is worthwhile to preprocess our data and remove all duplicate frames before invoking the render farm. This preprocessing operation does not require much computational effort from our side, yet it might reduce the amount of data transmitted substantially.

Conversely, if we want to perform a computation on data stored in an external database, it may be beneficial for the database to preprocess the data. To accomplish this, we not only send the database a request for certain data, but also a description of the operation we want to perform on it. The database may then spend a limited amount of time analyzing the data in terms of computational redundancy, and omit any identified redundancy from its reply. Again, with the proper time limit, this requires minimal computational effort from the database, yet may reduce the total amount of communication taking place.

When the computational context of data is known already when the data is recorded, it may be beneficial to perform the preprocessing step early on. Doing so may save the database some storage capacity, or at least bring the storage requirement more in line with the computational complexity of the data. As our database may not have great computational power, it is important that we limit preprocessing to computation that can be performed quickly.

With our examples so far, we have thought of machines connected via a network as atomic units of computation. However, also within a computer, there are many independent units capable of computing. These are the device controllers and other management systems that all run their own firmware. Traditionally, such systems are extremely limited in their computational power. This power has, however, been increasing steadily, and the potential of these devices to run application-level code has long been recognized [for example 126]. We have seen that preprocessing can reduce traffic on the network connecting computers. Similarly, preprocessing can reduce traffic on the system bus connecting components inside a computer.

For certain computational tasks, a useful separation into a computationally redundant and a computationally hard part may be possible. For others, it may be impossible to find such a clear-cut separation that is of any practical value. This does not mean that we cannot benefit from remote computational powers. Sometimes, a computational task is really a serialization of multiple subtasks, the specifics of each subtask depending on the outcome of the previous one. With such a compound task, we may be able to identify computational redundancy in each subtask as we get to it. In that case, making use of preprocessing for each of the subtasks requires a form of communication between the computational parties.

The party holding the data performs the preprocessing and communicates to the computationally powerful party the data relevant to the current subtask. The computationally powerful party in turn executes the subtask in order to find out the specifics of the next subtask. It then sends back these specifics, and the cycle repeats.

This protocol should be compared to the naive protocol, where all data is transmitted to the computationally powerful party. Thus, the worst-case is that we have to transmit all data, and any way to save on the amount of data transmitted should be considered an improvement. When each subtask requires only a small part of the data, the computational cost of preprocessing at the site of the data can really pay off.

With multi-stage preprocessing, computational redundancy has become a composite notion. It is no longer possible to trace a computation and pinpoint where preprocessing ends and the hard part of the computation begins. Instead, the two types of computation are interwoven, and computation switches back and forth between harvesting redundancy and essential computing. Computational redundancy is thus a multifaceted notion, and an investigation of computational redundancy must be multifaceted too. In particular, an analysis of computational redundancy should consider how the amount of data that is transmitted can be related to properties of the input. As we have seen multiple times now, properties can be expressed by parameters. Thus the analysis of computational redundancy becomes a parameterized analysis.

1.3 Parameterized Complexity Theory

Of the many contexts in which complexity may be encountered, one context has featured parameters most prominently. This is the context of computational complexity. In the formal development of a parameterized computational complexity theory, two dominant schools emerged around the turn of the century [44, 57]. However, the essential ingredients were already visible before that. We caught a glimpse of this in Section 1.2.2. Before we outline the two schools, their differences, and their similarities, we shall recapitulate some of the observations of Garey and Johnson [63].

Garey and Johnson

One of the classic **NP**-complete problems is the Partition problem. Here, we are given a finite sequence of numbers $x_1, x_2, \dots, x_m$, and are asked whether there exists a set $I \subseteq \{1, 2, \dots, m\}$ such that we have

$$\sum_{i \in I} x_i = \sum_{i \notin I} x_i.$$

Equivalently, this set I is such that we have $\sum_{i \in I} x_i = \frac{1}{2} \sum_{i \leq m} x_i$. Using dynamic programming, Garey and Johnson [63, Section 4.2] show that, for any constant c, the existence of a set $I \subseteq \{1, 2, \dots, m\}$ such that we have

$$\sum_{i \in I} x_i = c$$

can be determined within a time bound that is polynomial in m and c. Noting that we have

$$\frac{1}{2} \sum_{i \leq m} x_i \leq m \cdot \max\{x_1, x_2, \dots, x_m\},$$

they conclude that membership in Partition can be decided in a time polynomial in m and $\max\{x_1, x_2, \dots, x_m\}$. Algorithms with running times behaving this way are called *pseudo-polynomial time algorithms* by Garey and Johnson. It is also possible to express running times of pseudo-polynomial time algorithms in terms of input lengths. For this, we set n to the combined input length $|x_1| + |x_2| + \cdots + |x_m|$, and k to the maximum, $\max\{|x_1|, |x_2|, \dots, |x_m|\}$. From the previous observations it now follows that membership in Partition can be decided in a time that is polynomial in n and exponential in k. As the specific polynomial in n is independent of k, this is precisely the type of time bound that would later become known as *fixed-parameter tractable.*

Another problem Garey and Johnson look at is the **NP**-complete Clique problem, which we have seen before in Example 1.2.2. Suppose we have a graph G that has n vertices, none of which is connected to more than k others. As alluded to in Example 1.2.2, it is possible to determine the size of the largest clique in G in a time bound in the order of n^{k+1}. For a fixed value of n, this time bound is exponential in k. Likewise, for a fixed value of k, the time bound is polynomial in n. However, contrary to what we saw with PARTITION, the polynomial involved is not independent of k. Indeed, from these observations we cannot conclude that CLIQUE is fixed-parameter tractable with respect to the maximum vertex degree parameter. Nevertheless, running times of the form n^k are an object of study in modern parameterized computational complexity theory. We remark that CLIQUE *is* in fact fixed-parameter tractable with respect to the maximum vertex degree parameter [37, p. 10]: We can go through all vertices and, for each vertex, consider all possible subsets of neighbors. There are at most 2^k such subsets to consider per vertex.

Observations such as those about PARTITION and CLIQUE show that not all is lost when there is no polynomial-time algorithm for a problem of interest. Hard instances for the problem may be rare or, at least, there may be special cases of the problem that *can* be solved in polynomial time. Therefore, we want to include parameters in our analysis of computational complexity. The parameters we saw earlier, "maximum component length" and "maximum vertex degree", were, however, selected in a rather ad hoc fashion. While finding the right parameters may be an art [37, p. 12], a proper parameterized computational complexity theory needs to formalize the place and role of parameters.

Downey and Fellows

The first framework for parameterized computational complexity was given by Downey and Fellows [41, 44]. In their framework, a parameter is an explicit component of a problem instance and the problems considered are inherently parameterized. More specifically, a *parameterized decision problem* is a set of pairs $\langle x, k\rangle$, where k is a number called the *parameter*. A parameterized decision problem A is said to be *fixed-parameter tractable* if the time required to decide membership in A can be bounded like before. Specifically, for some function f and constant c, membership in A of a pair $\langle x, k\rangle$, where x is of length n, must be decidable in time $f(k) \cdot n^c$.

A renowned example of a parameterized decision problem is VERTEXCOVER. A vertex cover of an n-vertex graph G is a collection, C, of vertices from G such that every edge in G is incident to a vertex in C [40]. In the VERTEXCOVER problem, we are given a graph G and a number k, and are asked whether G has a vertex cover of at most k vertices. Like membership in PARTITION, membership in VERTEXCOVER can be decided in a time that is polynomial in n and exponential in k. Thus, deciding membership in VERTEXCOVER is fixed-parameter tractable. An easy way to see that this is the case is by considering the following recursive algorithm for deciding whether G has a vertex cover of at most k vertices.

```
1: We assert that $k$ is nonnegative, for otherwise $G$ cannot possibly have a
   vertex cover of at most $k$ vertices.
2: If there are no edges in $G$, then the empty set is a vertex cover of $G$ and it
   is guaranteed to contain at most $k$ vertices.
3: Else, let $v_1$ and $v_2$ be the endpoints of an edge in $G$. Any vertex cover of $G$
   must contain at least one of these vertices:
   3.1: If the graph obtained by removing $v_1$ from $G$ contains a vertex cover
        of at most $k - 1$ vertices, then $G$ contains a vertex cover of at most $k$
        vertices. This vertex cover of $G$ is obtained by adding $v_1$ to the vertex
        cover of the induced subgraph.
   3.2: Else, if the graph obtained by removing $v_2$ from $G$ contains a vertex
        cover of at most $k - 1$ vertices, then $G$ contains a vertex cover of at
        most $k$ vertices.
   3.3: Else, $G$ contains no vertex cover of at most $k$ vertices.
```

This algorithm makes at most two recursive calls to itself, each time reducing the value of k. Thus, no more than 2^k calls to this algorithm are made. As each individual call can be executed in a time polynomial in n, we find that deciding membership in VertexCover is fixed-parameter tractable.

The parameter in the parameterized decision problem VertexCover emerges naturally. We could say that we have parameterized the vertex cover problem by the solution size. Similarly, we could consider a parameterized version of the Clique problem, where we parameterize by the solution size. That is, we are given a graph G and a number k, and are asked whether the graph G contains a clique of at least k vertices. If we let n denote the number of vertices in G, we find that there are no more than n^k subsets of k vertices of G. From this, we immediately find that deciding membership in the parameterized version of Clique is possible within a time bound of the form n^k. At the same time, it is widely believed that deciding membership in Clique, parameterized by the solution size, is not fixed-parameter tractable [44, 37]. This is where things get interesting, because when parameterized differently, Clique may well be fixed-parameter tractable. Recall the definition of Clique from Example 1.2.2,

$$\textsc{Clique} = \{(G, l) \mid \text{there is a set of at least } l \text{ vertices of the graph } G \text{ in which each pair of vertices is connected by an edge}\},$$

and consider Clique parameterized by the minimum vertex cover size,

$$\textsc{Clique}_{\mathrm{VC}} = \{\langle (G, l), k\rangle \mid (G, l) \in \textsc{Clique} \text{ and } G \text{ has a vertex cover of at most } k \text{ vertices}\}.$$

An algorithm witnessing that $\textsc{Clique}_{\text{VC}}$ is fixed-parameter tractable may proceed as follows when given G, l, and k as input, where G has n vertices [37, Section 15.2.4].

```
1: Use our earlier algorithm for VertexCover to find a vertex cover, $C$,
   of $G$ that contains at most $k$ vertices. Note that the running time of this
   algorithm is as required for showing that VertexCover is fixed-parameter
   tractable.
2: If no such vertex cover exists, our input instance is not a member of
   $\textsc{Clique}_{\text{VC}}$.
3: A clique in $G$ contains at most one vertex outside $C$, so there are no more
   than $2^k(n-k)$ sets of vertices to consider as potential cliques in $G$. If any
   of these sets is a clique of at least $l$ vertices, then our input instance is a
   member of $\textsc{Clique}_{\text{VC}}$.
```

The parameterized decision problem $\textsc{Clique}_{\text{VC}}$ showcases some of the downsides of the framework of Downey and Fellows. Because the framework deals only with parameterized decision problems, it cannot be applied to a classical problem such as Clique. Instead, the framework requires us to resort to derived parameterized problems such as $\textsc{Clique}_{\text{VC}}$. In doing so, the framework obstructs an analysis of the parameterized computational complexity of nonmembers. For example, there can be two reasons why some instance $\langle (G,l),k\rangle$ is not a member of $\textsc{Clique}_{\text{VC}}$. It may be because (G,l) is not a member of Clique, but it may also be that the parameter value k is too small. The latter possibility is particularly troublesome, as for some parameterized problems the least sufficient parameter value is the key point of interest. When we parameterize Clique by the solution size, every graph G has a parameter value k such that G contains a clique of at most k vertices. On the other hand, there are tuples (G,l) for which there is no parameter value k such that $\langle (G,l),k\rangle$ is a member of $\textsc{Clique}_{\text{VC}}$. Thus, the Downey and Fellows framework cannot be used to inquire about the parameterized computational complexity of nonmembers of classical problems.

Flum and Grohe

An alternative framework for parameterized computational complexity was given by Flum and Grohe [57]. In their framework, problems are analyzed in light of *parameterization* functions. This way, there is no need to confine the analysis to a special type of parameterized decision problems. With Flum and Grohe, a *parameterization* is a function κ that maps an instance x of a problem A to a numeric parameter value $\kappa(x)$. The function κ is required to be computable in polynomial time. With this, we mean that if x has length n, the time required for computing $\kappa(x)$ can be bounded by a polynomial in n. Of course, the polynomial must be independent of x. The definition of fixed-parameter tractability of A with respect to κ is similar to the definition in the framework of Downey and Fellows. This time, however, the parameter value is not part of the input of a decision procedure for A, but obtained using κ. Specifically, for some function f and constant c, membership in A of an instance x of length n must be decidable in time $f(\kappa(x)) \cdot n^c$. As this approach separates parameters from instances, the framework allows for a study of the parameterized computational complexity of nonmembers. In this regard, it is an improvement over the framework of Downey and Fellows.

The parameters considered by Garey and Johnson are examples of parameters that can be modeled nicely in the Flum and Grohe framework. Thus, for these parameters, the framework of Flum and Grohe provides a way to analyze the complexity of nonmembers that was absent in the Downey and Fellows framework. Recall that for the Partition problem, an input instance x is a sequence of numbers $x_1, x_2, \dots, x_m$. The "maximum component length" parameter for such inputs can be expressed as a function

$$\kappa(x) = \max\{|x_1|, |x_2|, \dots, |x_m|\}.$$

Note that this function is computable in a time bounded polynomially in the length of the input, $|x_1| + |x_2| + \dots + |x_m|$. Therefore, the above function κ qualifies as a parameterization in the framework of Flum and Grohe. With Garey and Johnson, we saw that Partition has a pseudo-polynomial time algorithm. In the parlance of Flum and Grohe, we would say that Partition is fixed-parameter tractable with respect to the parameterization κ. A similar observation can be made about the parameter with respect to which Garey and Johnson consider Clique. This parameter is represented by a function that takes as input a graph x, and computes the highest degree of any of the vertices of x. The time required for this computation can be bounded polynomially in the number of vertices in the instance x. Thus, this parameter too can be modeled in the Flum and Grohe framework.

The minimum vertex cover size, as used with Clique$_{\text{VC}}$, is a more problematic parameter. Unless **NP** equals **P**, the minimum vertex cover size cannot be computed in polynomial time. Hence, it would not qualify as a parameterization in the sense of Flum and Grohe. A typical solution to this is to make the desired parameter value a part of the instance [for instance 37, Section 15.2.4]. Instead of looking at Clique with respect to a function that computes the minimum vertex cover size, we turn to Clique$_{\text{VC}}$. In general, we can define a parameterization in the style of Flum and Grohe for parameterized decision problems in the style of Downey and Fellows. For a set of pairs $\langle x, k\rangle$, where k is the parameter value, the appropriate parameterization is defined as

$$\kappa(\langle x, k\rangle) = k. \tag{1.3}$$

We have already seen that CLIQUE$_{\mathrm{VC}}$ is fixed-parameter tractable in the framework of Downey and Fellows. From that, we can conclude that CLIQUE$_{\mathrm{VC}}$ is also fixed-parameter tractable with respect to (1.3) in the Flum and Grohe framework. However, we are still unable to analyze the computational complexity of the unmodified classical CLIQUE problem with respect to the minimum vertex cover size. This objection can be responded to in two ways. We may hold the position that the minimum vertex cover size is not a proper parameter, as it cannot be computed in polynomial time, assuming **NP** differs from **P**. On the other hand, we should take into account that VERTEXCOVER is fixed-parameter tractable with respect to the size of its solution. That is, it is fixed-parameter tractable with respect to the parameterization that maps an instance (G, k) to k. For this reason, an algorithm for CLIQUE$_{\mathrm{VC}}$ with a running time as required by fixed-parameter tractability has sufficient time to compute the minimum vertex cover size.

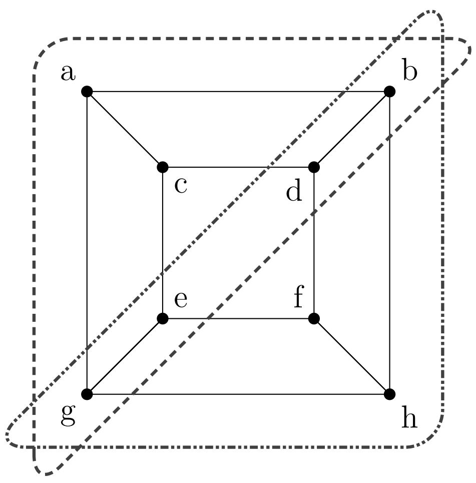
An important parameter of graphs that has a similar issue as the minimum vertex cover size is *treewidth* [129, 23, 40]. For many decision problems on graphs, it has been observed that the more an instance resembles a tree, the easier it is to decide membership. This is because information regarding the decision can be propagated along the branches of the tree. The resemblance to a tree can be formalized by considering ways in which a graph G can be decomposed into a tree that represents G.

1.3.1. DEFINITION. A *tree decomposition* of a graph G is a tree T of which the vertices are subsets of vertices of G, and that satisfies

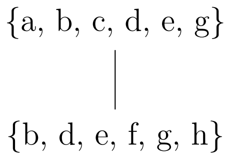
- each vertex of G occurs in at least one of the vertices of T,
- of each edge of G, the endpoints occur together in at least one of the vertices of T, and
- if a vertex of G occurs in two vertices of T, then it occurs in all vertices of T on the unique path between the two.

Each graph G has a trivial tree decomposition consisting of a single vertex that contains all vertices of G. However, as can be seen in Figure 1.6, more sophisticated decompositions may be possible.

As suggested by the name "tree decomposition", trees have an especially interesting decomposition. In Figure 1.7, it is shown that a tree can be decomposed in such a way that the decomposition mimics the original tree. In the tree decomposition in Figure 1.7b no sets occur with more than two elements. By the definition of a tree decomposition, any tree decomposition of a graph with at least one edge has a set with at least two elements. In that sense, the tree decomposition in Figure 1.7b is optimal. This leads to the following measure of how much a graph is like a tree.

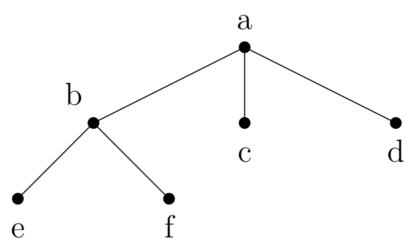


(a) The vertices of the cube graph can be put in two overlapping sets, indicated by the interrupted lines. These sets define a tree decomposition of the graph.

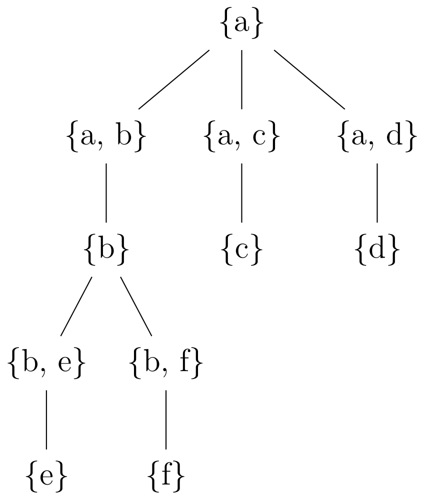


(b) The tree corresponding to the decomposition in Figure 1.6a consists of two vertices.

Figure 1.6: A tree decomposition of the cube graph of Figure 1.2a



(a) A tree



(b) A tree decomposition of the tree in Figure 1.7a. Observe that the decomposition retains the essence of the structure of the original tree.

Figure 1.7: A tree decomposition of a tree

1.3.2. Definition. The *width* of a tree decomposition is the number of elements in the largest set among the vertices of the decomposition, minus one. The *treewidth* of a graph is the least width among the widths of its tree decompositions.

Indeed, the idea behind the tree decomposition shown in Figure 1.7 can be used to prove that the treewidth of any tree with at least one edge is one. Making sure that trees have a treewidth of one is in fact the reason for the otherwise somewhat peculiar "minus one" in the definition of the width of a tree. It follows from Figure 1.6 that the treewidth of the cube graph is at most five. Yet, a tree decomposition of smaller width is possible, as shown in Figure 1.8. This decomposition happens to be optimal and the treewidth of the cube graph is three.

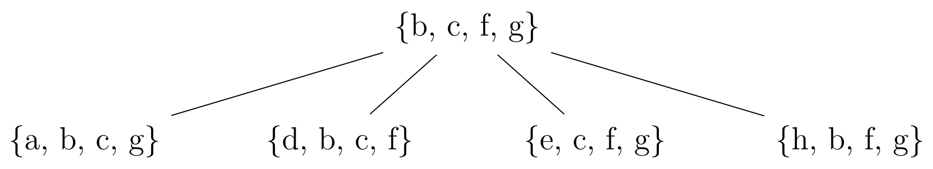


Figure 1.8: An alternative tree decomposition of the cube graph as shown in Figure 1.6a

By a fundamental result known as Courcelle's theorem, many graph problems become fixed-parameter tractable once parameterized by treewidth in the Downey and Fellows framework [44]. This includes, for example, Clique and Vertex-Cover [25]. However, the problem of deciding whether the treewidth of an arbitrary graph is less than or equal to a given number is itself **NP**-complete [12]. Therefore, unless **NP** equals **P**, treewidth does not qualify as a parameterization in the framework of Flum and Grohe. At the same time, like with the minimum vertex cover size, computing treewidth is fixed-parameter tractable with respect to itself [130, stated in terms of "branch-width"]. It is noted by Flum and Grohe that large parts of the theory of parameterized complexity go through if we allow for such parameterizations [57, p. 279].

It has also been recognized that there is little reason why a parameterization should be restricted to only map to a single number [54, 114]. A parameterization with a multidimensional image would represent multiple parameters at once. The interplay between parameters can, however, not be expressed in the Flum and Grohe framework. For example, the minimum vertex cover size of a given graph G may go down when we remove some of the edges in G. If we consider the number of edges we remove from a graph and the minimum possible vertex cover size of the resulting graph, we have two interacting parameters. To bring the minimum vertex cover size down, we need to increase the number of edges we remove. Note, though, that being allowed to remove an edge need not make it possible to lower the minimum vertex cover size. We need to remove at least three edges from the cube graph of Figure 1.2a before the minimum vertex cover size of the graph is no longer four. It is unclear what a parameterization in the framework of Flum and Grohe should map a graph to. Conceivably, it could map a graph to the *function* that maps the number of modifications performed to the minimum vertex cover size of the resulting graph. However, this goes against the spirit of fixed-parameter tractability analysis, where we would like to look at specific values for both of our parameters. On the one hand, the Flum and Grohe framework improves on the Downey and Fellows framework by separating parameterizations from problems. It isolates parameterizations as independent objects of study. On the other, it requires parameter values to be unique for a given instance, which limits its use in analyzing the interplay between different parameters.

1.4 Contributions

This thesis, in Section 2.2, Section 3.2.1, and Section 3.2.2, introduces a novel take on parameterized complexity theory, rooted firmly in mathematics. In this new framework, the shortcomings of the two existing frameworks are addressed. Our contributions are obtained in the five sections of Chapter 3.

We begin an investigation into the extent to which parameterizations can be used as a measure of complexity of individual instances. A comparison with an older notion of *instance complexity* from algorithmic complexity theory is made in Section 3.3.1. In Section 3.3.3, a further comparison between instance-based measures of algorithmic and computational complexity is carried out. A comparison with the notion of *sophistication* from algorithmic statistics is made in Section 3.4.

There may be many parameterizations that are of interest for a given problem, so we might hope for the existence of a parameterization that is somehow *optimal*. In Section 3.2.4 and Section 3.2.3, we provide a near-complete characterization of the problems for which there is an optimal parameterization with respect to fixed-parameter tractability. We find that there is often no optimal parameterization among those with which a problem is fixed-parameter tractable.

More generally, this thesis furthers our understanding of the class of fixed-parameter tractable problems. We explore what it means for a problem to be fixed-parameter tractable. Results of this kind are obtained as a by-product of a more general study of parameterized computability in Section 3.1.

An alternative characterization of the class of fixed-parameter tractable problems that does not mention parameterizations is suggested in Section 3.3.2. In that section, we find out what the parameterizations with respect to which a problem is fixed-parameter tractable tell us about the problem. We conjecture that the notion of complexity put forth by fixed-parameter tractability is characterized by the quotient group $\Delta_1^0/\mathbf{P}$ with respect to symmetric difference. Here, Δ_1^0 is the class of decidable sets, the first intersection level of the arithmetical hierarchy.

Lastly, in Section 3.5, we uncover a hierarchy inside the class of fixed-parameter tractable problems. This hierarchy is based on polynomial kernelization, which is a form of reducibility.

Chapter 2

Preliminaries

This chapter makes precise what most of the jargon that appears in our parameterized analysis of complexity means. The chapter is split into two parts, the second of which, Section 2.2, should not be skipped in a first reading. In that section, we introduce a novel framework for parameterized complexity theory, building on the frameworks discussed in Section 1.3. All of the theory in Chapter 3 will be developed in this new framework. We remark that many of our results are independent of the framework they are expressed in. However, using one of the traditional frameworks of Section 1.3 may make the theorems less general, or at the least less elegant. A brief summary of the notation that is specific to our framework can be found in the back of this thesis.

The first part of the current chapter, Section 2.1, does not contain anything not available in textbooks. We have divided it into subsections corresponding to particular research areas. While the section does include some background on our usage of certain terms, it, or segments of it, may be skipped at will or used only as a reference. The numbers of the pages on which definitions are listed can be found in the index at the back of this thesis.

2.1 Established Theory

We aim to bring together multiple notions of complexity. It is unavoidable that, in doing so, we use terminology from multiple areas of research. Broadly, each section of Chapter 3 deals with a form of complexity as it is encountered in a specific area of research. Definitions that are specific to a certain area of research are stated in the appropriate sections. More basic definitions that span multiple areas of research are included here. While we accompany all definitions with some context, we cannot give a complete introduction to each of the research areas that we draw definitions from.

2.1.1 Binary Representations

In this thesis, as in much of the computer science literature, all sets are countable. That means that for every set A, there exists a surjective function mapping the natural numbers, $\mathbb{N} = \{1, 2, 3, ...\}$ onto A. The elements of a countable set can be represented by finite *strings* of characters from a finite *alphabet*. We shall work solely with the binary alphabet $\mathbb{2} = \{\mathtt{0}, \mathtt{1}\}$, but all results can be generalized to more elaborate alphabets. The finite, nonempty binary strings, $\mathbb{2}^+$, can be enumerated in order of increasing length as $\mathtt{0}, \mathtt{1}, \mathtt{00}, \mathtt{01}, \mathtt{10}, ...$ Thus, there is a bijective correspondence between $\mathbb{N}$ and $\mathbb{2}^+$. Taking a cue from common practice in programming languages, we treat the two directions of this bijective correspondence as data type conversions. A string can serve "as an integer", that is "as a natural number", and a natural number can serve "as a string". These data type conversions are depicted in Table 2.1. In this thesis, we have tried to make all conversions explicit.

We have chosen to forego any consideration of the empty string and, in this thesis, we do not consider 0 to be a natural number. Of course, these choices are related to each other and make that the set of possible lengths of our strings equal the set of natural numbers. Our choices ensure that any statement about a natural number that is true "up to an additive constant" is also true "up to a multiplicative constant". Let us give an example. If, given two natural numbers x and y, there is a constant c such that we have $x \leq y + c$, then there is also a constant c' such that we have $x \leq c \cdot y$. The fact that this need not be the case if x was equal to 0 makes that 0 is often a special case that needs special treatment. By leaving out 0 from the start, we are not bothered by these pathological cases.

The elements of $\mathbb{2}$ are known as *bits*. The number of bits in a binary string x is called the *length* of x and denoted by $|x|$. We denote the set of all strings of some length n by $\mathbb{2}^n$. Observe that for any two natural numbers m and n, we have

$$m \leq n \implies |\mathrm{asStr}(m)| \leq |\mathrm{asStr}(n)|.$$

Furthermore, we implicitly take all our logarithms to base 2. This ensures that $\log n$ is within one bit of $|\mathrm{asStr}(n)|$. By not being pedantic about this difference

$$\begin{tikzcd}\mathbb{N} \arrow[r, bend left, "\text{asStr}"] & 2^+ \arrow[l, bend left, "\text{asInt}"]\end{tikzcd}$$

1	`0`
2	`1`
3	`00`
4	`01`
5	`10`
6	`11`
7	`000`
8	`001`
⋮	⋮

Table 2.1: A bijective correspondence between the natural numbers and the finite, nonempty binary strings. The two directions of this correspondence are denoted by asStr and asInt. Note that the number 0 and the empty string are never used in this thesis.

of at most one bit, we can say that n elements can be distinguished from one another using $\log n$ bits.

Two strings x and y can be concatenated to form a new string. This concatenation of x and y is written simply as xy. Note that we have $|xy| = |x| + |y|$. The n-fold concatenation of x with itself is written as x^n. In particular, we have

$$\mathtt{0}^n = \underbrace{\mathtt{000}\cdots\mathtt{0}}_{n \text{ copies of } \mathtt{0}} .$$

Given the concatenation of two strings, we cannot recover the two original strings. Indeed, an equation like `010111001` $= xy$ has no unique solution. We do not know what initial segment, or *prefix*, of `010111001` corresponds to x. If we want to be able to recover the components in a composite string, we cannot simply concatenate them.

2.1.1. Definition. Let Ω be any set. An injective function $f\colon \Omega \to 2^+$ is a *prefix-free encoding* of Ω if there are no two distinct elements ω_1 and ω_2 of Ω such that $f(\omega_1)$ is a prefix of $f(\omega_2)$.

With a suitable application of a prefix-free encoding of binary strings, it is possible to uniquely decompose a composite string [36]. Perhaps the simplest prefix-free encoding of binary strings is the *unary encoding*, where the position of the first `1` determines the encoded string. The unary encoding of a binary string x is

$$\text{unary}(x) = \mathtt{0}^{\text{asInt}(x)-1}\mathtt{1}.$$

Note that the unary encodings of any two distinct strings are different and that the unary encoding of a string always ends at the first occurrence of '`1`'. Therefore, unary encoding is indeed a prefix-free encoding of $\mathbb{2}^+$. Contrary to the situation we were in with direct concatenation, the equation $\texttt{010111001} = \mathrm{unary}(x)y$ *does* have a unique solution. Namely, it has the solution $x = \mathrm{asStr}(2) = \texttt{1}$ and $y = \texttt{0111001}$.

Unary encoding comes with a substantial blow-up in the length of a string. For any string x, we have $|\mathrm{unary}(x)| = \mathrm{asInt}(x)$, which is exponential in $|x|$. More frugal prefix-free encodings were developed by Elias [48]. The two most famous of these are known as *Elias gamma coding* and *Elias delta coding* [see also 133]. In Elias gamma coding, a string is prefixed by a unary encoding of its length,

$$\mathrm{Elias}_\gamma(x) = \mathrm{unary}(\mathrm{asStr}(|x|))x = \texttt{0}^{|x|-1}\texttt{1}x.$$

Thus, the position of the first occurrence of '`1`' tells us how many of the following bits make up the encoded string x. So, the equation $\texttt{010111001} = \mathrm{Elias}_\gamma(x)y$ has the unique solution $x = \texttt{01}$ and $y = \texttt{11001}$.

With Elias gamma coding, we have $|\mathrm{Elias}_\gamma(x)| = 2 \cdot |x|$. The same trick underlying Elias gamma coding can be used to bring the length of the encoding of x even closer to $|x|$. In Elias delta coding, we replace the unary encoding of the length of x by an Elias gamma encoding of that length,

$$\mathrm{Elias}_\delta(x) = \mathrm{Elias}_\gamma(\mathrm{asStr}(|x|))x = \texttt{0}^{|\mathrm{asStr}(|x|)|-1}\texttt{1}\,\mathrm{asStr}(|x|)x.$$

For a string x of length n, this gives us $|\mathrm{Elias}_\delta(x)| = 2 \cdot |\mathrm{asStr}(n)| + n$, which is approximately $n + 2 \cdot \log n$.

For our purposes, Elias delta coding is sufficiently economical. We shall use it whenever we need to pair strings.

2.1.2. Definition. A *pairing function* that combines two finite, nonempty binary strings x and y into a single string is given by

$$\langle x, y\rangle = \mathrm{Elias}_\delta(y)x.$$

Note that this pairing function reverses the order in which the components are presented. There is no fundamental reason for this technical detail other than that it will be convenient in Example 3.4.9 and in Section 3.4 in general. Returning to our example string one final time, we observe $\texttt{010111001} = \mathrm{Elias}_\delta(\texttt{1100})\texttt{1}$ and hence $\texttt{010111001} = \langle \texttt{1}, \texttt{1100}\rangle$.

Our pairing function does not use any prefix-free encoding for its first component. This has the effect that our pairing function is not a prefix-free encoding of $\mathbb{2}^+ \times \mathbb{2}^+$. At the same time, it results in $|\langle x, y\rangle|$ being approximately equal to $|x| + |y| + 2 \cdot \log |y|$.

Prefix-free encodings have an interesting relation to probability theory. The length of the prefix-free encoding of a string can be linked to the probability of that string in some probability mass function [36, 100].

2.1.3. Theorem (Kraft inequality). *If f is a prefix-free encoding of a set Ω, then we have*

$$\sum_{\omega\in\Omega} 2^{-|f(\omega)|} \leq 1.$$

Conversely, if there is a function $\ell\colon \Omega \to \mathbb{N}$ that satisfies $\sum_{\omega\in\Omega} 2^{-\ell(\omega)} \leq 1$, then there is a prefix-free encoding f of Ω such that, for all ω, we have $|f(\omega)| = \ell(\omega)$.

Thus, the quantity $2^{-|f(\omega)|}$ acts like a probability of ω. For technical reasons, the probability mass function is allowed to sum to less than 1 in the Kraft inequality. This can be fixed by either normalization, or by introducing a “slack object” that takes the remaining probability.

2.1.2 Algorithmic Complexity and Computability Theory

We take a pragmatic approach to computability theory and do not consider physical computers to be approximations of Turing machines. Instead, we consider Turing machines to be asymptotically correct mathematical models of physical computers. Our model of effective computation hence takes the form of “reasonable” *pseudocode*. That is, a programming language that is more or less obviously Turing complete, yet not stronger than that. A more traditional, but up-to-date textbook on Turing computability is the textbook by Soare [140].

With pseudocode, we can specify an *algorithm*, which represent the workings of a particular Turing machine [see also 131]. We shall not concern ourselves with the details of our pseudocode-language. Instead, we accept that it is possible to specify such a language in sufficient detail, and assume that we have an effective prefix-free encoding of algorithms. We shall refer to an encoded algorithm as a *procedure*. Note that a procedure ϕ has a length $|\phi|$, which is measured in bits.

As algorithms may take input and produce output, they implement partial functions. The functions they implement may be partial functions, because, for a given input, an algorithm may keep running forever and never settle on any output. When an algorithm terminates on all possible inputs and the associated function f is hence a total function, we say that f is *computable*.

Algorithmic complexity theory looks at procedures that generate a given string. The most widely used measure of algorithmic complexity is *Kolmogorov complexity* [36, 100, 46].

2.1.4. Definition. The *Kolmogorov complexity* of a string x is

$$\mathrm{K}(x) = \min\{|\phi| \mid \text{procedure } \phi \text{ produces } x \text{ when not provided with any input}\}.$$

The precise value of the Kolmogorov complexity of a string depends on the prefix-free encoding of algorithms that is used. However, Kolmogorov complexity enjoys an invariance theorem, stating that this dependency is limited to an additive constant [see, for instance 100]. In order not to blur the main ideas in this thesis, we shall not write "up to an additive constant" after every (in)equality involving Kolmogorov complexity. This is left implicit.

Complexity theory, on the other hand, looks at procedures that output a single bit, either `1` or `0`. If a procedure outputs a single bit and does so on all possible input strings, it is said to be a *decision procedure.* A decision procedure ϕ *decides* on membership in the set

$$A = \{x \mid \phi(x) = \mathtt{1}\}.$$

In turn, the set A is said to be *decidable.* If the procedure ϕ fails to halt on some inputs, the set $A = \{x \mid \phi(x) = \mathtt{1}\}$ is said to be *semidecidable.* The semidecidable sets are more commonly known as the "recursively enumerable" sets. However, we prefer to use the word "recursive" only for the algorithmic design pattern by that name. We do note that the semidecidable sets can equivalently be defined as those sets of which the members can be enumerated effectively.

The class of semidecidable sets contains sets that are not in the class of decidable sets. Moreover, not all sets are semidecidable. In fact, a set can be highly dissimilar to all semidecidable sets at once.

2.1.5. Definition. A set A is *immune* for a class of sets $\boldsymbol{C}$ if no infinitely large member of $\boldsymbol{C}$ is a subset of A.

A set A is *bi-immune* for a class of sets $\boldsymbol{C}$ if both A and its complement are immune for $\boldsymbol{C}$.

If no specific mention of a class of sets $\boldsymbol{C}$ is made, the class implicated is that of the semidecidable sets [131, 115].

Immune sets where introduced to computability theory by Post [123]. The application to arbitrary classes of sets goes back to Flajolet and Steyaert [55].

If the class $\boldsymbol{C}$ in Definition 2.1.5 has infinitely many infinitely large members, an immune set A satisfies infinitely many requirements. Namely, for every infinitely large set C in $\boldsymbol{C}$, the set A is so that $C \setminus A$ is nonempty. In many cases, constructing a set that meets infinitely many of such requirements is not straightforward. It may be complicated by the fact that satisfying one requirement may violate satisfying a requirement that was taken care of earlier. By imposing a priority ordering on the requirements and always choosing to satisfy the requirement with the highest priority, immune sets can be constructed. This construction method is known as the *finite injury priority method* [for details, see 46, Section 2.11].

By a slight stretch of notation, we may use a set as a function. The function associated with a set A is defined as

$$A(x) = \begin{cases} \mathtt{1} & \text{if } x \in A, \\ \mathtt{0} & \text{otherwise.} \end{cases}$$

Thus, a set is decidable when the function associated to it is computable. Moreover, a procedure that implements the function associated with a set A is a decision procedure for A. In case a set A is not decidable, we can investigate what would happen if it were decidable, by supposing the function associated with A is computable. In particular, we consider the sets that "become decidable" when this function is computable [131, 115, 140].

2.1.6. Definition. A set B is *Turing reducible* to a set A if there is an algorithm that

- is allowed to evaluate the function associated with A, and
- decides on membership in B.

Such an algorithm is known as a *Turing reduction.* The set A is known as an *oracle* and the evaluations of the function associated with A are known as *queries.*

The behavior of a Turing reduction may be heavily reliant on the oracle. Specifically, at any point in its execution, the outcomes of the queries it has made so far may influence the query it makes next. If a Turing reduction is allowed to have such a dependence on the outcomes of queries, it is said to be *adaptive.* Sometimes, it is desirable to consider reducibility where the queries are independent of the oracle.

2.1.7. Definition. A set B is *truth-table reducible* to a set A if there is a Turing reduction ϕ from B to A such that

- the queries made by ϕ depend only on its input and not on the oracle, and
- ϕ terminates on all inputs, regardless of the oracle.

Such a Turing reduction is known as a *truth-table reduction.*

For a given input to a truth-table reduction, we can gather the queries that would be made. The possible outcomes of these queries can be assembled as rows in a table. For each row in this table, we can compute the decision on membership that the truth-table reduction would make. For this reason, reductions of this kind are called truth-table reductions.

The power of a reduction can be restricted further. A truth-table reduction that always makes precisely one query and outputs the unmodified result of that query is known as a *many–one reduction.* If there is a many–one reduction from a set B to a set A, then B is said to be *many–one reducible* to A.

2.1.3 Proof Theory

The Turing machine was developed as a model for computation as it can conceivably be performed by humans [144, Section 9]. In that respect, it is not surprising that computability theory has close ties to proof theory. After all, proof theory should seek to model what humans could ultimately achieve in terms of reasoning [90, Section 37]. Accomplishments in human computation and human reasoning are intuitively subject to the same intellectual limitations.

The link between computability theory and proof theory has been formalized into what is known as the *Curry–Howard correspondence*, or the *formulas-as-types isomorphism* [143, Section 2.5]. This isomorphism draws a parallel between proofs and algorithms. Just like we have freedom in the choice of a pseudocode-language, there is freedom in the choice of a “proof language”, or *proof calculus* [90, 143]. We did not concern ourselves with the details of our pseudocode-language and neither shall we concern ourselves with the details of proof calculi. However, at the heart of every proof calculus lies a *formal (deductive) system* [90], and we shall briefly go over what is involved with such a system.

Typically, expositions of formal systems start with the definition of a *formal language*. A formal language is a set of finite strings of symbols from some finite alphabet. The strings in a formal language are called the *well-formed words*, or *well-formed formulas*. In this thesis, we are only concerned with formal systems of which the formal language is decidable.

2.1.8. Example. Suppose we want our formal system to deal with tree structures. We then need a way to represent such structures. One way to do so is by using the symbols ‘(’ and ‘)’ and ‘,’ and ‘$*$’. Using these symbols, we can serialize a nested tree structure, with ‘$*$’ representing a leaf in the tree. The structure of the tree in Figure 1.7a would thus be represented by the string “$((*,*),*,*)$”.

Not all strings consisting of our symbols represent a tree structure. For instance, the string “$()*$” has no meaningful interpretation and is therefore not well-formed. The set of strings that do represent tree structures is a decidable subset of the set of all strings of our symbols.

The goal of a formal system is to single out a subset of the well-formed formulas that is of some special interest. For instance, if the formal language is that of propositional logic, we may want to characterize the tautologies. To this end, a formal system contains a way of deriving well-formed formulas from other well-formed formulas by means of *rules of inference*. The idea here, is that the subset of interest is closed under these rules of inference. The rules of inference can be thought of as a set of procedures, where each procedure takes a number of well-formed formulas and produces a new one. The number of inputs is fixed for each procedure and if the procedure takes no inputs, then it is said to be an *axiom*. Thus, an axiom is a procedure that produces a well-formed formula. For the purposes of this thesis, we restrict our attention to formal systems where the rules of inference are represented by a decidable set of procedures.

2.1.9. Example (continued). Suppose that, in our system of tree structures, we want to derive the nonempty binary trees. This can be done with one axiom and one other rule of inference. The axiom is a procedure that takes no inputs and outputs the minimal binary tree, the tree with a root node and no branches: "$*$". The other rule of inference is a procedure that transforms two tree structures, x and y, into the tree structure "(x, y)". Using these two rules of inference, all binary trees can be derived, and nothing else. For example, it is impossible to generate our earlier example, "$((*, *), *, *)$" using only these two rules, because it does not represent a binary tree.

A *proof*, or *deduction*, in a formal system establishes that some given well-formed formula x is part of the subset of special interest. It does so by specifying a sequence of rules of inference that, together, output x. Thus, the proof constructs x from the axioms of the formal system.

In this thesis, we are only ever interested in the well-formed formulas of a formal language. This is most notable in how we approach decision problems. For example, when we model a graph-theoretic decision problem as a set of graphs, we expect the input to a decision procedure for the set to be a graph. We pay no attention to what would happen if the input does not represent a graph. In fact, we shall routinely assume that there is some computable bijection between the well-formed formulas and the binary strings, 2^+. A decision procedure for a graph problem can then be provided with a binary string as input, and we can act as if it was given a graph. To hide this conceptual difference between well-formed formulas and binary strings, we shall refer to the input of a decision procedure as a problem *instance*. Put differently, in the context of a decision problem modeled as a set, an instance is a syntactically correct specification of a potential member of the set. Where the details of the encoding of well-formed formulas as binary strings matter, we shall be explicit about the encoding we use.

2.1.4 Computational Complexity Theory

Our choice of using pseudocode-algorithms as a model of computation does not mean that our results are limited to that particular model of computation. Many other models of computation identify the exact same class of partial functions as computable. The Church–Turing thesis asserts that this class includes effective computation by an idealized human, in some intuitive, informal, sense [49, 68]. However, the Church–Turing thesis does not imply that *all* our results extend to all models of computation that are equivalent to the Turing machine.

We have mentioned that the precise value of the Kolmogorov complexity of a string depends on the prefix-free encoding of algorithms that is used. This dependence is limited to an additive constant by the existence of a *universal* procedure that takes a procedure ϕ and string x as inputs, and outputs $\phi(x)$ [100, 68]. Likewise, the dependence of Kolmogorov complexity on the model of computation is limited to an additive constant, as long as the models can simulate each other. This idea is made precise by Rogers [131] in the form of *acceptable numberings* of partial computable functions [see also 140]: When a model of computation corresponds to an acceptable numbering of functions in our reference model, its notion of Kolmogorov complexity equals ours. Equivalently, all models of computation that are equivalent to the Turing machine and have a universal procedure share a notion of Kolmogorov complexity.

There is no canonical representation, or, for that matter, formal definition, of a computational procedure by an idealized human. Therefore, the Church–Turing thesis does not entail that Kolmogorov complexity coincides with some intuitive human understanding of algorithmic complexity. Additionally, it does not entail that in different models, the same functions can be computed in the presence of a given bound on some computational resource. Indeed, an analysis of the resource requirements for computing a given function need not transcend the model of computation that is used.

The main computational resource we look at in this thesis is the number of steps taken by an algorithm, referred to as computation *time*. To more cleanly express the asymptotic behavior of computation-time usage, computation time is commonly measured as a function of the length of input instances. That is, the computation time of a procedure ϕ is a function t such that, for all n, the maximum number of steps taken by ϕ on any input of length n is $t(n)$. Typically, it is not so much the exact computation time of a procedure we are interested in, but rather an upper bound on the computation time. When we say that a procedure ϕ runs in time t, we mean that, for all n, the number of steps taken by ϕ on any input of length n is at most $t(n)$. A noteworthy class of functions in this regard are the *time-constructible functions*. These are the functions t for which there is a procedure ϕ and a constant c such that, for all strings x,

- ϕ outputs the string representation of $t(\text{asInt}(x))$ on input x, and
- ϕ takes at most $c \cdot t(|x|)$ steps on input x.

Most commonly used functions, among which all polynomials of degree at least 1, are time-constructible [13].

Many models of computation that have been studied can simulate each other with only polynomially-bounded overhead in computation time [49]. In fact, it has been conjectured that, for some definition of "reasonable", all reasonable machines can simulate each other with polynomially-bounded overhead. This extension to the Church–Turing thesis is, among other names, known as the *sequential computation thesis*. The name reflects the fact that the thesis is found to exclude some parallel models of computation [120, 49]. We shall not be concerned with whether or not the sequential computation thesis is true in general, but we shall assume a special case. As argued by Turing [144], the idealized human computer may be thought of as a sequential model of computation. We accept these arguments to the extent that the sequential computation thesis can be applied to the idealized human computer. In particular, we assume that the idealized human computer and our model of computation can indeed simulate each other with polynomially-bounded overhead. Like the Church–Turing thesis, this statement is necessarily informal. Specifically, there is no formal definition of what a single computational step of a human computer would be.

The sequential computation thesis is especially relevant in light of yet another thesis on the computational abilities of the idealized human computer. Around 1965, Cobham and Edmonds argued that the functions that should be called *efficiently computable* are those that can be computed in polynomial time [see also 68]. Thanks to the sequential computation thesis, efficient computability is a somewhat robust notion. We may assume that what is efficiently computable in our model of computation is also efficiently computable for the idealized human computer and vice versa.

The class of sets that have efficiently computable decision procedures in our model of computation is denoted by **P**. For other models of computation, the class of sets that have efficiently computable decision procedures may be different from **P**. We shall mention two alternative models of computation and their associated classes of efficiently decidable sets. These classes shall also be characterized in our model of computation. For more precise definitions than those provided here, we refer to the textbooks by Arora and Barak [13] and Goldreich [68].

Our model of computation is deterministic, since at any point in the execution of an algorithm, it is known what the next operation will be. We can relax this property and allow an algorithm to proceed in a nondeterministic fashion. For such a model of computation it is not immediately clear what would constitute a decision procedure. Suppose ϕ is a nondeterministic algorithm that terminates on all inputs, regardless of the nondeterministic choices in its execution. We say that ϕ decides on membership in a set A if an instance x is in A precisely when there is a possible execution of ϕ on input x that outputs `1`. If such a nondeterministic decision procedure runs in polynomial time, it is efficiently computable in the nondeterministic model of computation. The class of sets that have efficiently computable nondeterministic decision procedures is denoted by **NP**.

Our model of computation is *uniform*, meaning that for any procedure the same algorithm is used for all inputs. A model that relaxes this property is that where computation is represented by Boolean circuits, with a different circuit for each input length [see 13, 68]. An efficient decision procedure in this model is one where the size of the circuits can be bounded polynomially as a function of the input length. The corresponding class of sets that have efficient nonuniform decision procedures is denoted by $\mathbf{P_{/poly}}$.

The class $\mathbf{P_{/poly}}$ is not countable and consequently it is different from **P**. We remark that, more generally, there cannot be a prefix-free encoding of nonuniform circuits because there are uncountably many of them. Whether or not **NP** is different from **P** is a major unsolved problem in computer science.

Perhaps surprisingly, the two classes, **NP** and $\mathbf{P_{/poly}}$, can be characterized in our model of computation in similar ways. We shall state these as lemmas, but refer to the textbooks for proofs [for **NP**: 13, Section 2.1, and 68, Section 2.1.5] [for $\mathbf{P}_{/\mathbf{poly}}$: 13, Section 6.3, and 68, Section 3.1].

2.1.10. Lemma. *A set A is in* **NP** *if there is a polynomial p and a two-argument procedure ϕ that runs in polynomial time such that, for all n, we have*

$$\forall x \in 2^n : \Big(x \in A \iff \exists y \in 2^{p(n)} : \phi(x, y) = 1 \Big).$$

The variable y in this characterization is known as a *certificate* for membership of x in A.

2.1.11. Lemma. *A set A is in* $\mathbf{P}_{/\mathbf{poly}}$ *if there is a polynomial p and a two-argument procedure ϕ that runs in polynomial time such that, for all n, we have*

$$\exists y \in 2^{p(n)} : \forall x \in 2^n : \Big(x \in A \iff \phi(x, y) = 1 \Big).$$

In this case, the variable y is known as *advice* for instances of length n. Observe that the only difference between the characterization of **NP** and that of $\mathbf{P}_{/\mathbf{poly}}$ is the position of the existential quantification over y.

From the previous two lemmas, it follows that the classes **NP** and $\mathbf{P}_{/\mathbf{poly}}$ are closed under many–one reductions that are computable in polynomial time: If there is a polynomial-time computable many–one reduction from a set B to a set A and A is in either of these classes, then B is as well. Interestingly, there are sets in **NP** to which any other set in **NP** is polynomial-time many–one reducible. Such sets are said to be *complete* for **NP**. Recall that there are uncountably many sets in $\mathbf{P}_{/\mathbf{poly}}$. As there are only countably many reductions, this implies that there are no complete sets for $\mathbf{P}_{/\mathbf{poly}}$ with respect to polynomial-time many–one reducibility.

Efficient computation is not just about deciding membership in sets. Keeping with decision problems for a moment, we could consider a function that outputs the index of an instance in a listing of members of a set.

2.1.12. Definition. The *rank* of a string x relative to a set A, denoted by $\mathrm{rank}(x : A)$, is the number of elements in the set

$$\{ y \in A \mid \mathrm{asInt}(y) \leq \mathrm{asInt}(x) \}.$$

When there is a polynomial p such that, for all x, we have $\mathrm{rank}(x : A) \leq p(|x|)$, then A is said to be *sparse.* A set in relation to which the rank-function is computable in polynomial time is called *p-rankable.*

Thus, the rank counts the number of members of A that are less than or equal to a given element when converted to natural numbers. If the given element is not a member of A, the rank tells us how many members of A are less than the given element when converted to natural numbers.

Suppose a set A is p-rankable and let $x_1, x_2, x_3, \ldots$ be a listing of the members of A such that, for all i, the rank of x_i relative to A is i. We note that the inverse of the ranking function relative to A is computable in a time bounded by a polynomial of the length of the resulting instance. More precisely, there is a procedure ϕ and a polynomial p such that, for all i, we have

- $\phi(\mathrm{asStr}(i)) = x_i$, and
- the computation of $\phi(\mathrm{asStr}(i))$ terminates within $p(|x_i|)$ steps.

One way of constructing such a procedure ϕ is by using the rank-function in a binary search [79, Theorem 6.1].

The p-rankable sets were originally known as *strongly* $\mathbf{P}$*-rankable* [79, 67]. Here, the qualifier "strongly" is used to stress that the rank relative to a set A is defined not only for members of A, but also for nonmembers of A. The power of a procedure that computes the rank of an instance relative to a set A goes beyond that of a decision procedure for A. Indeed, the p-rankable sets form a subset of $\mathbf{P}$, because an instance x is a member of A precisely when we have

$$\mathrm{rank}(x : A) \neq \mathrm{rank}(\mathrm{asStr}(\mathrm{asInt}(x) - 1) : A).$$

Beyond efficient computability of variations of decision procedures, efficiently computable bijections are of interest too. Suppose that f is a bijection from 2^+ to itself that can be computed in polynomial time. Then, for every set $A \in \mathbf{P}$, the set $f(A) = \{f(x) \mid x \in A\}$ is also in $\mathbf{P}$. Observe that f maps members of A to members of $f(A)$, and likewise for nonmembers. In general, when a bijection keeps some kind of structure intact, the bijection is also referred to as an *isomorphism*. Furthermore, we say that sets that are related by an isomorphism, such as A and $f(A)$ in our example, are *isomorphic*. The *Berman–Hartmanis conjecture* [19] posits that all sets that are complete for $\mathbf{NP}$ are isomorphic to each other. This conjecture is believed to be untrue. It is at odds with the assumed existence of certain polynomial-time computable functions without a polynomial-time computable inverse [156, 99]. However, the two are not mutually exclusive and some functions that are not invertible may exist even if the conjecture is true [76, 5].

A class of sets that will appear in examples throughout this thesis is the class of sets A that are isomorphic to $A \times 2^+$.

2.1.13. Definition. A set A is a *p-cylinder* if there exists an isomorphism $g\colon 2^+ \to 2^+ \times 2^+$ such that

- g and its inverse g^{-1} are computable in polynomial time, and
- for all x we have $x \in A \iff g(x) \in A \times 2^+$.

2.1.14. Example. The entire set $\mathbb{2}^+$ and the empty set are trivial examples of p-cylinders. For a somewhat less trivial example, consider the function f on natural numbers defined as

$$f(n) = \max\{m \mid n \text{ is divisible by } 2^{m-1}\}.$$

Observe that, for all n, we have $f(n) \leq n$, with equality only for $n = 1$ and $n = 2$. Thus, repeated application of f leads to either 1 or 2. Denote the value that we end up with when starting from a number n by $f^*(n)$ and consider the set

$$A = \{\text{asStr}(n) \mid f^*(n) = 1\}.$$

We claim that this set is a p-cylinder. To see why, first observe that, for all n,

$$\frac{n}{2^{f(n)-1}}$$

is odd. This inspires an isomorphism $g\colon \mathbb{2}^+ \to \mathbb{2}^+ \times \mathbb{2}^+$ for which the inverse is given by

$$g^{-1}(\text{asStr}(m), \text{asStr}(n)) = \text{asStr}\left(2^{m-1} \cdot (2n-1)\right).$$

For completeness, we note that g is thus defined as

$$g(\text{asStr}(n)) = \left(\text{asStr}\left(f(n)\right), \text{asStr}\left(\frac{n/2^{f(n)-1}+1}{2}\right)\right).$$

All these expressions can be evaluated in polynomial time. Furthermore, f and g were constructed so that the second criterion in Definition 2.1.13 is met. Hence, A is a p-cylinder.

The set A in the above example is a member of **P**, but there are also p-cylinders outside of **P** [6]. Regardless of whether or not **P** equals **NP**, many **NP**-complete sets are known to be p-cylinders. As a consequence of the Berman–Hartmanis conjecture, it is thus conjectured that all **NP**-complete sets are p-cylinders [see also 80].

The sequential computation thesis gives certain classes of decision problems some independence of the chosen model of computation. Sometimes, we do not care for this independence and choose to work directly with properties of procedures that are specific to our model of computation. One such property we may look at is the precise computation time used by an algorithm. As mentioned before, this computation time, or *running time*, is commonly measured as a function of the length of input instances. For a given function f, we can consider the sets that have a decision procedure of which the running time is bounded from above by f. We say that f is an upper bound on the running time of a procedure ϕ if, for all n, the running time of ϕ on any input of length n is at most $f(n)$.

We shall be interested primarily in the behavior of running time bounds up to a multiplicative constant. Thus, for a function f, we look at sets that have a decision procedure with a running time that is upper bounded by f up to a multiplicative constant. The class of such sets we shall denote by f by $\mathbf{TIME}(f(n))$. This means the following for a set A that is in $\mathbf{TIME}(f(n))$: There is a decision procedure ϕ for A and a constant c such that, for all n, the running time of ϕ on any input of length n is at most $c \cdot f(n)$.

In terms of these classes of sets, we have

$$\mathbf{P} = \bigcup_{c \in \mathbb{N}} \mathbf{TIME}(n^c).$$

Note that in the expression inside the parentheses, we always use n as a variable representing the length of instances. For convenience, we shall often assume that all coefficients and exponents in polynomials are natural numbers. The advantage of this assumption is that it makes the set of polynomials countable and thus easy to work with.

2.1.5 Order Theory

Comparing objects is an essential part of most qualitative analyses. We would like to know whether one thing is perhaps bigger, faster, or better than another. When our objects are natural numbers and the order of interest is the usual less-than-or-equal-to relation, $\leq$, comparing objects is straightforward. In this order, any two objects can be compared. Moreover, the order is a *linear order* [38], because it also is

antisymmetric meaning that if objects a and b satisfy both $a \leq b$ and $b \leq a$, then they are equal, and

transitive meaning that if objects a, b and c satisfy both $a \leq b$ and $b \leq c$, then they also satisfy $a \leq c$.

Not all classes of objects have a meaningful linear order. This is the case, for instance, with classes of functions. We could try to compare two functions based on whether one is *eventually* less than or equal to another. To keep things simple, we assume our functions map natural numbers to natural numbers. Two properties of the natural numbers in particular shall come in handy. They are all positive, and every set of natural numbers contains an element that is less than or equal to all others in the set, a so-called *least element*. We could say that a function f is eventually less than or equal to a function g if we have

$$\lim_{n \to \infty} f(n) \leq \lim_{n \to \infty} g(n).$$

An order defined this way will not be antisymmetric, but, for now, that is not a problem. What is a problem, however, is that not all functions have a limit at infinity. For instance, the limit of a function that oscillates is not defined. In response to this, we could turn to the *limit inferior*, which, for a function $f\colon \mathbb{N} \to \mathbb{N}$, can be defined as

$$\lim_{n\to\infty} \min\{f(m) \mid m \geq n\}.$$

Regardless of the specifics of f, the minimum, $\min\{f(m) \mid m \geq n\}$, is non-decreasing as a function of n. Thus, the limit inferior is defined for oscillating functions too. Still, comparing functions based on their limit inferior may not be satisfactory.

A function like the identity function, that maps every number to itself, does not have a finite limit or limit inferior. We could choose to simply assign infinity to the limit of such functions, but we would then lose all information about the asymptotic behavior of functions. A function that shoots to infinity quickly and one that does so very slowly would have the same limit and limit inferior.

If we want to take into account the asymptotic behavior of functions, one way to compare two functions is by looking at their ratio. We say that a function f *grows much slower* than a function g when we have

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

When f grows much slower than g, we write $f \in \mathrm{o}(g)$. In line with this notation, $\mathrm{o}(g)$ is the class of all functions that grow much slower than g.

Note that we do not have $f \in \mathrm{o}(f)$ for any function f. Indeed, no function grows much slower than itself. A weaker notion is sometimes required, where we only want to express that one function *does not grow faster* than another. For this, we make use of a dual to the limit inferior, the *limit superior* and say that a function f does not grow faster than a function g when we have

$$\lim_{n\to\infty} \max\left\{\frac{f(m)}{g(m)} \;\middle|\; m \geq n\right\} < \infty.$$

When f does not grow faster than g, we write $f \in \mathcal{O}(g)$, where $\mathcal{O}(g)$ is thus the class of all functions that do not grow faster than g. As no function f grows faster than itself, we have $f \in \mathcal{O}(f)$.

For a function f and a constant c, we denote the function that maps a number n to $c + f(n)$ by $c + f$, and the function that maps n to $c \cdot f(n)$ by $c \cdot f$. Let f and g be two functions and c a constant. Observe that if we have $f \in \mathcal{O}(g)$, then we also have $c + f \in \mathcal{O}(g)$ and $c \cdot f \in \mathcal{O}(g)$. Likewise, if we have $f \in \mathrm{o}(g)$, then we also have $c \cdot f \in \mathrm{o}(g)$. The additive case, $c + f \in \mathrm{o}(g)$, is also true, but only because we have excluded 0 from $\mathbb{N}$, which makes it impossible for g to converge to 0. Note that the argument to $\mathcal{O}$ and o is a function and not an expression containing a free variable. However, sometimes we feel it is clearer to write an expression instead. Whenever we do so, we shall use n as a free variable. Thus, $\mathcal{O}(f)$ is the same as

$\mathcal{O}(f(n))$. The latter notation is somewhat unfortunate, as $f(n)$ is suggestive of a particular function value, in which case the latter class would equal $\mathcal{O}(1)$. Yet, the notation $\mathcal{O}(f(n))$ is more in line with the notation used for the class of sets with a running time bounded by a function in $\mathcal{O}(f)$, namely $\mathbf{TIME}(f(n))$.

The does-not-grow-faster-than relation is not antisymmetric. For instance, the functions given by $f(n) = n$ and $g(n) = 2 \cdot n$ are unequal, yet we have both $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(f)$. At the same time, the order *is* transitive. In such cases, it can be beneficial to look at *equivalence classes* instead. For the does-not-grow-faster-than relation, two functions f and g are in the same equivalence class whenever we have both $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(f)$. The order extends to the level of such equivalence classes, and on these equivalence classes the order is antisymmetric, transitive, and also

reflexive meaning that every object a satisfies $a \leq a$.

An order that is reflexive, antisymmetric and transitive is called a *partial order*.

Another example of a partial order, besides the equivalence classes of functions ordered by growth rate we have seen above, is provided by families of sets. The set inclusion order, $\subseteq$, is a partial order on any family of sets. Consider the family of sets

$$\{1\}, \quad \{3\}, \quad \{1,2\}, \quad \{1,2,3\}, \quad \{1,3,4\}, \tag{2.1}$$

for which the inclusion order is depicted in Figure 2.1. This family consists of subsets of $\{1,2,3,4\}$. While it does not include all subsets of $\{1,2,3,4\}$, any subset can be *covered* by sets from this family.

2.1.15. Definition. A family of sets η is a *cover* of a set A if it satisfies

$$A \subseteq \bigcup_{S \in \eta} S.$$

The family of sets (2.1) has no least element, nor does it have a *greatest element*, which would in this case be a set that contains all others of the family. It does, however, contain two *maximal elements*, sets that are not contained in any set of the family except themselves. Families of sets where every two sets are contained in another are of particular interest to us [see also 1, 38].

$$\begin{tikzcd}
\{1,2,3\} & \{1,3,4\} \\
\{1,2\} \arrow[u] & \\
\{1\} \arrow[u] \arrow[uu, bend left] \arrow[uur] & \{3\} \arrow[uul] \arrow[uu]
\end{tikzcd}$$

Figure 2.1: Set inclusion as a partial order on a family of sets. Arrows connecting each set to itself have been omitted.

2.1.16. Definition. A family of sets η is *directed* if for every two sets S_1 and S_2 in η there is a set $S_{1,2}$ in η that satisfies

$$S_1 \subseteq S_{1,2} \qquad \text{and} \qquad S_2 \subseteq S_{1,2}.$$

For sets S_1 and S_2 in a family of sets, a set that contains both S_1 and S_2 is known as an *upper bound* of the two sets. Dually, a *lower bound* of the two sets is a set that is contained in both of them. An upper bound of two sets S_1 and S_2 that is included in all other upper bounds of S_1 and S_2 is a *least upper bound.* Likewise, a lower bound that includes all other lower bounds is a *greatest lower bound.* A family of sets does not necessarily contain all least upper bounds and greatest lower bounds. For example, the family of sets (2.1), does not contain a least upper bound of the sets $\{1\}$ and $\{3\}$, or of the sets $\{1, 2, 3\}$ and $\{1, 3, 4\}$. By contrast, the family of sets that consists of *all* subsets of $\{1, 2, 3, 4\}$ does contain all least upper bounds and greatest lower bounds.

2.1.17. Definition. A collection of objects $\mathscr{L}$ with a partial order is a *lattice* if every two objects of $\mathscr{L}$ have a greatest lower bound and a least upper bound.

For any two objects a and b of a lattice, we denote the least upper bound of a and b by $a \vee b$. The greatest lower bound of a and b is denoted by $a \wedge b$. Most pleasant to work with are lattices where these two operations interact nicely [38].

2.1.18. Definition. A lattice $\mathscr{L}$ is *distributive* if every three objects a, b, c of $\mathscr{L}$ satisfy

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c).$$

Observe that any finite lattice, such as the family of subsets of $\{1, 2, 3, 4\}$, contains a least element and a greatest element. An infinite lattice does not need to contain these elements. The family of all *finite* subsets of the natural numbers ordered by set inclusion is an example of an infinite lattice that does not contain a greatest element. On the other hand, the family of *all* subsets of the natural numbers is an infinite lattice that contains both a least element and a greatest element.

2.1.19. Definition. A lattice is *bounded* if it has a least element and a greatest element.

If the order on a lattice codifies a relation like “bigger”, “faster”, or “better”, then certainly the greatest element of a bounded lattice is “big”, “fast”, or “good”. More generally, if any object of a lattice is “big”, “fast”, or “good”, then all “bigger”, “faster”, or “better” objects are too. This intuition guides us to a formalization of what it means to be “big”, “fast”, or “good”.

2.1.20. Definition. Let $\mathscr{L}$ be a lattice with an order indicated by $\preccurlyeq$. A subset $\mathscr{F}$ of objects of $\mathscr{L}$ is *upward closed* if, for all $a \in \mathscr{F}$ and $b \in \mathscr{L}$ satisfying $a \preccurlyeq b$, we have $b \in \mathscr{F}$.

Suppose that two objects, a and b, of a lattice are in some sense "good". Furthermore, suppose that whatever it is that makes these objects "good" is unique in the sense that this same property is present in both a and b. Intuitively, this "goodness" property must then be present in the greatest lower bound, $a \wedge b$, as well. This leads to the following definition of subsets of a lattice that may represent a notion of being "big", "fast", or "good".

2.1.21. Definition. A nonempty subset of objects of a lattice is a *filter* if it is upward closed and closed under taking greatest lower bounds.

It is possible that the essence of what makes an object of a lattice "good" can be found in a single object, a, of the lattice. In that case, a is a least element in the filter of "good" objects. This filter is then the set of all objects in the lattice that are better than, or equally good as a.

2.1.22. Definition. Let $\mathscr{L}$ be a lattice with an order indicated by $\preccurlyeq$. A filter $\mathscr{F}$ in $\mathscr{L}$ is *principal* if there is an object $a \in \mathscr{L}$ such that we have

$$\mathscr{F} = \{b \mid b \in \mathscr{L} \text{ and } a \preccurlyeq b\}.$$

If a lattice contains a greatest element, then it has a trivial principal filter consisting of just this greatest element. Let $\mathscr{L}$ be the family of subsets of natural numbers, ordered by set inclusion. A filter in $\mathscr{L}$ can be thought of as a collection of sets that are somehow "big enough". In this lattice, the trivial principal filter, $\{\mathbb{N}\}$, conveys the idea that only the biggest set is big enough. Another filter, that is not principal, is the filter of subsets of $\mathbb{N}$ with a finite complement. This is the *cofinite filter*, or *Fréchet filter*, on $\mathbb{N}$. Its interpretation is most clearly demonstrated via quantifiers [110]. For a predicate on natural numbers, P, the statement

$$\forall n\colon P(n) \tag{2.2}$$

means that the set $\{n \mid P(n)\}$ contains all natural numbers. Equivalently, (2.2) states that the set $\{n \mid P(n)\}$ is one of the "big" sets in the trivial filter in $\mathscr{L}$,

$$\{n \mid P(n)\} \in \{\mathbb{N}\}.$$

If our notion of a "big" set is that represented by the cofinite filter, we use the $\forall^\infty$-quantifier. Thus,

$$\forall^\infty n\colon P(n) \tag{2.3}$$

means that $\{n \mid P(n)\}$ is in the cofinite filter. In other words, (2.3) states that P is true of all but finitely many natural numbers n.

The dual of this quantifier, $\exists^\infty$, is defined by

$$\exists^\infty n\colon P(n) \iff \neg\forall^\infty n\colon \neg P(n).$$

In other words, $\exists^\infty n\colon P(n)$ states that there exist infinitely many numbers n of which P is true. For this quantifier, the relation between its notation and its meaning is somewhat more intuitive than for $\forall^\infty$.

We remark that we can use the $\forall^\infty$-quantifier in an alternative definition of our grows-much-slower-than and does-not-grow-faster-than relations. For functions f and g, we have $f \in \mathrm{o}(g)$ precisely when we have

$$\forall c\colon \forall^\infty n\colon f(n) < \frac{1}{c} \cdot g(n).$$

Likewise, $f \in \mathcal{O}(g)$ means the same as

$$\exists c\colon \forall^\infty n\colon f(n) \leq c \cdot g(n).$$

2.2 A New Framework

The two prominent frameworks for a parameterized analysis of complexity are those of Downey and Fellows and of Flum and Grohe, introduced in Section 1.3. Both are rooted in computational complexity theory and they each have their own theoretical shortcomings. We shall build a more universal framework and resolve the shortcomings of these earlier frameworks that we have identified in Section 1.3.

Several forms of complexity that could be related to parameters of instances were listed in Section 1.2. In the next chapter, we shall use our new framework in a more concise analysis of these forms of complexity. This shows that our framework can be applied outside the theory of computational complexity. Indeed, our framework serves as a mathematical model of complexity that is applicable to a broad range of complexities. Still, all of our notions of complexity are involved with computation in one way or another. Our framework is therefore not limited to a notion of a *parameterization*. It also includes a notion of a *parameterized procedure*, which is a computation that takes as input not just an instance, but also a parameter value.

Desiderata

The picture that emerges from Section 1.2 is that a parameter of a problem instance is any of a collection of related properties the object may have. For instance, a graph may have any number of vertices, hence the number of vertices in a graph is a parameter of the graph. Another parameter of a graph would be its number of edges. This corresponds to the property schema that we have seen in Example 1.2.6. Each specific property we shall refer to as a *parameter value*. Hence, if the number of edges is the parameter at hand, then '9' is a possible parameter value, expressing that a graph has 9 edges. A *parameterization*, then, should link an instance x to the parameter values that apply to x. Note that we shall use the term *parameter* loosely so that it encompasses the various meanings it has in the literature. The terms *parameter value* and *parameterization* will be made precise shortly and we shall never use them informally. By making parameterizations *independent* of decision problems, we resolve one of the two shortcomings we identified in the framework of Downey and Fellows. Namely, a parameterized analysis of the complexity of classical decision problems will be possible.

In our framework, we want to relate notions of complexity to parameter values. In order for every instance to have a complexity assigned to it, we want parameterizations to link each instance to at least one parameter value. We summarize this desire by saying that we want parameterizations to be, in some sense, *total*. Wanting to have a complexity assigned to all instances can be motivated by an engineering view on taking measurements. We feel that if instances are bigger than your measuring device, you need a bigger measuring device. In our case, we want to use parameterizations as a measuring device for measuring complexity. Requiring parameterizations to be total makes sure that we shall be able to deal with nonmembers of decision problems. This resolves the other shortcoming we identified in the framework of Downey and Fellows.

Additionally, we desire parameterizations, and in particular parameter values, to be *general*. We have seen that if we restrict each parameter value to be a natural number, as with Flum and Grohe, the interplay between parameter values cannot be expressed. In our framework, we want to support even parameters where the parameter values are not ordered linearly. However, as we shall see, we do well to put in place at least some requirements regarding an order on parameter values.

Parameter values naturally have a stronger-than order given by set inclusion: If the set of instances that have a property P is included in the set of instances that have property Q, then property P is stronger than property Q. In our graph example, we need to generalize the properties we look at somewhat to make this order visible. The set of graphs that have 4 vertices is disjoint from the set of graphs that have 5 vertices. However, the set of graphs that have *at most* 4 vertices is a subset of the set of graphs that have *at most* 5 vertices. In that sense, the former property is stronger than the latter. This order works for more diverse collections of parameter values as well. For example, every graph that has at most 4 vertices has at most 9 edges. Thus, again, the former is a stronger property than the latter. Conversely, the property of having at most 9 edges is weaker than the property of having at most 4 vertices. At the same time, the property of having at most 9 edges is incomparable to the property of having at most 5 vertices. The full graph with 5 vertices has 10 edges, and an empty graph with 6 vertices has no edges.

Let us take a look at what this order can mean for computations that involve parameters. Suppose we have some function that we can compute on graphs with a given property, say graphs that have at most 4 vertices. Often, this function can be *extended* to cover graphs with a weaker property, say graphs that have at most 5 vertices. If we keep on extending our function to be applicable to graphs with more and more vertices, we approach a function that works on all graphs. We desire our parameterizations to be *consistent* in that the function we end up with does not depend on the way we go from property to property.

2.2.1. Example. Consider the following two parameters concerning a graph G, where e and w represent parameter values corresponding to each of the parameters.

1. The number of edges in G is at most e.

2. The number of `1`s in the representation of the number of vertices in G as a binary string is at most w.

Admittedly, the second parameter is rather contrived, but its behavior with respect to the first parameter is of interest. For no values of e and w is either parameter weaker than the other. However, for both parameters we find that if we keep increasing the parameter value, we will eventually include any graph.

Now, suppose we want to build a function by defining it on graphs with a certain property and then iteratively extending it to weaker properties. Once we have chosen one of our two parameters, we must stick to that choice. This is so because, as we saw, for no values of e and w is either parameter weaker than the other. As a result, nothing stops us from building completely different functions depending on what parameter we choose to start with.

We want to prevent situations such as in the example above. Concretely, we want to disallow parameterizations that incorporate just the two parameters mentioned in the example, and nothing else. One way to achieve consistency, is by requiring that for every two parameter values, there is another that is weaker than both.

Definitions

In our framework, we want to allow parameter values to have all sorts of structure. To make this possible, we choose to make as few assumptions about parameter values as necessary. One thing we shall assume is that in any parameterized context, the set of possible parameter values is countable. This assumption has the effect that parameter values can be encoded as binary strings and can be processed algorithmically. In fact, we shall leave any interpretation of parameter values to the user of a parameterization and proceed as if parameter values *are* binary strings. Thus, we take care of the desire to make parameterizations, and parameter values in particular, as *general* as possible.

This leaves us with three desiderata to consider when defining parameterizations. Namely, a parameterization should be *independent*, *total* and *consistent*. The first of these can be met simply by avoiding any dependence on a decision problem in the definition of a parameterization. For the other two, we look at the sets of instances that each parameter value identifies. If a parameter value, a string k, represents the property of having at most 4 vertices, then k identifies the set of graphs with at most 4 vertices. This suggests defining a parameterization as a family of sets indexed by parameter values.

2.2.2. Definition. A *parameterization* is a directed cover of 2^+, indexed by 2^+. The elements of a parameterization are called its *slices*.

By requiring a parameterization to be a cover of 2^+, any instance is present in at least one slice of a parameterization. Thus, this definition of parameterizations satisfies our desire of parameterizations being *total*, The requirement that a

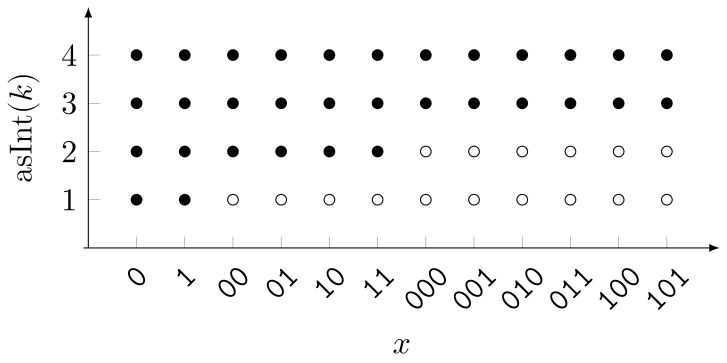


Figure 2.2: An initial segment of the length parameterization. Filled marks indicate members of the slices of the parameterization, whereas open marks indicate nonmembers. Note that the elements of the parameterization, depicted as rows in the figure, are finite sets. Furthermore, the inclusion order on the elements of the parameterization matches the natural enumeration order on the set 2^+ of parameter values. Neither of these two properties is required to hold for parameterizations in general.

parameterization is directed makes sure that situations as in Example 2.2.1 cannot occur and we have a form of *consistency*.

For strings, the most-used parameter in computer science must be the *length*. This parameter can be represented by a parameterization.

2.2.3. EXAMPLE. The *length parameterization* [see also 57, Example 1.6] is defined as

$$(\{x \mid |x| \leq \text{asInt}(k)\})_{k \in 2^+}.$$

In this parameterization, depicted in Figure 2.2, the parameter value k acts as an upper bound on the lengths of instances. We remark that when instances are not just strings, but more structured objects such as graphs, a "length" is only defined in light of an encoding.

Parameterizations are intended to also measure structures of instances other than their lengths. Note that the length of a string is commonly associated with the symbol n. To highlight the role of parameterizations as more general measures of structure, we use a similar-looking symbol for parameterizations: η. By convention from parameterized complexity theory, we shall mostly use x for instances and k for parameter values. The slice of a parameterization η that corresponds to a parameter value k is denoted by η_k.

In our framework, parameter values are mapped to sets of instances. In this regard, our framework differs from that of Flum and Grohe [57], where instances are mapped to parameter values. Of course, given a parameterization η and an instance x, we can consider the set $\{k \mid x \in \eta_k\}$. These are the parameter values that the parameterization η links to the instance x. Contrary to Flum and

Grohe, we make no demands regarding the computability of such a mapping from instances to parameter values.

Recall that parameter values in the frameworks of Downey and Fellows [44] and of Flum and Grohe [57] are typically natural numbers. This allows us to compare parameter values assigned to an instance by different parameterizations using the usual less-than-or-equal-to order [95]. In our more general framework, where instances are associated with sets of binary strings, comparing parameterizations is less straightforward. To facilitate a comparison between parameterizations, we shall look at the length, in bits, of the shortest parameter value for some given instance.

2.2.4. Definition. Given a parameterization η, the *minimization function* of η is defined as

$$\mu_\eta(x) = \min\{|k| \mid x \in \eta_k\}.$$

Note that μ_η minimizes with respect to the length of parameter values and not with respect to the inclusion order on the slices of the parameterization.

Like in the framework of Downey and Fellows [44], in our framework an instance can be associated with multiple parameter values. Therefore, a parameterized analysis of decision problems requires a generalization of decision procedures. A parameterized decision procedure, or *parameterized procedure* for short, is a special kind of procedure that takes two arguments. In line with ordinary decision procedures, we require parameterized procedures to be total, meaning that their computation terminates on all possible inputs. However, we do not require the output of a parameterized procedure to be either `1` or `0`, representing the judgments 'yes' and 'no'. Any other output a parameterized procedure produces we interpret as `?`, representing the judgment 'unknown'. In summary, a parameterized procedure is a procedure that takes two strings as input and produces an output in the set $\{\mathtt{1}, \mathtt{0}, \mathtt{?}\}$.

We would like to associate parameterized procedures to sets in a way similar to how decision procedures are associated to sets. In the presence of parameterizations, this is possible for some parameterized procedures.

2.2.5. Definition. A parameterized procedure ϕ *converges* to a set A on a parameterization η if, for all strings x and k, we have

$$x \in \eta_k \implies \phi(x, k) = A(x).$$

A parameterization η is consistent in the sense that it is directed. Therefore, when a parameterized procedure ϕ converges to a set A on η, we can think of A as a *limit* of ϕ. However, the set to which a parameterized procedure converges may depend on the parameterization.

2.2.6. Example. Consider the parameterizations η and ζ given by

$$\eta_k = \begin{cases} 2^+ & \text{if } k = 0, \\ \emptyset & \text{otherwise,} \end{cases} \qquad \text{and} \qquad \zeta_k = \begin{cases} \emptyset & \text{if } k = 0, \\ 2^+ & \text{otherwise,} \end{cases}$$

and the parameterized procedure ϕ given by

$$\phi(x, k) = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise.} \end{cases}$$

These definitions are so that ϕ converges to 2^+ on η, but to $\emptyset$ on ζ.

The above example shows that a parameterized procedure can have multiple limits. One way to enforce the limit of a parameterized procedure to be unique is by strengthening our notion of convergence. We do so by turning to a special kind of parameterizations.

2.2.7. Definition. A parameterization η is *point-cofinite* if each instance $x \in 2^+$ is excluded from only finitely many slices of η.

The length parameterization of Example 2.2.3 is an example of a point-cofinite parameterization. With point-cofinite parameterizations, a situation as in Example 2.2.6 cannot occur. A parameterized procedure ϕ may converge on two different point-cofinite parameterizations, but the set to which ϕ converges will be the same either way. In other words, as far as convergence on point-cofinite parameterizations is concerned, the limit of a parameterized procedure, if it exists, is unique. Therefore, when dealing with convergence in the context of point-cofinite parameterizations, we need not mention a point-cofinite parameterization explicitly. We may simply say that a parameterized procedure ϕ converges to a set A. This then means that there exists a point-cofinite parameterization η such that ϕ converges to A on η. The existence of a set A to which a parameterized procedure converges is is hence a property of the parameterized procedure alone.

2.2.8. Definition. A parameterized procedure ϕ is *convergent* if there is a set to which it converges. In more detail, this means ϕ is convergent if there is a set A and a point-cofinite parameterization η such that ϕ converges to A on η.

Suppose a parameterized procedure ϕ converges to a set A on a point-cofinite parameterization η. Because η is point-cofinite, the set A does not depend on the specifics of η, but only on the fact that it is point-cofinite. This does not mean that whenever for some instance x and parameter value k we have $\phi(x, k) = 1$, we can conclude that x is a member of A. Likewise, whenever we have $\phi(x, k) = 0$, we cannot conclude that x is not a member of A. All we know is that for each instance x there are only finitely many parameter values k such that $\phi(x, k)$ is not in agreement with membership of x in A. In other words, we do not *know* when the output of ϕ is correct, only that it will eventually be correct.

An alternative way to enforce that a parameterized procedure has only one limit set, is by requiring the procedure to know when it is correct. A parameterized procedure ϕ knows when it is correct if there is a set A such that, for all x and k, the output of $\phi(x, k)$ is either ? or $A(x)$. In order for this set A to be unique, we require that the procedure ϕ must converge to A on some parameterization η. Because the output of ϕ is never wrong, we can derive a family of sets from it that could potentially serve as such a parameterization η.

2.2.9. Definition. A parameterized procedure ϕ is *direct* if

$$(\{x \mid \phi(x,k) \neq ?\})_{k \in 2^+}$$

is a parameterization on which ϕ converges to some set A.

A parameterized procedure can be convergent, direct, neither, or both. In the parameterized version of computational complexity theory, parameterized procedures that are both convergent and direct often emerge quite naturally.

2.2.10. Example. In Section 1.3, we have looked at a parameterized analysis of the Clique problem,

$$\text{Clique} = \{(G, l) \mid \text{there is a set of at least } l \text{ vertices of the graph } G \text{ in which each pair of vertices is connected by an edge}\},$$

with respect to the minimum vertex cover size. We found that in the framework of Downey and Fellows, such an analysis was not very natural: It required the construction of an intricate parameterized decision problem where the instances of the form (G, l) were replaced by structures of the form $\langle (G, l), k \rangle$.

In our framework, we can consider the unmodified Clique problem with respect to the point-cofinite vertex cover parameterization

$$\eta = (\{(G, l) \mid G \text{ has a vertex cover of at most asInt}(k) \text{ vertices}\})_{k \in 2^+}.$$

In the framework of Downey and Fellows, the Clique problem parameterized by the minimum vertex cover size was found to be fixed-parameter tractable. For our framework, this means that there is a direct parameterized procedure ϕ such that

- ϕ converges to Clique on η, and
- for some function f and constant c, the running time of ϕ on any instance of length n and parameter value k is at most $f(k) \cdot n^c$.

Note that because η is point-cofinite, this procedure ϕ is not only direct, but also convergent.

Thus, an analysis of fixed-parameter tractability can be performed in our framework. Moreover, this is possible without resorting to intricate parameterized decision problems and the parameterization can be reused for other problems. We remark that a more fundamental definition of fixed-parameter tractability will be given in Section 3.2.1.

When the parameter value is held fixed, the behavior of a direct parameterized procedure can be thought of as an approximation of a decision procedure. Indeed, such procedures are of interest also without any parameter values playing a part.

2.2.11. Definition. Given a function t, a procedure ϕ is a *t-approximation* for a set A if it satisfies, for every string x,

- $\phi(x)$ outputs either ? or $A(x)$, and
- ϕ terminates on input x within $t(|x|)$ steps.

A t-approximation ϕ is said to *decide* the elements of its *domain*

$$\mathrm{dom}(\phi) = \{x \mid \phi(x) \neq \texttt{?}\}.$$

The *polytime-approximations* [93, 18] are t-approximations where t is a polynomial. Likewise, a procedure is an *$\mathcal{O}(f)$-approximation* if it is a t-approximation for some $t \in \mathcal{O}(f)$. The fact that the bound is a bound on the running time is left implicit. For time-constructible functions t, the domain of a t-approximation is decidable in time t. In particular, the domain of a polytime-approximation is in **P**, without an increase in the degree of the polynomial. The domain of a polytime-approximation is also known as the *definite part* of the approximation [93].

Chapter 3

Parameterizations

The main results of this thesis are obtained in the current chapter. In the following sections, we seek to analyze a selection of notions of complexity in a unified, parameterized, way. This analysis is carried out using the framework developed in Section 2.2. We look at complexity as it is used in *computability theory*, in *computational complexity theory*, in *algorithmic complexity theory*, in *algorithmic statistics*, and in the study of *computational redundancy*. In the first four of these fields, complexity expresses a distance from some reference notion of "simplicity". With the last of these fields, computational redundancy, it is the other way around and complexity is the reference with respect to which simplicity is measured.

In computability theory, the objects of study are decision problems, modeled as sets. A set has "negligible complexity", if it is decidable. We avoid the word simplicity here, because in the context of computability theory, the term *simple set* means something entirely different. The focus, then, is on how undecidable a given set with non-negligible complexity is. Section 3.1 rephrases one possible way to answer this question in terms of parameterizations. Our parameterized analysis of computability provides a new insight concerning parameterized computational complexity. We find an answer to the question: *what sets are fixed-parameter tractable?* More precisely, we characterize the sets for which a parameterization exists with respect to which they are fixed-parameter tractable.

Like computability theory, computational complexity theory deals with decision problems modeled as sets. Simplicity in computational complexity theory is routinely identified with decidability in polynomial time. Parameterized computational complexity theory is thus a way to assess how far a set is removed from being decidable in polynomial time. In effect, parameterized computational complexity theory offers a way to assess the computational complexity of individual problem instances. However, the current literature on parameterized computational complexity offers little in the way of a comparison of parameterizations. Given two parameterizations with respect to which some specific set is fixed-parameter tractable, we may want to know which of the two is "better". An order on parameterizations is developed in Section 3.2. This order is rich in structure. Moreover, we are able to show that for most sets, there is no "best" parameterization with respect to which they are fixed-parameter tractable.

Besides looking at how much computation time is needed for deciding on membership in a set, we can look at the required length of a decision procedure. Some decision procedures can be expressed as an algorithm of only a few lines, while others necessarily use very many lines. This take on complexity, where a set is simple if it has a short decision procedure, is at the heart of algorithmic complexity theory. A parameterized treatment of this form of complexity is the topic of Section 3.3. This parameterized treatment is shown to be more nuanced than the traditional treatment, as it can take considerations regarding uniformity into account. Because we use a unifying framework for the analysis of both computational and algorithmic complexity, the interplay between the two notions can be studied. This allows us to show that, at the level of individual instances, high algorithmic complexity implies high computational complexity.

An application of reasoning along the lines of algorithmic complexity can be found in statistics. One of the central themes in statistics is model selection. In model selection, inferences about the nature of a statistical process are made on the basis of a data sample taken from that process. When judging the likelihood of a statistical model in a set of candidate models, the number of variables included in the model should be taken into account. A model with very many variables can potentially be tuned to match almost any data sample. In that regard, a model should not be too complex. Correspondingly, a model is simple if it offers few possibilities for tweaking, or, in terms of algorithmic complexity, if it has a short description. This algorithmic approach to statistics is faced with the challenge of deciding what part of the information in a data sample is relevant for model selection. In our parameterized algorithmic statistics of Section 3.4, we conclude that this "useful information" is context dependent.

The study of computational redundancy brings us back to decision problems. Previously, we asked what part of the information in an object is relevant for model selection. Now, we ask what part of the information in an object is, in some sense, irrelevant for deciding on membership of an instance in a given set. Observe that the complexity of an object increases as the size of this irrelevant part decreases. Thus, computational redundancy expresses a kind of "anti-complexity". The reference against which we measure computational redundancy is the information in an object. An object is simple when much of its information is computationally redundant. For a simple object, the length of a description of the object is not a good measure of its computational complexity. As it turns out, there are many ways in which an object with a long description could be reduced to objects with shorter descriptions. Multiple possible ways are explored in Section 3.5. How much computational redundancy can be extracted from the description of an object depends on the specifics of the reduction under consideration.

Apart from the analysis of several notions of complexity, as summarized in

Field, §	*Reference*	*Question*
Computability Theory, 3.1	Decidability	What sets are fixed-parameter tractable?
Computational Complexity Theory, 3.2	Decidability in Polynomial Time	Is there a best parameterization?
Algorithmic Complexity Theory, 3.3	Short Decision Procedures	How does algorithmic complexity relate to computational complexity?
Algorithmic Statistics, 3.4	Simple Models	What part of the information in an object is useful information?
Computational Redundancy, 3.5	Object Description Length	How much computational redundancy can be extracted from the description of an object?

Table 3.1: Notions of complexity in different fields of research. Each notion expresses a distance from some reference measure of simplicity. For computational redundancy, the role of the reference measure is inverted, and it is simplicity that is expressed as a distance from the reference measure. Our parameterized analysis of complexity addresses questions related to these different notions of complexity.

Table 3.1, we study parameterizations themselves. The first three sections of the current chapter climb a ladder of abstractions away from the complexity of individual instances. In Section 3.1, we go from grouping instances of comparable complexity to considering collections of such groupings. More specifically, we move from a slicewise look at complexity to the analysis of parameterizations as collections of slices. The subsequent section, Section 3.2, takes this one step further, into the analysis of algebraic structures, filters, comprised of parameterizations. Finally, Section 3.3 is effectively concerned with the lattice of such filters. It relates operations on sets to changes in the corresponding filter of parameterizations.

The results of the study of parameterizations in these three sections can be summarized as a nice structural progression. In Section 3.1, we fix a parameterized complexity class $\boldsymbol{C}$, and look at those sets A for which there is a parameterization η such that A is in $\boldsymbol{C}$ with η. We conclude that doing so is not very fruitful and we need to keep track of parameterizations explicitly. In Section 3.2, we fix a parameterized complexity class $\boldsymbol{C}$ and a set A, and look at those parameterizations with which A is in $\boldsymbol{C}$. Doing so, we obtain collections of parameterizations that exhibit an algebraic structure. In Section 3.3, we fix a parameterized complexity class $\boldsymbol{C}$ and a collection of parameterizations $\mathcal{F}$, and look at those sets A for which $\mathcal{F}$ is the collection of parameterizations with which A is in $\boldsymbol{C}$. We conjecture that the sets thus obtained are precisely the sets that have, in some specific way, the same distribution of computational complexity.

3.1 as Limit Computability

Computability theory [131] is one of the pillars of mathematical logic. In formal systems, and in particular in proof calculi, questions regarding the decidability of the set of theorems are of central importance [90, 143]. At the same time, computability theory can be seen as a theory of computational complexity with complete disregard for resource usage.

A parameterized study of decidability was started already in 1965 [124, 66], some thirty years before a parameterized investigation of computational complexity took off. In this section, we shall recast some of the classical results in an explicitly parameterized framework, noting the new insights thus obtained. The central idea is that a hierarchy of undecidability, the difference hierarchy, can be used to characterize the structure of undecidability inside sets. Not all instances of an undecidable set are equally responsible for the severity of the set's undecidability. Unsurprisingly, the use of parameters for the analysis of undecidability leads to questions surrounding the computability of the parameters themselves. Some of our insights on this front will be relevant to forms of complexity other than decidability.

Synopsis

A set is of negligible complexity in computability theory if it is decidable. In Section 3.1.1, we look at the use of parameterizations for expressing undecidability as a form of complexity. Ideally, a parameterized analysis of computability would allow us to distinguish between different degrees of undecidability. We find that unless some undecidability is permitted in the parameterizations that we take into account, our framework can only deal with decidable sets. In fact, even when we allow for semidecidability in our parameterizations, our framework still only handles decidable sets.

In Section 3.1.2, we shift our focus to the behavior of convergent parameterized procedures that do not converge on any *decidable* point-cofinite parameterization. The type of behavior we look at generalizes semidecidability and makes a parameterized study of computability possible also for sets that are not decidable. The measure of complexity that arises is shown to be closely related to the difference hierarchy and, more generally, to computability in the limit.

For our parameterized analysis of undecidable sets, we were forced to take into account undecidable parameterizations. However, some control of the undecidability of parameterizations is possible. In Section 3.1.3, we show that the reach of parameterized computability analysis can be tied in with types of reducibility to the halting set. The types of reducibility we discuss are truth-table reducibility and Turing reducibility. Together with the possibility of restricting to decidable parameterizations, this gives us three possible scopes for parameterized computability analysis. The two extremes, decidable sets and sets Turing reducible to the halting set, are connected to the existing frameworks for parameterized complexity theory. This provides a new perspective on these frameworks, and shows precisely how that of Flum and Grohe is more restrictive than that of Downey and Fellows.

The different degrees of undecidability of parameterizations is exploited in Section 3.1.4. They are used to show the existence of a set that is far from semidecidable, yet at the same time exhibits some structure. Given two instances, it is easy to decide which of the two is more likely to be a member of the set. This result is not new, but our parameterized framework enables a proof that is much more transparent than the known proof.

Most of the results in this entire section are refinements of the work by Witteveen and Torenvliet [154]. Their early versions of the ideas on a parameterized exploration of computability were not yet part of an overarching theory, as is the case in this thesis. Broadly, we can conclude from our parameterized exploration of computability that parameterized complexity needs to explicitly keep track of parameterizations. We must not try to classify sets based on whether or not a parameterization *exists* with respect to which the set has certain properties. Otherwise, we would find that the properties are irrelevant: The entire classification would depend solely on the undecidability allowed of parameterizations and the degree of undecidability of the set at hand.

3.1.1 Decidable and Semidecidable Parameterizations

Arguably the most elementary parameterizations are decidable parameterizations. With *decidable*, we mean that there is a procedure that, when given some input $(x, k) \in 2^+ \times 2^+$, decides whether slice k of the parameterization includes x. From a computability perspective, such parameterizations are fairly uninteresting.

3.1.1. THEOREM. *The following statements about a set A are equivalent.*

1. *A is decidable.*
2. *There is a direct parameterized procedure converging to A.*
3. *There is a parameterized procedure converging to A on a decidable parameterization.*

Proof:
$1 \implies 2$. We can adapt a decision procedure for A so that it is a direct parameterized procedure converging to A. This is done by simply ignoring the second input, the parameter value. The resulting parameterized procedure correctly decides membership in A regardless of the parameter value. Corresponding to this parameterized procedure is the parameterization that consists only of copies of the full set 2^+.

$2 \implies 3$. More generally, the parameterization corresponding to any direct parameterized procedure that converges to A is decidable. To decide whether some given x is in some slice k of a parameterization, we may use the parameterized procedure. If the procedure outputs `?`, then slice k of the parameterization does not include x, otherwise it does.

$3 \implies 1$. Because a parameterization is a cover, for every instance x, there is a parameter value k such that slice k of the parameterization includes x. Given an instance x, such a k can be found by running the decision procedure for the parameterization on $(x, \mathrm{asStr}(1))$, $(x, \mathrm{asStr}(2))$, $(x, \mathrm{asStr}(3))$, and so on, until we find an accepted pair. Membership of x in A can then be decided by running the parameterized procedure on the pair thus found. □

In other words, with respect to parameterized procedures, the decidable parameterizations characterize the decidable sets. Note that the decidability requirement for decidable parameterizations is uniform in the parameter k: A single decision procedure has to be available that works for all values of k. Therefore, the only class we could reasonably call *uniformly fixed-parameter decidable* is the class of decidable sets. As far as our parameterized analysis of decidability is concerned, this is not very interesting yet.

A naive broadening of our scope does not get us anywhere new. Calling a parameterization *semidecidable* when its slices are semidecidable uniformly in the parameter value only leads to an extension of Theorem 3.1.1. Recall that “uniformly in the parameter value” means that the semidecidability of the slices is witnessed by a single procedure that takes two arguments. The partial application of this procedure to a given parameter value yields a witness of the semidecidability of the corresponding slice.

3.1.2. THEOREM. *The statements about a set A of Theorem 3.1.1 are equivalent to*

4. There is a parameterized procedure converging to A on a semidecidable parameterization.

Proof:
$3 \implies 4$. Since a decidable parameterization is also a semidecidable parameterization, this is immediate.

$4 \implies 1$. As in the proof of ($3 \implies 1$), it suffices to show that a parameter value k can be found such that slice k of the parameterization includes x. To see that this is indeed the case, let η be a semidecidable parameterization and consider the set

$$\{(x, k) \mid x \in \eta_k\}.$$

Because η is semidecidable uniformly in the parameter value, this set is semidecidable. Equivalently, there is a procedure that enumerates this set. As in the proof of ($3 \implies 1$), the key insight is now that η is a cover. This means that we can run the enumeration and simply wait until a pair $(x_i, k_i),\ i \in \mathbb{N}$ comes along for which we have $x_i = x$. The corresponding parameter value k_i is such that the slice of η that is associated with k_i includes x. □

The parameterization corresponding to a direct parameterized procedure is necessarily decidable and cannot be merely semidecidable. This is a consequence of the fact that parameterized procedures are total. While they may output `?` instead of either `1` or `0`, they are not permitted to run forever on any input. It follows that classes of sets in parameterized complexity theory must be closed under taking complements. For every parameterized procedure that converges to a set on some parameterization, there is a parameterized procedure that converges to the complement of that set on the same parameterization. The derived procedure simply retains the output if it is `?` and flips it if it is either `1` or `0`.

Furthermore, it is possible to bound the number of steps taken by a parameterized procedure by a function of the parameter value alone. That is, restricting to resource-bounded parameterized procedures does not affect the class of sets that occur as limits with respect to decidable parameterizations. The following theorem is stated in terms of a specific running time bound, but can be modified for other types of resources and for slower-growing bounds.

3.1.3. Theorem. *For every decidable set there is a direct parameterized procedure that converges to it and takes a number of steps that only depends on the parameter value. On an input (x, k), the procedure terminates within $\mathcal{O}(|k|)$ steps, with the hidden constant not depending on the set at hand.*

Proof:
Let A be a decidable set and ϕ a decision procedure for A. We define a direct parameterized procedure that converges to A and meets the running-time requirement of the theorem. On input (x, k), the procedure simulates ϕ for up to $|k|$ steps. If the simulation terminates, the procedure outputs the result of the simulation. Otherwise, it outputs `?`.

Let η be the parameterization associated with the direct parameterized procedure defined above. For every two parameter values k, k' such that $|k| \leq |k'|$ holds, we have $\eta_k \subseteq \eta_{k'}$. Also, for every instance x, the number of steps needed by the decision procedure is finite. Therefore, for every instance x there will be a parameter value k such that x is included in slice η_k. Hence, our procedure meets the requirements of the theorem. We remark that η is point-cofinite. Our parameterized procedure is therefore not only direct, but also convergent. □

When resource bounds are of interest, it may be natural to restrict attention to decidable parameterizations. Indeed, parameterized complexity theory as developed by Flum and Grohe [57] demands decidability of parameterizations.

More specifically, in their framework, parameter values must be computable. By Theorem 3.1.3, the existence of a decidable parameterization with which a set A is in a given parameterized complexity class tells us very little about A.

3.1.4. Corollary. *Suppose we are working in a parameterized framework where parameter values must be computable. Let* $\boldsymbol{C}$ *be a class of sets that are decidable in some given parameterized time bound. A set* A *is in* $\boldsymbol{C}$ *with respect to some parameterization if and only if* A *is decidable.*

This rather informal corollary follows from Theorem 3.1.3 because of the way complexity classes in parameterized complexity theory are constructed. Typically, resource bounds in parameterized complexity theory allow for arbitrary resource usage as a function of the parameter value.

3.1.5. Example. Let A be a decidable set and ϕ a decision procedure for A. In line with the proof of Theorem 3.1.3, we define, for an instance x, a parameterization in the style of Flum and Grohe as

$$\kappa(x) = \underbrace{\mathtt{111}\cdots\mathtt{1}}_{t\text{ times}},$$

where t is the number of steps ϕ uses in deciding membership of x in A. By construction, A is fixed-parameter tractable with respect to this parameterization in the framework of Flum and Grohe. Note that a computing $\kappa(x)$ is possible in roughly $\kappa(x)$ steps. Since A was an arbitrary decidable set, it follows that every decidable set is fixed-parameter tractable with respect to some parameterization.

This example reflects the idea behind Corollary 3.1.4, and can be summarized as follows.

3.1.6. Slogan. *When parameter values must be computable, a set can be fixed-parameter tractable precisely if it is decidable.*

More broadly, Corollary 3.1.4 tells us that parameterized analysis of computational complexity is at least as much about parameterizations as it is about sets.

3.1.2 Bounded Undecidability

With Corollary 3.1.4, we have identified the sets that can be analyzed in a parameterized framework where parameterizations are decidable or semidecidable. We seek to extend this result and classify the sets that can be analyzed in a parameterized framework with parameterizations of varying degrees of undecidability. Before an extended classification can be obtained in Section 3.1.3, we must gain control over the undecidability of parameterizations. Therefore, we take a closer look at the nature of convergence in convergent parameterized procedures. Recall from Definition 2.2.8 that a parameterized procedure is convergent if it converges to some set on some point-cofinite parameterization.

3.1.7. Lemma. *A parameterized procedure ϕ is convergent if and only if, for all x, the sets*

$$\{i \mid \phi(x, \text{asStr}(i)) \neq \phi(x, \text{asStr}(i+1))\} \text{ and}$$
$$\{k \mid \phi(x,k) = \texttt{?}\}$$

are finite.

Proof:
$\Longrightarrow$. Suppose ϕ converges to a set A on a point-cofinite parameterization η. For every (x, k), the output of $\phi(x,k)$ can only differ from $A(x)$ when x is not included in η_k. Because x is included in all but finitely many slices of η, the set of parameter values where ϕ changes its decision must hence be finite. In particular, ϕ cannot output `?` infinitely often for any fixed x and varying k.

$\Longleftarrow$. Let A consist of the instances x for which there are infinitely many values of k such that $\phi(x,k)$ outputs `1`. Consider the family of sets

$$(\{x \mid \phi(x,k) = A(x) \text{ and } |x| \leq \text{asInt}(k)\})_{k \in 2^+}.$$

Each element of this family is a finite set. Because ϕ cannot output `?` infinitely often, every instance x is included in all but finitely many elements of this family. Hence, this family is a point-cofinite parameterization and ϕ converges to A on it. □

Observe that in the right-to-left direction of the above proof, the finiteness of the elements of the family of sets matters. While a family like

$$(\{x \mid x \neq k\})_{k \in 2^+}$$

is a point-cofinite cover of 2^+, it is not directed and therefore not a point-cofinite parameterization. Making the sets in the family finite the way we did, is a fittingly technical solution to this troublesome technicality.

A direct parameterized procedure is one where the only changes on some instance x are from `?` to a ‘correct’ decision or vice versa. As we have seen, the limit sets of direct parameterized procedures were decidable. A parameterized investigation of sets that are not necessarily decidable starts with parameterized procedures that are convergent, but not necessarily direct. We shall look at parameterized procedures that are allowed to “change their mind” multiple times, but not indefinitely.

3.1.8. Definition. Let f be a function from 2^+ to $\mathbb{N}$. A parameterized procedure ϕ is *f-alternating* if for every instance x there are at most $f(x)$ values of i satisfying

$$\phi(x, \text{asStr}(i)) \neq \phi(x, \text{asStr}(i+1)).$$

Note that in this definition, we have imposed a linear order on the parameter values. In the presence of a parameterization, this imposed order need not be related to the inclusion order on the slices of the parameterization.

Extending Lemma 3.1.7, the convergent parameterized procedures, as described on page 58, can be characterized in yet another way. A parameterized procedure is convergent if, for some f, it is f-alternating and for every instance x it outputs `?` for only finitely many parameter values. The limit set of a convergent parameterized procedure is said to be *limit computable.* When the function f maps its inputs to a constant n, the limit set of an f-alternating parameterized procedure is said to be *weakly n-computably enumerable* [115, 50]. Such sets are encountered in the study of the difference hierarchy [46], which we discussed briefly in the introductory Section 1.2.1. In fact, the weakly n-computably enumerable sets constitute the Δ_n^{-1} level of that hierarchy. Recall that the class Δ_n^{-1} is defined as the intersection of Σ_n^{-1} and Π_n^{-1}. As observed before, classes of limits of parameterized procedures are closed under taking complements. Therefore, it is no surprise that our parameterized approach favors the Δ levels of the difference hierarchy. A complement of a set in a Σ level is in the corresponding Π level, and likewise the other way around.

Indeed, the difference hierarchy is strict. Already in the standard proofs thereof [14, 52, 124], we can see a parameterized line of reasoning at work. We shall give a different and more general proof, exposing a fine-grained structure among the limits of f-alternating sets. Central to this proof is the ability to bound the running time of an f-alternating parameterized procedure as a function of the parameter value exclusively. This bounding of the number of steps taken comes at the cost of an increase in the number of alternations of at most one.

3.1.9. Theorem. *Let ϕ be an f-alternating parameterized procedure that converges to a set A. There exists an $(1 + f)$-alternating parameterized procedure that converges to A and, on every input (x, k), terminates within $\mathcal{O}(|k|)$ steps. Here, the hidden constant does not depend on ϕ.*

Proof:
We define a parameterized procedure ϕ' with the desired properties as follows. On input (x, k), spend $|k|$ steps in total on simulating ϕ with varying parameter values: More specifically, ϕ' computes an initial segment of the sequence

$$(\phi(x, \text{asStr}(1)), \quad \phi(x, \text{asStr}(2)), \quad \phi(x, \text{asStr}(3)), \quad \ldots). \tag{3.1}$$

After spending $|k|$ steps doing so, ϕ' yields the output of the last completed simulation, or `?` if no simulation finished.

Observe that arbitrarily long initial segments of the sequence (3.1) are computed by ϕ'. The length of the initial segment that can be computed in $|k|$ steps is unbounded as a function of the length of k. By Lemma 3.1.7, this means that ϕ' converges to the same set as ϕ.

The number of changes that occur for a given x in the sequence (3.1) is related to f. After $\phi(x, \mathrm{asStr}(1))$ is included in the sequence, the output of ϕ' can change at most $f(x)$ times. The first output reproduced from a completed simulation of ϕ need not be `?`. Therefore, it is possible that the number of changes in the output of ϕ' is one higher than $f(x)$. □

Naturally, Theorem 3.1.9 can be adapted for resources other than time. Most importantly, the theorem implies that, when f is computable, it is possible to diagonalize against the f-alternating parameterized procedures.

3.1.10. LEMMA. *Let f and g be computable functions such that for all x we have $f(x) < g(x)$. There exists a set that is the limit of a g-alternating parameterized procedure, but not of any f-alternating parameterized procedure.*

Proof:
We shall diagonalize against the f-alternating parameterized procedures. In simulating a parameterized procedure, we can keep track of the number of steps taken. Therefore, we can enumerate the parameterized procedures with a running time bounded by the square of the length of the parameter value. Moreover, the number of alternations can be counted. Thus, we can enumerate the $(1+f)$-alternating parameterized procedures with a running time that is at most the square of the length of the parameter value. Let $\psi_1, \psi_2, \psi_3, \ldots$ be such an enumeration and consider the parameterized procedure

$$\psi(x, k) = \begin{cases} \mathtt{1} & \text{if } \psi_{\mathrm{asInt}(x)}(x, k) = \mathtt{0}, \\ \mathtt{0} & \text{otherwise.} \end{cases}$$

This procedure never produces `?` and is indifferent to whether $\psi_{\mathrm{asInt}(x)}(x, k)$ outputs `1` or `?`. Not keeping track of `?` is permitted in this setting since there is no need to code for unknown values. The procedure ψ thus defined is $(1+f)$-alternating and hence it is g-alternating. By Theorem 3.1.9, this construction diagonalizes against all f-alternating parameterized procedures. Accordingly, there is no f-alternating parameterized procedure that converges to the same set as ψ. □

The requirement on the relationship between the functions f and g can be weakened.

3.1.11. THEOREM. *Let f and g be computable functions such that for infinitely many x we have $f(x) < g(x)$. There exists a set that is the limit of a g-alternating parameterized procedure, but not of any f-alternating parameterized procedure.*

Proof:
Because f and g were both assumed to be computable, so is the infinite set $S = \{x \mid f(x) < g(x)\}$. Let $x_1, x_2, x_3, \ldots$ be an effective enumeration of the

members of S. We can prove the theorem by applying the previous lemma to these elements.

Consider the functions f' and g' obtained by restricting f and g to S as

$$f'(\text{asStr}(i)) = f(x_i) \qquad \text{and} \qquad g'(\text{asStr}(i)) = g(x_i).$$

These functions are so that for all x we have $f'(x) < g'(x)$. By Lemma 3.1.10, there is hence a set A that is the limit of a g'-alternating parameterized procedure, but not of any f'-alternating parameterized procedure. Using such a set A, a set that is the limit of a g-alternating parameterized procedure, but not of any f-alternating procedure is defined by

$$\{x \mid \exists i \colon x = x_i \ \text{ and } \ \text{asStr}(i) \in A\}.$$

In effect, this set is the result of using only the members of S for the diagonalization in the proof of Lemma 3.1.10. □

Similar theorems have been published [50, 14], but our parameterized framework inspired a more elegant proof.

3.1.3 Reducibility to the Halting Set

A classification of which sets are open to a parameterized analysis with parameterizations of varying undecidability is now within reach. To begin with, we observe that the undecidable halting set,

$$\text{Halt} = \{x \in 2^+ \mid x \text{ encodes a procedure that takes no input and terminates}\},$$

appears as early as is possible in the hierarchy of limits of f-alternating parameterized procedures.

3.1.12. Lemma. *There is a* 1*-alternating parameterized procedure that converges to* Halt.

Proof:
Let us define such a parameterized procedure ϕ. On input (x, k) it simulates the procedure encoded by x for up to asInt(k) steps. If the simulation terminates, then ϕ outputs `1`, else it outputs `0`. Given any instance x, for large enough values of asInt(k) the output of ϕ will correspond to membership of x in Halt. For members of Halt, ϕ may change its decision once, as asInt(k) becomes large enough. Otherwise, ϕ outputs `0`. From this, it follows that ϕ is 1-alternating. □

As it turns out, the class of limits of f-alternating parameterized procedures where f is computable is closed under truth-table reducibility as defined in Definition 2.1.7. The halting set is complete for this class [50, 14, 46].

3.1.13. Theorem. *There is a computable function f and an f-alternating parameterized procedure converging to a set A if and only if A is truth-table reducible to* Halt.

Proof:
$\Longrightarrow$. Let f be a computable function and ϕ an f-alternating parameterized procedure converging to a set A. The idea is to locate, given an instance x, the last change in the output of ϕ for varying parameter values, using the halting set. To this end, let ψ_x be a procedure that takes a number m as input and searches for the parameter value at which the output of ϕ on input x changes for the mth time. If ϕ changes fewer than m times, then ψ_x does not halt on input m. Otherwise, the output of ψ_x is the mth value of i satisfying $\phi(x, \text{asStr}(i)) \neq \phi(x, \text{asStr}(i+1))$.

Using at most $f(x)$ queries to Halt, we can find the greatest value of m such that $\psi_x(m)$ terminates. A truth-table reduction from A to Halt may compute this m and output $\phi(x, \text{asStr}(\psi_x(m)+1))$. Thus, the reduction finds the final decision of ϕ regarding membership of x in A.

$\Longleftarrow$. Let ψ be the 1-alternating parameterized procedure converging to Halt that we constructed in the proof of Lemma 3.1.12. We shall use ψ to define a computable function f and an f-alternating parameterized procedure ϕ converging to A. On input (x, k), our ϕ simulates the reduction from A to Halt to obtain a truth-table for x. Let $q_1, q_2, q_3, \ldots, q_m$ be the queries prescribed by this truth-table. Next, ϕ evaluates the truth-table using $\psi(q_1, k), \psi(q_2, k), \psi(q_3, k), \ldots, \psi(q_m, k)$ as decisions for the queries. For sufficiently large values of $\text{asInt}(k)$, the queries will all be answered correctly. Before that, the decisions for the queries as provided by ψ can each only change from `0` to `1`, but never back. Therefore, ϕ gets to evaluate the truth-table with at most $m+1$ different sets of answers to the queries. As a result, ϕ changes its decision for a given x at most m times. Because m can be computed from x, the function $f(x) = m$ is computable. $\square$

With regard to the halting set, truth-table reducibility coincides with a weaker variant, aptly named *weak truth-table reducibility* [115, 46]. However, it is not as general as Turing reducibility, which, as seen in Definition 2.1.6, may be adaptive. The reason for this is that the proof of Theorem 3.1.11 can be changed so that it also diagonalizes against all computable functions. As we shall soon see, the resulting set is Turing reducible to the halting set. By Theorem 3.1.11 and Theorem 3.1.13, it cannot also be truth-table reducible to the halting set. This tells us something about the sets that can be analyzed in a parameterized framework with undecidable parameterizations. Looking at f-alternating parameterized procedures, fewer sets can be obtained as limits if we require the function f to be computable.

We want to adapt the proof of Theorem 3.1.11 so that we also diagonalize against all computable functions bounding the number of alternations. Therefore, we interpret the instance part, x, of the input (x, k) of our parameterized procedure as a pair $x = \langle x_1, x_2 \rangle$. We then use $\text{asInt}(x_1)$ as the index of a parameterized procedure to diagonalize against, and asInt(x_2) as the index of a computable bounding function. Crucially, we do not need to know the time required for computing the bounding function, or even whether the computation will terminate at all. We shall increasing the time spent computing the bounding function with the length of the parameter value, $|k|$. By doing so, we eventually compute each terminating function to completion. To be precise, the second procedure, indexed by asInt(x_2), is simulated on x for $|k|$ steps. In case the simulation does not finish, we proceed as if it had produced 0 as output. Otherwise, we use the output of the simulation as the maximum number of alternations allowed to be made by the parameterized procedure indexed by asInt(x_1). After thus computing the bound, we turn to simulating the parameterized procedure indexed by asInt(x_1). This is done like in the proof of Theorem 3.1.11, this time using the computed bound on the number of alternations, instead of a fixed one. Thus, we can diagonalize against all parameterized procedures that, for some computable function f, are f-alternating.

This observation shows that there are sets that are Turing reducible to the halting set, yet not truth-table reducible to the halting set [50, 14, 46]. Recall that truth-table reducibility was a more demanding form of Turing reducibility. Therefore, all sets that are truth-table reducible to the halting set are also Turing reducible to the halting set. The difference hierarchy was extended by Ershov [53] to include the class of sets that are Turing reducible to the halting set. In order to do so, levels of the hierarchy were introduced beyond those with finite indices, starting at ω, the first infinite ordinal. The sets that are truth-table reducible to the halting set form the Δ^{-1}_{ω} level of the transfinite difference hierarchy. For larger, yet constructible [131], ordinals α, the Δ^{-1}_{α} level matches the class of sets that are Turing reducible to the halting set [53, 50]. By Post's theorem [122, 131], we can fit this class of sets in the arithmetical hierarchy. A set is Turing reducible to the halting set precisely when it sits at the Δ^0_2 level of the arithmetical hierarchy.

In essence, it was recognized by Shoenfield [138] that a set is Turing reducible to Halt precisely when a convergent parameterized procedure converges to it. This characterization is known as the *limit lemma* [115, 46]. The original framing of the lemma was around approximations that could change only finitely often for any instance. In terms of f-alternating parameterized procedures, the characterization forgoes any computability requirements on f and augments Theorem 3.1.13.

3.1.14. Theorem. *There is a parameterized procedure converging to a set A if and only if A is Turing reducible to* Halt.

Proof:
$\Longrightarrow$. The same approach as in the proof of Theorem 3.1.13 is applicable. However, while the required number of queries to Halt is still finite, we no longer have a computable upper bound for it. Thus, the reduction we obtain need not be a truth-table reduction. Instead, we obtain a Turing reduction.

Note that we may use the adaptive nature of Turing reductions and employ a binary search strategy to locate the final change of the parameterized procedure. When doing so, the number of queries to Halt scales only as the logarithm of the number of alternations. In the end, this does not help us, since there need not be a computable upper bound on this number of alternations.

$\Longleftarrow$. The proof of Theorem 3.1.13 does not carry over to the current theorem immediately. The reduction from A to Halt may be adaptive, so we cannot gather all queries at once. In particular, for certain incorrect answers to the queries, the reduction may never stop making new queries. Even when the reduction only makes a finite number of queries, its computation after it has gotten the answers to the queries need not terminate. While we have no way of answering all queries correctly, a solution to this is available along the lines of Theorem 3.1.9.

Let ψ be the parameterized procedure that converges to Halt as constructed in the proof of Lemma 3.1.12. We shall use ψ to define a parameterized procedure ϕ converging to A. On input (x, k), our ϕ simulates the reduction from A to Halt for up to $|k|$ steps. When the reduction queries membership of some q in Halt, the query is answered according to $\psi(q, k)$. The steps needed to compute $\psi(q, k)$ are not counted toward the number of steps used in simulating the reduction. If the reduction arrives at a decision within the allotted number of steps, the decision is propagated by ϕ. Otherwise, ϕ yields ?. For sufficiently long parameter values, this procedure can simulate the reduction to completion and answers all queries in agreement with Halt. □

Earlier, we recovered the class of decidable sets in the parameterized context via decidable parameterizations. Theorem 3.1.13 and Theorem 3.1.14 can be used in a similar fashion. By the first, we could call a set a *bounded parameterized limit* precisely when it sits at the Δ^{-1}_{ω} level of the difference hierarchy. The second suggests to call the sets at the Δ^0_2 level of the arithmetical hierarchy the *parameterized limits*. Moreover, by Theorem 3.1.9, these two classes are insensitive to resource bounds that are independent of the parameter value. For example, suppose the running time of some parameterized procedure ϕ is, say, exponential in the length of the instance part of its input. Note that this resource bound does not specify its dependency on the parameter value and we assume this dependence can be anything. Furthermore, assume that ϕ is convergent and it converges to a set A. From Theorem 3.1.9, we know that there is another parameterized procedure converging to A of which the running time does not depend on the instance at all. Hence, the fact that ϕ has a running time that is exponential in the length of the instance is irrelevant to the classification of A. That is, this running time says nothing about the membership of A in either Δ^{-1}_{ω} or Δ^0_2. By contrast, the specifics of a convergent or direct parameterized procedure do tell us something about its limit set. Our results regarding the classification of sets on the basis of the specifics of parameterized procedures is summarized in Table 3.2.

Like what we found in Corollary 3.1.4, our results connect to a school of pa-

parameterized procedures	*sets*
direct	Δ^0_1, decidable
f-alternating	Δ^{-1}_ω, truth-table reducible to HALT
convergent	Δ^0_2, Turing reducible to HALT

Table 3.2: The choice of which parameterized procedures to work with determines which sets occur as limits of our parameterized procedures. We have identified three classes of sets corresponding to three classes of parameterized procedures. Note that, in this table, the function f is assumed to be computable.

rameterized complexity theory. In the style pioneered by Downey and Fellows [44], parameterized complexity theory is more permissive than it is with Flum and Grohe [57]. With Downey and Fellows, a parameter value is no longer required to be computable as a function of an instance. Owing to this lack of a computability requirement, we could say that in their framework, a parameter value may be *promised.* By Theorem 3.1.14, the existence of a parameterization with which a set A is in a given parameterized complexity class tells us very little about A.

3.1.15. COROLLARY. *Suppose we are working in a parameterized framework where parameter values may be promised. Let* $\boldsymbol{C}$ *be a class of sets that are decidable in some given parameterized time bound. A set* A *is in* $\boldsymbol{C}$ *with respect to some parameterization if and only if* A *is in the* Δ^0_2 *level of the arithmetical hierarchy.*

To illustrate this corollary, we turn to fixed-parameter tractability in the Downey and Fellows framework.

3.1.16. EXAMPLE. The Downey and Fellows framework deals with parameterized sets, so we first need to settle on a way to deal with classical sets in this framework. First, observe that a decision procedure for a parameterized set in the sense of Downey and Fellows can be thought of as a parameterized procedure. We shall only consider parameterized sets that can be decided by convergent parameterized procedures, as defined in Definition 2.2.8. The motivation for this is that, conceptually, we model resource bounds as a function of the parameter value. For sufficiently large parameter values, we ought to have enough of a computational resource at our disposal to be able to decide membership. Increasing the resource bound beyond that point should not change our decision about membership. Let us say that a parameterized version of a set A is any parameterized set B that satisfies

$$A = \{x \mid \exists k\colon \langle x, k\rangle \in B\}.$$

It follows from Theorem 3.1.14 that the sets in Δ^0_2 are precisely those to which a parameterized procedure converges. In other words, there is a parameterized version of a set A if and only if A is in Δ^0_2. From Theorem 3.1.9 it follows that this remains true if we confine our attention to parameterized procedures with a running time as required by fixed-parameter tractability.

Comparing Corollary 3.1.15 to Corollary 3.1.4, we see that we have gone one step up in the Δ levels of the arithmetical hierarchy. Because Δ^0_1 is the class of decidable sets, computable parameter values are related to Δ^0_1 in the same way that promised parameter values are related to Δ^0_2. These classifications correspond to the first and last row in Table 3.2. Completing our treatment of these two extremes, we can contrast Slogan 3.1.6 with the following summary of Corollary 3.1.15.

3.1.17. SLOGAN. *When parameter values may be promised, a set can be fixed-parameter tractable precisely if it is in* Δ^0_2.

3.1.4 Subparameterizations

So far, we have focused mainly on sets that appear as limits of parameterized procedures. In particular, our focus has been on convergent parameterized procedures. There are parameterized procedures that converge on some point-cofinite parameterization. We shall now examine in a little more detail the structure of point-cofinite parameterizations. Recall that in a point-cofinite parameterization, each instance occurs in all but finitely many slices. More formally, a parameterization η is point-cofinite if it satisfies

$$\forall x\colon \forall^\infty k\colon x \in \eta_k.$$

We could say that this property is a local property, as it describes the behavior of the parameterization at individual instances. Parameterizations most commonly encountered in practice [113] often enjoy a global variant, namely they satisfy

$$\forall k\colon \forall^\infty k'\colon \eta_k \subseteq \eta_{k'}. \tag{3.2}$$

Indeed, this property is sufficient for a parameterization to be a point-cofinite parameterization, yet it is not necessary. A point-cofinite parameterization η may contain a slice that is incomparable to infinitely many others, which would prevent η from adhering to (3.2).

3.1.18. EXAMPLE. An example of a point-cofinite parameterization that does not adhere to (3.2) is the parameterization given by

$$\eta = (\{x \mid k \text{ is not a substring of } x\})_{k \in 2^+}.$$

Since no string x has a substring that is longer than $|x|$, every string x is in every slice η_k for which we have $|x| < |k|$. Therefore, η is point-cofinite.

Observe that for any two numbers i and j, if we have $i \neq j$, then the slice with parameter value $\texttt{1}\texttt{0}^i\texttt{1}$ is incomparable to that with parameter value $\texttt{1}\texttt{0}^j\texttt{1}$. From this, it follows that η does not satisfy (3.2).

To ascertain that η is indeed a parameterization, we also need to verify that it is directed. For this, it suffices to note than two slices η_{k_1} and η_{k_2} are both included in the slice $\eta_{k_1 k_2}$. Indeed, if a string x would contain the concatenation $k_1 k_2$ as a substring, then it would also contain both k_1 and k_2 as substrings.

At the same time, a selection of slices from a parameterization such that the selection does obey (3.2) can always be made. In fact, this *subset* of the parameterization, a *subparameterization*, may be chosen so that it is ordered linearly by inclusion.

3.1.19. Lemma. *Every parameterization has a subset of its slices that forms a linearly ordered point-cofinite parameterization. Specifically, for every parameterization η, there is a set $I \subseteq 2^+$ that satisfies*

- $(\eta_k)_{k \in I}$ *is a point-cofinite parameterization, and*
- *for all $k \in I$ and $k' \in I$, we have $\eta_k \subseteq \eta_{k'}$ or $\eta_{k'} \subseteq \eta_k$.*

If, for all k and k' in 2^+, we can effectively decide whether $\eta_k \subseteq \eta_{k'}$ holds, then a set I as described above exists that is decidable.

Proof:
Associated with a parameterization η is a reflexive and transitive order on the set of all parameter values, 2^+. This order stems from the inclusion order on the slices of η. A parameter value k is ordered before or at a parameter value k' if we have $\eta_k \subseteq \eta_{k'}$. In the presence of such orders, the linearly ordered subsets are called *chains*. Accordingly, a chain C of parameter values is a subset of 2^+ such that for all $k \in C$ and $k' \in C$ we have $\eta_k \subseteq \eta_{k'}$ or $\eta_{k'} \subseteq \eta_k$. Certain chains of parameter values are of particular interest in the context of the current lemma. These are the chains that dominate all parameter values. A chain C is a *cofinal chain* [1] if for every parameter value $k \in 2^+$ there is a parameter value $k' \in C$ such that we have $\eta_k \subseteq \eta_{k'}$. Setting aside the decidability constraint for a moment, we find that any cofinal chain of parameter values can serve as a witness set I in the current lemma. Because a parameterization is directed, we can be sure that it contains cofinal chains.

When the order on parameter values is decidable, a decidable cofinal chain can be constructed. We do so by going through all parameter values and selectively including some of them. Whenever a parameter value is included, we implicitly add the corresponding slice of the parameterization to our subparameterization. Our criterion for including a parameter value in our aspiring cofinal chain is a greedy one. Let $k_1, k_2, k_3, \ldots$ be an enumeration of all parameter values. We start our subparameterization by including k_1. After that, we build our subparameterization by repeating the following steps.

1: Let i be the last number such that k_i was included in our subparameterization.

2: Find the first number j such that each of the slices $\eta_{k_1}, \eta_{k_2}, \eta_{k_3}, \dots, \eta_{k_i}$ is included in η_{k_j}. Such a number j exists because η is directed.

3: Include k_j in our subparameterization.

This procedure is effective and by construction the resulting subset is ordered linearly and cofinal. In each round through these steps, we may resume the search for the desired number j where we finished the previous round. Thus, we may assume that j only increases, which ensures that the subparameterization is decidable. To check whether a parameter value k is included in the subparameterization, we run the construction until we have $k = k_j$. If k is not included at the corresponding round of our construction, it will never be. ☐

If the parameterization we start with includes 2^+ as a slice, it has a subparameterization consisting only of that slice. This is a rather uninteresting scenario, so we shall focus on parameterizations that do not include 2^+. Of such parameterizations, all cofinal subsets contain infinitely many different slices. Any cofinal subset of a such a parameterization is itself a parameterization after suitably indexing its elements by binary strings. However, the corresponding subset of the original index set need not be decidable. If it were, then each convergent parameterized procedure would converge on some decidable point-cofinite parameterization. This would mean that we might as well assume that our convergent parameterized procedures are direct parameterized procedures. In turn, this would violate the results summarized in Table 3.2.

We can even force the subset of the index set corresponding to a subparameterization to be particularly undecidable: The subset can be made so that it nor its complement contains an infinite semidecidable set. In other words, there is a parameterization such that the set of parameter values corresponding to a particular subparameterization is bi-immune in accordance with Definition 2.1.5. This claim will be proven as Lemma 3.1.21 shortly.

In preparation for this, it is helpful to have a linear order on the subsets of 2^+. The set inclusion order, $\subseteq$, is not a linear order. A convenient linear order on the subsets of 2^+ comes about when we map a set $A \subseteq 2^+$ to the real number

$$\text{asReal}(A) = \sum_{x \in A} 2^{-\text{asInt}(x)}.$$

We order one subset of 2^+ before another if its corresponding real number is smaller than that of the other. That the mapping to real numbers is not injective is a technical detail that is of no consequence to our analysis. To turn our order into a true linear order, we need to decide what to do in case the subsets are different but the corresponding real numbers are the same. In that case, one of the subsets is infinite and the other is not, and we may simply order the infinite subset before the finite one.

When a set A is the limit of a parameterized procedure, the real number asReal(A) is said to be *computably approximable* [9]. Historically, computably approximable real numbers have also been called *limit computable* real numbers [66]. Justifying this terminology, a parameterized procedure ϕ that converges to a set A can be thought of as approximating asReal(A). Considering, for a parameter value k, the set

$$A_{\phi,k} = \{x \mid \phi(x,k) = \mathtt{1}\},$$

we find that the set of real numbers $\{\text{asReal}(A_{\phi,k}) \mid k \in \mathtt{2}^+\}$ has a single limit point, namely asReal(A). Moreover, if the parameterized procedure is f-alternating for some computable function f, the real number is said to be *ω-computably enumerable* [9]. Thus, the class of computably approximable real numbers is Δ^0_2, while Δ^{-1}_ω is the class of ω-computably enumerable real numbers.

A parameterized procedure ϕ is called *normed* [9] if it satisfies, for all x and k,

$$\phi(x,k) = \mathtt{1} \implies \text{asInt}(x) \leq |k|.$$

This definition is motivated purely by practical considerations and it holds no information about the limit of the parameterized procedure. Nevertheless, normed procedures will be of use to us in the next two lemmas.

3.1.20. Lemma. *Let ϕ be an f-alternating parameterized procedure that converges to a set A. There exists a normed $(1+f)$-alternating parameterized procedure that converges to A and, on every input (x,k), terminates within $\mathcal{O}(|k|)$ steps. Here, the hidden constant does not depend on ϕ.*

Proof:
This lemma is a minor modification of Theorem 3.1.9. The parameterized procedure constructed in the proof of that theorem can be made into a normed parameterized procedure. To that end, on input (x,k), the procedure first checks whether asInt(x) is at most $|k|$. If not, it outputs `?` and does not proceed any further. The extra computation can be performed in a number of steps bounded linearly in $|k|$. $\square$

Given a normed parameterized procedure ϕ and two parameter values k, k', it is decidable whether asReal$(A_{\phi,k}) \leq$ asReal$(A_{\phi,k'})$ holds. For this, we run ϕ on the first $\max\{\text{asInt}(k), \text{asInt}(k')\}$ instances x, once with the parameter value k and once with k'. The first instance on which ϕ outputs `1` with only one of the two parameter values determines which of asReal$(A_{\phi,k})$ and asReal$(A_{\phi,k'})$ is bigger. If no such instance is found, then the two real numbers are equal. In the case where the running time of ϕ is bounded linearly in the length of the parameter value, such a decision can be made within a stringent time bound. Specifically, a decision can be made in a number of steps that is quadratic in $|k| + |k'|$.

We can now show that there are parameterizations with subparameterizations of which the corresponding set of parameter values is highly undecidable.

3.1.21. Lemma. *There is a parameterization with a subparameterization of which the corresponding subset of parameter values is bi-immune.*

Proof:
Let A be a set such that asReal(A) is computably approximable, but not ω-computably enumerable. Because the inclusion of Δ^{-1}_{ω} in Δ^0_2 is strict, such a set A exists. Furthermore, let ϕ be a normed parameterized procedure converging to A with a running time bounded linearly in the length of the parameter value. Such a procedure exists by Lemma 3.1.20. Lastly, define a set of parameter values

$$C = \{k \mid \text{asReal}(A_{\phi,k}) < \text{asReal}(A)\},$$

reminiscent of a Dedekind cut. Note that the set of real numbers

$$\{\text{asReal}(A_{\phi,k}) \mid k \in 2^+\}$$

has a single limit point, asReal(A). Therefore, the set C defines a subparameterization of the point-cofinite parameterization

$$(\{x \mid \phi(x,k) = A(x) \;\text{ and }\; \text{asInt}(x) \le |k|\})_{k \in 2^+}$$

on which ϕ converges to A. We claim that both C and its complement do not contain any infinite semidecidable subset, and thus that C is bi-immune.

Because ϕ is normed, the order on the real numbers associated with the parameter values is decidable. Moreover, the set of real numbers associated with C contains no greatest element. This allows an argument similar to the one used for Lemma 3.1.19. If C contains an infinite semidecidable subset, there is an infinite decidable set $\{k_1, k_2, k_3, ...\} \subseteq C$ satisfying

$$\forall i < j\colon \; \text{asReal}(A_{\phi,k_i}) < \text{asReal}(A_{\phi,k_j}).$$

Now, consider a parameterized procedure that maps $(x, \text{asStr}(i))$ to $\phi(x, k_i)$. The approximation of asReal(A) corresponding to this parameterized procedure is a monotonically increasing one. Accordingly, for the computable function f that maps x to $2^{\text{asInt}(x)}$, this parameterized procedure is f-alternating. However, A was defined so that it was not the limit of an f-alternating parameterized procedure for any computable function f. Hence C does not contain an infinite semidecidable subset. A similar argument, inverting the order, holds for the complement of C, completing the proof that C is bi-immune. □

The details of the above proof are akin to those used by Jockusch [88] in his study of sets with “selector” functions. Such functions show that a set has some structure, especially if we impose a polynomial bound on the running time of the functions [136].

3.1.22. Definition. A set A is *p-selective* if there is a function $f\colon 2^+ \times 2^+ \to 2^+$ that is computable in polynomial time and satisfies, for all strings x, y,

- $f(x, y) \in \{x, y\}$, and
- $x \in A \vee y \in A \implies f(x, y) \in A$.

Exploiting the decidability of the order on the real numbers associated with parameter values further, we find a strengthening of Lemma 3.1.21. This shows that there are sets that are at the same time both highly undecidable and rich in structure.

3.1.23. Theorem. *There is a parameterization with a subparameterization of which the corresponding subset of parameter values is bi-immune, yet p-selective.*

Proof:
The bi-immune set C constructed in the proof of Lemma 3.1.21 is also p-selective. Given parameter values k_1, k_2, we can find which is associated with the smallest real number in a number of steps that is bounded polynomially in $|k_1| + |k_2|$. If any of the parameter values is in C, the one associated with the smallest real number is. $\square$

The existence of a p-selective bi-immune set was claimed before by Goldsmith, Joseph, and Young [70]. However, the proof that was provided was a convoluted finite injury priority argument [see also 46, Section 2.11]. In our parameterized framework, such arguments can be made more transparent, as demonstrated by the proofs of Lemma 3.1.21 and Theorem 3.1.23. The number of alternations of a parameterized procedure is related to the number of injuries in a finite injury construction. In that sense, our proof does not differ from the original proof. However, we feel that by their framing, our proofs contribute more to an intuitive understanding of why the statements are true.

3.2 as Computational Tractability

The computational complexity of a set is focused predominantly on the worst-case or average-case complexity of its instances. This has been so since the identification of efficient computability by Cobham and Edmonds around 1965 [for some background, see 68]. Intractability results [34, 63] generally consider the hardest instances, while average-case complexity looks at complexity of the bulk. Yet, even very hard sets can have simple instances and often lots of them.

The indiscriminate judgment of the computational complexity of sets was addressed by Lynch [102] in 1975. She dealt with the distribution of complexity inside sets by examining intrinsically hard subsets. Although fixed-parameter tractability would only be established some two decades later [41], a parameterized computational complexity theory was thus started. In this section, we shall build a parameterized theory of computational complexity on these early foundations. While the traditional parameterized classes are easily recovered, our analysis will be concerned with parameterizations as independent measures of complexity. Much of our theory revolves around collections of parameterizations that put a given set in a parameterized complexity class. Such collections function as an interface to the complexity of the sets that gave rise to them. Moreover, they reveal that no practically fixed-parameter tractable sets have optimal parameterizations.

Synopsis

Our investigation of parameterized computational complexity theory starts off in Section 3.2.1 with rediscovering standard parameterized complexity classes. We shall arrive at these classes in a way that showcases their relation to classical computational complexity theory. The guiding insight, here, is that intractable decision problems may still have sets of instances on which membership can be decided easily. We have seen examples of this in Section 1.2.2. Compared to the more empirical approach to parameterized complexity classes that is commonplace, our approach is theoretical and more systematic.

In Section 3.1, we found that parameterizations deserve to be studied independently of decision problems. Our framework for parameterized complexity theory readily isolates parameterizations as distinct entities and makes such a study possible. This is helped greatly by the theoretical quality of our approach to parameterized complexity classes. A structural analysis of parameterizations is carried out in Section 3.2.2. The way parameterizations relate to each other makes that parameterizations form a rich algebraic structure. We show that this structure doubles as a ranking of parameterizations.

Having established a way to rank parameterizations, we ask under what circumstances optimal parameterizations exist. This question is addressed for a nonuniform parameterized complexity theory in Section 3.2.3. The uniform version is the subject of Section 3.2.4. As it turns out, for most problems that arise naturally, there are no optimal parameterizations among those that make the problem fixed-parameter tractable.

Previously, for a given problem we looked at whether parameterizations exist that make the problem fixed-parameter tractable. We found that this was not sensitive to variations in parameterized computational tractability. Instead, we should look at the collective of parameterizations that make the problem fixed-parameter tractable. The observation that most natural problems do not have optimal parameterizations is a result of this new perspective.

3.2.1 Stratified Computational Complexity

The intrinsically hard parts of a set examined by Lynch are those on which no decision procedure is efficient infinitely often. This idea can be made precise using polytime-approximations. A hard part of a set A, then, is a selection of instances of which no polytime-approximation for A is able to decide infinitely many. Recall from Definition 2.2.11 that the elements of which an approximation is able to decide membership make up the domain of the approximation.

3.2.1. Definition. A set C is a *polytime-core* of a set A if for every polytime-approximation ϕ for A the intersection $C \cap \mathrm{dom}(\phi)$ is finite.

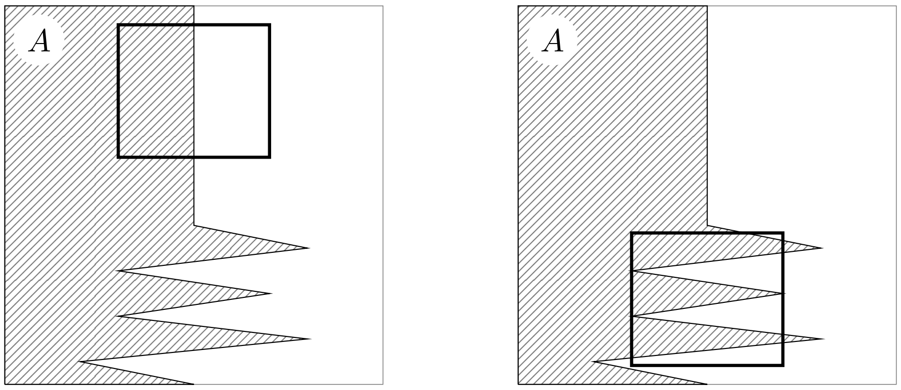
A set that is decidable in polynomial time does not have an infinite polytime-core. Lynch [102] observed that the converse is true as well: Every set outside **P** has an infinite polytime-core. Note that a polytime-core of a set need not be a subset of that set. If it is, it is known as a *proper* core. We shall give a name to a kind of dual to a core as well.

3.2.2. Definition. A set S is a *polytime-segment* of a set A if there is a polytime-approximation ϕ for A such that we have $S = \mathrm{dom}(\phi)$.

Replacing the polytime designator by a function t we obtain the definition of a *t-segment*. When we do not care about constant multiplicative factors, we resort to the class of functions $\mathcal{O}(f)$ and obtain the definition of an *$\mathcal{O}(f)$-segment*. For these last two cases, the fact that the resource being bounded is time is left implicit.

Thus, a set is a polytime-core if it has a finite intersection with every polytime-segment. Note that these definitions are free from any considerations regarding density. The only distinction made is between finite and infinite subsets in the definition of a polytime-core.

3.2.3. Example. We can illustrate the concept underlying polytime-cores and polytime-segments in order to get a more intuitive understanding. The region highlighted in Figure 3.1a represents a polytime-segment and the region highlighted in Figure 3.1b represents a polytime-core. Note that membership in a polytime-core itself need not be hard to decide, as witnessed by the simple boundary of the polytime-core in Figure 3.1b.



(a) A polytime-segment. Within the polytime-segment, there is a simple boundary between the members of A and the nonmembers of A.

(b) A polytime-core. Within the polytime-core, the members of A and the nonmembers of A cannot be separated easily.

Figure 3.1: We visualize the set of all strings as a bounded surface and depict a set A as a shaded region. Where the line between the inside of A and the outside of A is jagged, A is hard to distinguish from its complement.

Subsets of polytime-cores are polytime-cores for the same set. Perhaps surprisingly, this means that, for any set, the empty set is a polytime-core. While this may be counterintuitive, it is of no concern technically. Likewise, if we add a finite number of elements to a core or remove a finite number of elements from a core, we still have a core. This complicates thinking of the members of a core as inherently hard instances of a set. Any specific instance can arbitrarily be made part of, or excluded from a polytime-core. However, adding or removing an infinite number of elements to a core is not always possible. We can order a collection of cores by inclusion up to *finite variations*, meaning that a core D is greater than a core C if

- $D \setminus C$ has infinitely many elements, and
- $C \setminus D$ has finitely many elements.

This order is a partial order on any collection of sets. For some sets the collection of polytime-cores contains a *maximal* element, as discussed on page 49, with respect to inclusion up to finite variations. If this is the case, the set is split into an easy part and a hard part.

3.2.4. Theorem. *A set has a maximal (up to finite variations) polytime-core if and only if it has a maximal polytime-segment.*

Proof:
$\Longleftarrow$. The complement of a maximal polytime-segment of a set is a polytime-core of that set: Otherwise, some polytime-approximation of the set would decide

infinitely many instances outside the polytime-segment. By combining polytime-approximations, these infinitely many instances could be added to our polytime-segment, in violation of its maximality.

Any polytime-core that is the complement of a polytime-segment cannot be extended by infinitely many instances, hence such a core is a maximal polytime-core.

$\Longrightarrow$. We claim that the complement of every maximal polytime-core of a set A is a polytime-segment of A. Suppose, toward a contradiction, that C is a maximal polytime-core of A and that the complement of A is not a polytime-segment of A. Let $S_1, S_2, S_3, \ldots$ be an enumeration of the polytime-segments of A. Note that since each segment corresponds to an approximation for A, of which there are at most countably many, there are at most countably many segments. Furthermore, although it is not relevant for this proof, we note that this enumeration will be nonuniform. Now, for all j, there are infinitely many elements in the complement of C outside the polytime-segment $\bigcup_{i \leq j} S_i$. Consequently, we would be able to extend C with infinitely many elements, one for each j, contradicting the maximality of C. Hence, the complement of a maximal polytime-core of A must be a polytime-segment of A. Being the complement of a polytime-core, this polytime-segment is maximal. $\Box$

It follows that the complement of a maximal polytime-segment is a maximal polytime-core. Because polytime-segments are necessarily in **P**, we get the following.

3.2.5. Corollary. *For any given set, a maximal polytime-core, if it exists, is in* **P**.

Of course, within a maximal polytime-core C of a set A, membership of an instance in A is hard to decide. Even though C is in **P**, no polytime-approximation for A decides membership in A for infinitely many elements of C.

Recall that a set is outside **P** precisely when it has an infinite polytime-core [102]. Some sets are so far removed from **P** that they have no infinite polytime-segments. This is a form of immunity against **P**. In line with Theorem 3.2.4, we have an elegant characterization of the sets that are bi-immune for **P** in terms of polytime-cores. This alternative characterization was already observed by Balcázar and Schöning [18] [see also 27].

3.2.6. Theorem. *A set A is* **P**-bi-immune *if and only if* 2^+ *is a polytime-core of* A.

Proof:
$\Longleftarrow$. We shall prove the contrapositive formulation of the claim, namely that if A is not **P**-bi-immune, then 2^+ is not a polytime-core of A. In case A is not **P**-bi-immune, there is an infinite set $S \in$ **P** that is either entirely inside of A, or

entirely outside of A. Either way, we can define a polytime-approximation for A of which S is the domain. As S is an infinite subset of 2^+, it cannot be the case that 2^+ is a polytime-core of A.

$\Longrightarrow$. In a similar vein, if 2^+ is not a polytime-core of A, then there is a polytime-approximation for A with an infinite domain S. Because S is the domain of a polytime-approximation for A, both $S \cap A$ and $S \cap A^{\mathrm{C}}$ are in **P**. Since S is infinite, at least one of these sets is infinite, from which it follows that A cannot be **P**-bi-immune. $\square$

Thus a set is **P**-bi-immune if it has the largest polytime-core possible. In general, we do not care for any easily recognizable redundancy in a set. Therefore, it is useful to give a name to sets that fall apart into an easy and a hard part. In other words, we are interested in sets with a maximal polytime-core.

3.2.7. Definition. A set is *almost* **P**-*bi-immune* if it has a maximal polytime-core.

This definition builds on the notion of “almost **P**-immune” sets, which are defined as the disjoint union of a **P**-immune set and a set in **P**. The equivalence of this last definition to one involving maximal proper polytime-cores was observed by Orponen [116].

Note that sets in **P**, and finite sets in particular, have maximal polytime-cores and are therefore almost **P**-bi-immune. This may seem somewhat peculiar, but is in agreement with the definitions used by Orponen [116] and of no objection in our theory.

With polytime-segments and polytime-cores we make no distinction between the members and the nonmembers of a set. Segments as well as cores of a set are maximal if they cannot be extended by infinitely many members or nonmembers of the set. For segments, this extension must be made in uniform fashion, while for cores it need not. In [118, 117], sets of which every polytime-segment can be extended by infinitely many *members* into a larger polytime-segment are called **P**-*levelable*. Thus, conceptually, a **P**-levelable set can be approximated from within by a sequence of polytime-segments, each segment infinitely larger than the one before. For our analysis, we need a more general definition that covers sets of which every polytime-segment can be extended by infinitely many arbitrary instances. That is, we do not want to confine our attention to just the members of a set. While tempting, such sets should not be called **P**-bi-levelable: That name, which does not occur in the literature, would more naturally describe sets that are **P**-levelable *and* have a **P**-levelable complement. For us, either being **P**-levelable *or* having a **P**-levelable complement suffices.

3.2.8. Definition. A set is **P**-*semi-levelable* if it has no maximal polytime-segment.

By Theorem 3.2.4 and in line with the work of Orponen, Russo, and Schöning [118], a set is **P**-semi-levelable precisely when it is not almost **P**-bi-immune. Furthermore, every **P**-levelable set is **P**-semi-levelable. It was observed by Orponen, Russo, and Schöning [117] that very many (natural) intractable sets are **P**-levelable. As a consequence, we find that there are many **P**-semi-levelable sets.

Having two complementary definitions, almost **P**-bi-immune and **P**-semi-levelable, may seem superfluous. On page 107, however, we shall take these notions into a parameterized setting, where they will no longer be complementary.

The structure of the collection of polytime-segments of a set tells us something about how computational complexity is distributed over its instances. This is especially true when the set at hand is **P**-semi-levelable. At a conceptual level, the collection of polytime-segments of a set holds information about how the set relates to **P**. Every polytime-segment of a set represents a part of the set that is decidable in polynomial time and that is in that sense "in **P**". The collection of polytime-segment thus pulls the computational complexity of a set apart into layers: It provides a *stratified* view of computational complexity. This insight inspires the definition of a complexity class that lifts **P** into a setting of parameterized analysis. One way to formalize this lifting is by means of the nonuniform operator $\boldsymbol{X}_{\mathbf{nu}}$.

3.2.9. Definition. A set A is in $\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}$ with parameterization η if for every parameter value k, the set η_k is a polytime-segment of A.

Lifting **P** to different fields of analysis is not new and several ways of doing so have been studied previously. Using $\exists$ as an operator, the nondeterministic counterpart, **NP**, of **P** has been obtained as $\exists\mathbf{P}$. In probabilistic complexity theory, the $\boldsymbol{BP}$ operator was derived from the complexity class $\boldsymbol{BP}\mathbf{P}$ by Schöning [134]. This operator was then used to define various probabilistic complexity classes based on their deterministic counterparts.

The class $\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}$ is nonuniform in two ways. Firstly, a set may be in $\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}$ without there being a procedure to instantiate polytime-approximations from their corresponding parameter values. Secondly, even if there was, the parameter dependence of the polynomial time bounds of the approximations may not be of a computational nature. These considerations spur a fully uniform alternative to the nonuniform $\boldsymbol{X}_{\mathbf{nu}}$ operator. Corresponding parameterized complexity classes are called *strongly uniform* by Downey and Fellows [44].

3.2.10. Definition. A set A is in $\boldsymbol{X}\mathbf{P}$ with parameterization η if there is a direct parameterized procedure ϕ and a computable function f satisfying

- η is the parameterization corresponding to ϕ in accordance with Definition 2.2.9, and
- for every parameter value k, with t_k mapping n to $f(k) \cdot n^{f(k)}$, the partial application of ϕ to k yields a t_k-approximation for A.

Here, the partial application of ϕ to k is the function that maps an instance x to $\phi(x,k)$.

Note that we indicate nonuniformity with the $_{\mathbf{nu}}$-subscript, but use no modifier for the fully uniform case. In other places, e.g. `https://complexityzoo.uwaterloo.ca/Complexity_Zoo:X`, special notation is used instead to denote uniform classes.

In the current century, the class $\boldsymbol{X}\mathbf{P}$ has been called *slicewise* **P**, giving a name to the $\boldsymbol{X}$ operator [56]. However, in earlier work, Downey and Fellows [44] used "slicewise **P**" to denote a different complexity class, namely **FPT**. The definition of $\boldsymbol{X}\mathbf{P}$ is very permissive in that the exponent in the running time of the polytime-approximations may be unbounded. In **FPT**, the class of *fixed-parameter tractable* sets, this freedom is restricted.

3.2.11. Definition. A set A is in $\mathbf{FPT_{nu}}$ with parameterization η if there is a polynomial p such that for every parameter value k the set η_k is an $\mathcal{O}(p)$-segment of A. Here, the hidden constant may depend on k.

The slices of a parameterization with which a set is in $\mathbf{FPT_{nu}}$ are polytime-segments where the degree of the polynomials is bounded by a constant. Observe that this amounts to a reversal of quantifiers when compared to $\mathbf{XP_{nu}}$. With $\mathbf{FPT_{nu}}$ there is a single polynomial that is used for all parameter values. On the other hand, with $\boldsymbol{X}\mathbf{P_{nu}}$ every parameter value may have a different polynomial associated to it. Since we have $\mathbf{P} = \bigcup_c \mathbf{TIME}(n^c)$, we can express this reversal of quantifiers symbolically as

$$\boldsymbol{X}\mathbf{P_{nu}} = \boldsymbol{X}\Big(\bigcup_c \mathbf{TIME}(n^c)\Big)_{\mathbf{nu}}$$

and

$$\mathbf{FPT_{nu}} = \bigcup_c \Big(\boldsymbol{X}\mathbf{TIME}(n^c)_{\mathbf{nu}}\Big).$$

The fully uniform version of these equalities directs us to the definition of fixed-parameter tractability.

3.2.12. Definition. A set A is in **FPT** with parameterization η if there is a direct parameterized procedure ϕ, a computable function f, and a polynomial p satisfying

- η is the parameterization corresponding to ϕ, and
- for every parameter value k, the partial application of ϕ to k yields an $(f(k)\cdot p)$-approximation for A.

This definition is not too different from either of the traditional definitions that we have seen in Section 1.3. We observed in Example 3.1.16 that decision procedures in the framework of Downey and Fellows roughly correspond to parameterized procedures in our framework. Accordingly, their notion of fixed-parameter tractability matches Definition 3.2.12 rather nicely. A comparison with fixed-parameter tractability in the framework of Flum and Grohe can also be made. Suppose we have a set A that is fixed-parameter tractable in that framework. This means that to each string x a parameter value $\kappa(x)$ is associated with which we can bound the running time of a decision procedure for A appropriately. The mapping of strings to parameter values can be turned into a point-cofinite parameterization in our framework as

$$\eta = (\{x \mid \operatorname{asInt}(\kappa(x)) \leq \operatorname{asInt}(k)\})_{k \in 2^+}.$$

From the fact that A is fixed-parameter tractable in the Flum and Grohe framework, it follows that it is in **FPT** with parameterization η. In Section 3.4.4, the function κ will reappear as a polytime-computable parameter estimator for the parameterization η defined above.

In our definitions of ***XP*** and **FPT**, Definition 3.2.10 and Definition 3.2.12, we have required the parameterized procedures to be direct. As a consequence of this, a parameterization η with which a set is in ***XP*** or **FPT** must be decidable. Moreover, the computation time required for deciding whether an instance x is in slice k of η is in line with the time bound for the respective class. For example, for **FPT**, there is a computable function f and polynomial p such that deciding whether x is in η_k is possible in a time bounded by $f(k) \cdot p(|x|)$. On page 28 in Section 1.3, we argued that it is reasonable to require decidability of parameterizations within such time bounds.

As alluded to in Section 1.3, many sets have been found to be fixed-parameter tractable with more or less natural parameterizations [42, 113, 37]. Moreover, parameter values associated with typical instances are small in practice [45, 44]. In that way, fixed-parameter tractability is a successful notion of efficient computability.

3.2.13. Example. A less practically motivated way to obtain sets and parameterizations that are in **FPT** is available for p-cylinders as defined in Definition 2.1.13. Given a decidable p-cylinder A and a corresponding isomorphism $g\colon 2^+ \to 2^+ \times 2^+$, denote by g_1 the first component of the image of g. Thus, if g maps x to (y, z), then g_1 maps x to y. Now, consider the point-cofinite parameterization based on g_1 that is given by

$$\eta = (\{x \mid \operatorname{asInt}(g_1(x)) \leq \operatorname{asInt}(k)\})_{k \in 2^+}.$$

To see that A is in **FPT** with η, fix a decision procedure ϕ for A and consider the direct parameterized procedure that, on input (x, k), proceeds as follows.

```
1: If $\operatorname{asInt}(g_1(x))$ is greater than $\operatorname{asInt}(k)$, return ?.
```

2: Else, since we have $x \in \eta_k$ and must decide on membership of x in A,
  return the output of ϕ on input $g_1(x)$.

The running time of the first step of this procedure can be bounded polynomially in $|x|$, while that of the second step can be bounded purely as a function of k. Thus, this parameterized procedure witnesses that A is in **FPT** with η.

When a set A is in a parameterized complexity class, for example **FPT**, with a parameterization η, we write $(A, \eta) \in \mathbf{FPT}$. We remark that members of (nonuniform) $\boldsymbol{X}\mathbf{P}$ need not be in (nonuniform) **FPT** [44, 57]. Likewise, the classes $\boldsymbol{X}\mathbf{P}$ and **FPT** are strictly smaller than their nonuniform counterparts, $\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}$ and $\mathbf{FPT}_{\mathbf{nu}}$ [43]. These relations are visualized in Figure 3.2.

$$\begin{tikzcd}
& \boldsymbol{X}\mathbf{P}_{\mathbf{nu}} & \\
\boldsymbol{X}\mathbf{P} \arrow[ur, phantom, "\subsetneq" sloped] & & \mathbf{FPT}_{\mathbf{nu}} \arrow[ul, phantom, "\subsetneq" sloped] \\
& \mathbf{FPT} \arrow[ul, phantom, "\subsetneq" sloped] \arrow[ur, phantom, "\subsetneq" sloped] &
\end{tikzcd}$$

Figure 3.2: The relations between several uniform and nonuniform parameterized complexity classes.

3.2.2 Order Theory for Parameterizations

We are now in a position to study the distribution of complexity inside a set in terms of the parameterizations that put the set in one of our classes. To this end, two collections of parameterizations that put a given set in some parameterized complexity class are of central importance.

3.2.14. Definition. Given a parameterized complexity class $\boldsymbol{C}$ and a set A, we denote the collection of parameterizations with which A is in $\boldsymbol{C}$ by

$$\mathcal{F}_{\boldsymbol{C}}(A) = \{\eta \mid (A, \eta) \in \boldsymbol{C}\}.$$

More generally, we consider the parameterizations in light of a parameterized complexity class irrespective of a particular set,

$$\mathcal{L}_{\boldsymbol{C}} = \{\eta \mid \exists A\colon (A, \eta) \in \boldsymbol{C}\}.$$

As with polytime-cores, we are not so much interested in finite variations on the slices of a parameterization. On the basis of Theorem 3.2.4, we do not expect any meaningful theory to be possible that is sensitive to finite variations. Instead, we are mostly interested in parameterizations of which the slices grow in infinitely large steps.

3.2.15. Definition. A parameterization η has *imix* (infinitely many infinite extensions) if for every parameter value k there is a parameter value k' such that the set $\eta_{k'} \setminus \eta_k$ is infinite.

Conceptually, a parameterization that has imix embodies a form of levelability. Instead of referring to polytime-segments, this form of levelability pertains to the slices of the parameterization. Of course, for some set of interest, these slices may be polytime-segments.

Recall that a parameterization is a directed cover. Because of that, if $\eta_{k'} \setminus \eta_k$ is infinite, there exists a slice that is an infinite superset of η_k.

3.2.16. Example. All slices of the length parameterization of Example 2.2.3 are finite. Therefore, the length parameterization does not have imix. Conversely, each slice of the p-cylinder parameterizations of Example 3.2.13 introduces infinitely many elements over the preceding slices. Hence, these parameterizations do have imix.

The length parameterization is not very interesting because any decidable set A is fixed-parameter tractable with it. Indeed, the running time of a decision procedure for A on any finite set of instances can be bounded by a constant. More broadly, there is no real benefit in using a direct parameterized procedure of which the associated parameterization does not have imix. Such a parameterization η contains a slice, S, that has no infinite extensions and if A is in **FPT** with η, then S is a polytime-segment of A. Deciding membership in A for elements of S requires no parameterizations. Additionally, outside S, the parameterization η is no more useful than the length parameterization.

There may be more than one direct parameterized procedure converging to the same set. As the corresponding parameterizations, say η and ζ, could be different, we would like to be able to compare parameterizations. In order to do so, we need an order on the parameterizations. This order should be meaningful to our parameterized theory of computational complexity. In particular, we desire membership in a given set A in one of our parameterized complexity classes, say **FPT**, to be somehow preserved by this order. If A is in **FPT** with η and the order tells us to prefer η over ζ, than A should also be in **FPT** with ζ.

Intuitively, comparing our parameterizations η and ζ is most relevant when the parameterized procedures have similar parameterized running times. In that case, it may be possible to argue in favor of one procedure over the other based on the parameterizations alone. If the parameterized running times are similar, the preferred procedure, for an instance x, is the one corresponding to the lowest value of $\mu_\eta(x)$ and $\mu_\zeta(x)$. We can take this idea further and compare the parameterizations on all instances at once. Given two parameterizations, we look at bounds on the minimization function of the one in terms of the minimization function of the other. Thus, consider the required minimum length of a parameter value in one parameterization for instances of a bounded parameter length in another parameterization.

3.2.17. Definition. Given a parameterization η and a parameterization ζ, the *gap* function, $\mathrm{gap}_{\eta,\zeta}\colon \mathbb{N} \to \mathbb{N} \cup \{\infty\}$, is defined as

$$\mathrm{gap}_{\eta,\zeta}(m) = \max\{\mu_\eta(x) \mid x \in 2^+ \text{ and } \mu_\zeta(x) \leq m\},$$

where we take the maximum of the empty set to be 0.

The name 'gap' suggests that we should take the maximum not of $\mu_\eta(x)$, but of $\mu_\eta(x) - m$. This more semantically correct version can be obtained from the version as defined above and vice versa. As a result, the choice of a definition is theoretically inconsequential. Our stripped-down definition is, however, slightly more convenient to work with.

Comparing parameterizations using the gap function enables us to define a nonuniform and a uniform order on parameterizations. We would like to formalize when a parameterization η is to be preferred over, or *below*, another. This is the case, when a bound on minimum length of a parameter value for the other parameterization can be turned into such a bound for η. Similar orders have been considered by Komusiewicz and Niedermeier [95] and Fellows, Jansen, and Rosamond [54].

3.2.18. Definition. A parameterization η is below a parameterization ζ in the *nonuniform* order $\preccurlyeq_{\mathrm{nu}}$ if we have, for all m,

$$\mathrm{gap}_{\eta,\zeta}(m) < \infty.$$

Alternatively, this nonuniform order can be related to the inclusion of the slices of one parameterization in the slices of another. If a parameterization η is below a parameterization ζ, then all slices of ζ are included in slices of η, but not the other way around. Informally, η takes bigger steps in covering 2^+ than ζ.

3.2.19. Lemma. *The following statements about parameterizations η and ζ are equivalent.*

1. $\eta \preccurlyeq_{\mathrm{nu}} \zeta$.
2. $\forall k\colon \exists k'\colon \zeta_k \subseteq \eta_{k'}$.

Proof:
1 $\implies$ 2. Let ζ_k be any slice of ζ and observe that for all $x \in \zeta_k$, we have $\mu_\zeta(x) \leq |k|$. Hence, by definition of the gap function, for all $x \in \zeta_k$, we also have $\mu_\eta(x) \leq \mathrm{gap}_{\eta,\zeta}(|k|)$. Assuming we have $\eta \preccurlyeq_{\mathrm{nu}} \zeta$, this latter bound is finite and the elements of ζ_k can thus be found spread over finitely many slices of η. Specifically,

the elements of ζ_k are spread over the slices of η corresponding to parameter values with a length of at most $\mathrm{gap}_{\eta,\zeta}(|k|)$. Moreover, because parameterizations are directed, there is a parameter value k' such that we have

$$\bigcup_{\substack{j \text{ with} \\ |j| \leq \mathrm{gap}_{\eta,\zeta}(|k|)}} \eta_j \subseteq \eta_{k'}.$$

With this k', we also have $\zeta_k \subseteq \eta_{k'}$.

$2 \implies 1$. Let m be any constant and consider the set of strings

$$S_m = \{x \mid \mu_\zeta(x) \leq m\} = \bigcup_{\substack{j \text{ with} \\ |j| \leq m}} \zeta_j.$$

We need to show that $\max\{\mu_\eta(x) \mid x \in S_m\}$ is finite. Observe that S_m is built from finitely many slices of ζ. Because parameterizations are directed, there is therefore a parameter value k such that we have $S_m \subseteq \zeta_k$. Hence, by assumption, there is a parameter value k' such that we also have $S_m \subseteq \eta_{k'}$. From this, it follows that the value of μ_η cannot be greater than $|k'|$ for elements of S_m. As m was arbitrary, this proves that we have $\eta \preccurlyeq_{\mathrm{nu}} \zeta$. □

We note that two parameterizations η and ζ may be equivalent according to $\preccurlyeq_{\mathrm{nu}}$, while the growth rate of μ_η is wildly different from that of μ_ζ. For instance, for all x we may have $\mu_\eta(x) = 2^{2^{\mu_\zeta(x)}}$. However, such a relation between parameterizations is in itself no reason to prefer one over the other. To see why, consider the running time of a direct parameterized procedure ϕ that witnesses that some set is in **FPT**. This running time can be bounded from above by the product of a function of the parameter value and a polynomial of the length of the instance. That is, there is a computable function f and a polynomial p such that the running time of ϕ on any two input strings x and k is at most $f(k) \cdot p(|x|)$. The function of the parameter value, f, provides a specific relationship between parameter values and running times. Even if we have $\mu_\eta(x) = 2^{2^{\mu_\zeta(x)}}$, we can only compare running times if we take into account the different functions of the parameter value. The parameterized procedure corresponding to ζ may in the end have a milder dependence on the parameter values than that corresponding to η. Thus, the growth rate of the minimization function is not a useful indicator of computational complexity. At best, the minimization function can be used to compare the computational complexity of individual instances qualitatively. If the minimization function assigns a higher value to some x than to some y, then the computational complexity of x can be said to be higher than that of y. Still, we cannot use the growth rate for a comparison of parameterizations. We shall see in Example 3.2.28 that our order based on the gap function nevertheless gives some information on the comparison of running times. Before that, however, we shall show how it is that our order provides a compromise between mathematical elegance and practical relevance. The elegance lies in the fact that we obtain a rich mathematical structure, the relevance lies in the fact that the order preserves fixed-parameter tractability.

From a computational standpoint, a bound on the gap between two parameterizations is only useful if it is computable. By Theorem 3.1.1, the uniform variant of the order on parameterizations is of interest for parameterizations arising from direct parameterized procedures.

3.2.20. Definition. A parameterization η is below a parameterization ζ in the *uniform* order $\preccurlyeq$ if there is a computable function f such that we have, for all m,

$$\text{gap}_{\eta,\zeta}(m) \leq f(m).$$

3.2.21. Example. We shall compare two parameters of graphs, the number of vertices and the number of edges. Consider the parameterizations corresponding to the number of vertices and the number of edges in a graph given by

$$\eta = (\{G \mid G \text{ has at most asInt}(k) \text{ vertices}\})_{k \in 2^+},$$
$$\zeta = (\{G \mid G \text{ has at most asInt}(k) \text{ edges}\})_{k \in 2^+}.$$

Because any graph that has v vertices has at most $\binom{v}{2}$ edges, we have $\zeta \preccurlyeq \eta$. The two parameterizations are, however, not equivalent, because a graph with e edges can have an arbitrarily large number of vertices. Hence, we also have $\eta \not\preccurlyeq \zeta$.

We observe the relationship $\preccurlyeq \subset \preccurlyeq_{\text{nu}}$ between our uniform order and our nonuniform order. The orders provide structure to the class of parameterizations.

3.2.22. Lemma. *Both $\preccurlyeq_{\text{nu}}$ and $\preccurlyeq$ are reflexive and transitive orders on the class of parameterizations.*

Proof:
Reflexivity follows from the observation that for every parameterization η the gap function $\text{gap}_{\eta,\eta}$ is bounded by the identity function. For transitivity, two remarks suffice. Firstly, the composition of finite bounding functions is again a finite bounding function. Secondly, the composition of computable functions is computable. □

Neither order is antisymmetric. Therefore, it is convenient to work with the associated partially ordered set of *equivalence classes* instead of with parameterizations directly. If we use minimization functions to compare the computational complexity of individual instances, then equivalent parameterizations provide similar results. Suppose we have equivalent parameterizations η and ζ, and two instances, x and y that are assigned the same value by μ_η. In this case, the difference between $\mu_\zeta(x)$ and $\mu_\zeta(y)$ can be bounded by a constant that depends on

$\mu_\eta(x) = \mu_\eta(y)$, but not on x or y. Likewise, if we have $\mu_\zeta(x) = \mu_\zeta(y)$, then the difference between $\mu_\eta(x)$ and $\mu_\eta(y)$ can be bounded.

Note that every parameterization with imix is unequal to any parameterization without imix, both using the nonuniform as well as using the uniform order.

3.2.23. Lemma. *Let η and ζ be parameterizations, where η has imix but ζ does not. One of $\eta \preccurlyeq_{\mathrm{nu}} \zeta$ and $\zeta \preccurlyeq_{\mathrm{nu}} \eta$ fails to hold (and similarly for $\preccurlyeq$).*

Proof:
The statement for the uniform order follows from that for the nonuniform order by the inclusion $\preccurlyeq \subset \preccurlyeq_{\mathrm{nu}}$.

Let m be so that no parameter value of length at least m is associated with a slice of ζ that has an infinite extension. Such an m exists for every parameterization that does not have imix, because parameterizations are directed.

Suppose we have $\eta \preccurlyeq_{\mathrm{nu}} \zeta$. There is then a parameter value k such that η_k is a superset of all slices ζ_j for which we have $|j| \leq m$. Since η has imix, there exists a parameter value k' such that $\eta_{k'} \setminus \eta_k$ is infinite. We find that $\zeta \preccurlyeq_{\mathrm{nu}} \eta$ fails because $\mathrm{gap}_{\zeta,\eta}(|k'|)$ must be infinite.

Suppose we have $\zeta \preccurlyeq_{\mathrm{nu}} \eta$. If $\eta \preccurlyeq_{\mathrm{nu}} \zeta$ were to hold as well, then $\mathrm{gap}_{\eta,\zeta}(m)$ would be finite. However, this would mean that there is a parameter value k' such that $\eta_{k'}$ is infinitely larger than any ζ_j, whenever we have $|j| \leq m$. As ζ does not have imix, this would violate the assumed relationship $\zeta \preccurlyeq_{\mathrm{nu}} \eta$. □

In the context of orders on parameterizations, we refer to the equivalence classes of parameterizations when speaking simply of parameterizations. As we have seen in Lemma 3.2.23, parameterizations with imix will remain distinct from those without when employing this convention.

We claim that in relation to $\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}$ and $\mathbf{FPT}_{\mathbf{nu}}$, the nonuniform order $\preccurlyeq_{\mathrm{nu}}$ is a natural order on parameterizations to look at. Likewise, in relation to $\boldsymbol{X}\mathbf{P}$ and $\mathbf{FPT}$, the uniform order $\preccurlyeq$ is a natural order to consider. As we shall soon see, there are two main reasons for this. On the one hand, these orders open up parameterized analysis of complexity to algebraic methods. On the other, these orders represent a way to compare the speed of convergence represented by different parameterizations. This, in turn, provides an abstract way to compare the computational power required by parameterized decision procedures for different sets.

Nonuniform Complexity Classes

When we look at $\mathcal{L}_{\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}}$ or $\mathcal{L}_{\mathbf{FPT}_{\mathbf{nu}}}$ in terms of equivalence classes ordered by $\preccurlyeq_{\mathrm{nu}}$, we recognize a familiar [38] structure.

3.2.24. Theorem. *Ordered by $\preccurlyeq_{\mathrm{nu}}$, the equivalence classes in $\mathcal{L}_{\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}}$ and $\mathcal{L}_{\mathbf{FPT}_{\mathbf{nu}}}$ form bounded distributive lattices.*

Proof:
Observe that the slices of any parameterization in $\mathscr{L}_{\boldsymbol{XP}_{\mathbf{nu}}}$ are all in $\mathbf{P}$. Likewise, for each parameterization η in $\mathscr{L}_{\mathbf{FPT}_{\mathbf{nu}}}$ there is a constant c such that all slices of η are in $\mathbf{TIME}(n^c)$. The converses of both statements are witnessed by the empty set: The empty set is put in $\boldsymbol{XP}_{\mathbf{nu}}$ or $\mathbf{FPT}_{\mathbf{nu}}$ by parameterizations of which all slices are in $\mathbf{P}$ or, for some c, in $\mathbf{TIME}(n^c)$, respectively.

For the current proof we shall focus on $\mathscr{L}_{\mathbf{FPT}_{\mathbf{nu}}}$. The proof for $\mathscr{L}_{\boldsymbol{XP}_{\mathbf{nu}}}$ is slightly less involved.

$\mathscr{L}_{\mathbf{FPT}_{\mathbf{nu}}}$ is bounded. A least element of $\mathscr{L}_{\mathbf{FPT}_{\mathbf{nu}}}$ is present in the form of the parameterization consisting of 'full' slices only, $(2^+)_{k\in 2^+}$. Indeed, any constant bounds the gap from this parameterization to another and thus this parameterization is below any other with respect to $\preccurlyeq_{\mathrm{nu}}$.

A greatest element can be found in any parameterization of which the slices are all finite. Take, for instance, the length parameterization,

$$(\{x \mid |x| \leq \mathrm{asInt}(k)\})_{k\in 2^+}.$$

Regardless of the bound on the length of parameter values, the gap from any parameterization to this parameterization is the maximum of a finite set. The gap is therefore always finite. Hence, this parameterization is above any other with respect to $\preccurlyeq_{\mathrm{nu}}$.

$\mathscr{L}_{\mathbf{FPT}_{\mathbf{nu}}}$ contains greatest lower bounds. Given two parameterizations, $\eta \in \mathscr{L}_{\mathbf{FPT}_{\mathbf{nu}}}$ and $\eta' \in \mathscr{L}_{\mathbf{FPT}_{\mathbf{nu}}}$, we can construct a new parameterization that acts as a greatest lower bound for the two. Define this new parameterization as

$$\zeta = (\eta_k \cup \eta'_{k'})_{\langle k,k'\rangle\in 2^+} = (\{x \mid x \in \eta_k \vee x \in \eta'_{k'}\})_{\langle k,k'\rangle\in 2^+}.$$

Because both η and η' are directed covers, ζ is as well. Hence ζ is indeed a parameterization. Moreover, if, for some c, all slices of η and all slices of η' are in $\mathbf{TIME}(n^c)$, then all slices of ζ are in $\mathbf{TIME}(n^c)$. As η and η' were picked from $\mathscr{L}_{\mathbf{FPT}_{\mathbf{nu}}}$, we find that ζ must also be a member of $\mathscr{L}_{\mathbf{FPT}_{\mathbf{nu}}}$.

By construction, every slice of either η or η' is included in a slice of ζ. It follows that the gap from ζ to both η and η' is finite and thus that ζ is below each of them. It remains to show that ζ is the greatest among such lower bounds. For this, suppose that some ζ' too is below both η and η' and consider $\mathrm{gap}_{\zeta',\zeta}$. Because our pairing function is length-increasing in both arguments, we have, for every m,

$$\mathrm{gap}_{\zeta',\zeta}(m) \leq \max\{\mathrm{gap}_{\zeta',\eta}(m),\ \mathrm{gap}_{\zeta',\eta'}(m)\}.$$

As both elements of the set on the right-hand side are finite by assumption, we find that the gap from ζ' to ζ is finite for all values of m. Thus, ζ' is below ζ, and ζ is a greatest lower bound for η and η' with respect to $\preccurlyeq_{\mathrm{nu}}$, as desired.

$\mathscr{L}_{\mathbf{FPT_{nu}}}$ contains least upper bounds. In a similar vein, the existence of least upper bounds can be shown. Taking the slicewise intersection of two given parameterizations, η and η', we obtain

$$\zeta = (\eta_k \cap \eta'_{k'})_{\langle k,k'\rangle \in 2^+} = (\{x \mid x \in \eta_k \wedge x \in \eta'_{k'}\})_{\langle k,k'\rangle \in 2^+}.$$

Like the parameterization constructed previously, this defines a directed cover, thus, indeed, a parameterization. When η and η' are members of $\mathscr{L}_{\mathbf{FPT_{nu}}}$, this parameterization is a member of $\mathscr{L}_{\mathbf{FPT_{nu}}}$ too.

Using that our pairing function is length-increasing, we find, for all m, that $\mathrm{gap}_{\eta,\zeta}(m)$ and $\mathrm{gap}_{\eta',\zeta}(m)$ are both at most m. Thus, ζ is an upper bound on η and η'. To see that it is a least upper bound, suppose that ζ' is another upper bound on η and η'. Our pairing function is so that, for all k and k', we have $|\langle k, k'\rangle| \leq |k| + 2 \cdot |k'|$. Because of this, we also have, for all m,

$$\mathrm{gap}_{\zeta,\zeta'}(m) \leq \mathrm{gap}_{\eta,\zeta'}(m) + 2 \cdot \mathrm{gap}_{\eta',\zeta'}(m).$$

Since the right-hand side of this equation is finite, we can conclude that we have $\zeta \preccurlyeq_{\mathrm{nu}} \zeta'$. Accordingly, ζ is a least upper bound for η and η' with respect to $\preccurlyeq_{\mathrm{nu}}$.

$\mathscr{L}_{\mathbf{FPT_{nu}}}$ is distributive. Let η and η' be two parameterizations in $\mathscr{L}_{\mathbf{FPT_{nu}}}$. As is customary, we denote their greatest lower bound by $\eta \wedge \eta'$ and their least upper bound by $\eta \vee \eta'$. Beware that in the above construction, $\eta \wedge \eta'$ was based on a disjunction $x \in \eta_k \vee x \in \eta'_{k'}$ and vice versa for $\eta \vee \eta'$. Also, recall that in this context, we use parameterizations in place of their equivalence classes according to $\preccurlyeq_{\mathrm{nu}}$. We shall show that, for any three parameterizations η, η', and η'' in $\mathscr{L}_{\mathbf{FPT_{nu}}}$, the equivalence class of

$$\zeta = \eta \vee (\eta' \wedge \eta'')$$

equals that of

$$\zeta' = (\eta \vee \eta') \wedge (\eta \vee \eta'').$$

The dual, where $\wedge$ and $\vee$ are interchanged, is implied [38].

By our constructions of greatest lower bounds and of least upper bounds, we may interpret a parameter value for ζ as a triplet $\langle i, \langle j, k\rangle\rangle$. If an instance x is in a slice $\langle i, \langle j, k\rangle\rangle$ of ζ, it is also in slice $\langle\langle i, j\rangle, \langle i, k\rangle\rangle$ of ζ'. Made explicit, this comes down to the logical implication

$$x \in \eta_i \wedge (x \in \eta'_j \vee x \in \eta''_k) \implies (x \in \eta_i \wedge x \in \eta'_j) \vee (x \in \eta_i \wedge x \in \eta''_k).$$

Our pairing function is so that $|\langle i, \langle j, k\rangle\rangle|$ is at least $|i|+|j|+|k|$ and $|\langle\langle i, j\rangle, \langle i, k\rangle\rangle|$ is at most $|i| + 2 \cdot |j| + 2(|i| + 2 \cdot |k|) \leq 4(|i| + |j| + |k|)$. It follows that, for all m, we have $\mathrm{gap}_{\zeta',\zeta}(m) \leq 4m$. Hence, it follows that we have $\zeta' \preccurlyeq_{\mathrm{nu}} \zeta$.

For the converse, observe that if an instance x is in a slice $\langle\langle i_1, j\rangle, \langle i_2, k\rangle\rangle$ of ζ', it must also be in slice $\langle i_1, \langle j, k\rangle\rangle$ or in slice $\langle i_2, \langle j, k\rangle\rangle$ of ζ. This follows from the logical implication

$$(x \in \eta_{i_1} \wedge x \in \eta'_j) \vee (x \in \eta_{i_2} \wedge x \in \eta''_k) \implies \begin{cases} x \in \eta_{i_1} \wedge (x \in \eta'_j \vee x \in \eta''_k) \text{ or} \\ x \in \eta_{i_2} \wedge (x \in \eta'_j \vee x \in \eta''_k). \end{cases}$$

An argument concerning length bounds of our pairing function similar to the one used before now gets us a bound on the gap from ζ to ζ'. For all m, we have $\text{gap}_{\zeta,\zeta'}(m) \leq 4m$. Thus we also have $\zeta \preccurlyeq_{\text{nu}} \zeta'$ and both parameterizations are in the same equivalence class. □

For this proof, it is important that the definition of a parameterization makes no strong demands on the inclusion order of slices. Most expositions of parameterized computational complexity theory appear to tacitly assume that the inclusion order is a linear order. In the most prevalent parameterizations, parameter values are interpreted as numbers. The inclusion order on the slices of these parameterizations follows the standard numerical order on the corresponding parameter values. However, in the previous proof, a linear inclusion order on slices is not preserved by the constructions of greatest lower bounds and of least upper bounds.

The previous theorem shows that $\preccurlyeq_{\text{nu}}$ is a mathematically sensible order, as it provides structure to $\mathcal{L}_{\boldsymbol{XP}_{\text{nu}}}$ and $\mathcal{L}_{\mathbf{FPT}_{\text{nu}}}$. The following companion theorem goes one step further and shows that, moreover, $\preccurlyeq_{\text{nu}}$ is a meaningful order. It is meaningful to the extent that, for every set A, it also provides structure to $\mathcal{F}_{\boldsymbol{XP}_{\text{nu}}}(A)$ and $\mathcal{F}_{\mathbf{FPT}_{\text{nu}}}(A)$. The structure it provides these collections with relates nicely to the overarching structure of $\mathcal{L}_{\boldsymbol{XP}_{\text{nu}}}$ and $\mathcal{L}_{\mathbf{FPT}_{\text{nu}}}$, respectively [see also 38, 1]. If we accept $\mathbf{FPT}_{\mathbf{nu}}$ as a meaningful complexity class, then we are forced to accept $\preccurlyeq_{\text{nu}}$ as a meaningful order on parameterizations. We have seen that the minimization functions of equivalent parameterizations may have wildly different growth rates. While this may be deemed undesirable, it corresponds to a feature of $\mathbf{FPT}_{\mathbf{nu}}$. In the definition of $\mathbf{FPT}_{\mathbf{nu}}$ there are no restrictions on the parameter-dependence of the running time of the polytime-approximations considered.

3.2.25. THEOREM. *Let A be a set. Ordered by $\preccurlyeq_{\text{nu}}$, the collections $\mathcal{F}_{\boldsymbol{XP}_{\text{nu}}}(A)$ and $\mathcal{F}_{\mathbf{FPT}_{\text{nu}}}(A)$ are filters in $\mathcal{L}_{\boldsymbol{XP}_{\text{nu}}}$ and $\mathcal{L}_{\mathbf{FPT}_{\text{nu}}}$, respectively.*

Proof:
Like with Theorem 3.2.24, we present a proof for the $\mathbf{FPT}_{\mathbf{nu}}$ case only. All ingredients of a proof for the $\boldsymbol{XP}_{\mathbf{nu}}$ case are included in this proof.

$\mathcal{F}_{\mathbf{FPT}_{\text{nu}}}(A)$ **is nonempty.** For every polynomial p, every finite set is an $\mathcal{O}(p)$-segment of A. Therefore, every parameterization of which all slices are finite is in $\mathcal{F}_{\mathbf{FPT}_{\text{nu}}}(A)$. An example of such a parameterization is the length parameterization.

$\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$ **is upward closed.** Let η be a member of $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$ and ζ be any parameterization in $\mathcal{L}_{\mathbf{FPT}_{\mathbf{nu}}}$ for which we have $\eta \preccurlyeq_{\mathrm{nu}} \zeta$. Furthermore, let c be so that the slices of ζ are all in $\mathbf{TIME}(n^c)$. It suffices to prove the existence of a polynomial p such that the slices of ζ are $\mathcal{O}(p)$-segments of A.

By Lemma 3.2.19, since we have $\eta \preccurlyeq_{\mathrm{nu}} \zeta$, given a parameter value k, there is a parameter value k' such that we have $\zeta_k \subseteq \eta_{k'}$. Also, as η is in $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$, there is a polynomial q such that any slice of η is the domain of an $\mathcal{O}(q)$-approximation for A. We can construct an approximation for A that runs the approximation with domain $\eta_{k'}$, but only for members of ζ_k. With p mapping n to $n^c + q(n)$, the running time of such an approximation can be kept in $\mathcal{O}(p)$. This p is a polynomial that is independent of the parameter value k, so it meets our requirements.

$\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$ **contains greatest lower bounds.** Given parameterizations η and η' taken from $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$, consider the greatest lower bound ζ as constructed in the proof of Theorem 3.2.24. There are polynomials q and q' such that every slice of η is an $\mathcal{O}(q)$-approximation for A and every slice of η' is an $\mathcal{O}(q')$-approximation for A. By definition, each slice of ζ is the union of a slice of η and a slice of η'. The corresponding approximations on the constituent slices can be combined into an approximation of which the slice of ζ at hand is the domain. Now, let p be the polynomial that maps n to $q(n) + q'(n)$. Regarding the running time of the composite approximation, we find that it can be kept in $\mathcal{O}(p)$. Because p is independent of the specific slice of ζ, this puts ζ in $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$. $\square$

The above theorem allows us to think of $\preccurlyeq_{\mathrm{nu}}$ as a nonuniform ranking of how powerful parameterizations are. Two parameterizations that are related by $\preccurlyeq_{\mathrm{nu}}$ can be compared with regard to the way in which their slices cover the set of all strings. Informally, as seen in Lemma 3.2.19, a parameterization that is below another according to $\preccurlyeq_{\mathrm{nu}}$ takes bigger steps in covering $\mathbb{2}^+$. In that sense, a parameterization that is below another according to $\preccurlyeq_{\mathrm{nu}}$ is representative of a faster convergence. Note that, here, “faster” is not measured in terms of time. Instead, it relates to how big the steps are with which the slices of a parameterization grow.

With this in mind, the properties outlined in the previous theorem have an interpretation in terms of how powerful parameterizations are. Suppose that a set is in $\boldsymbol{XP}_{\mathbf{nu}}$ or $\mathbf{FPT}_{\mathbf{nu}}$ with parameterizations η and ζ. The *upward closed* property entails that the set is also in the parameterized complexity class with all parameterizations less powerful than η or ζ. The inclusion of greatest lower bounds means that η and ζ can be combined into a parameterization that is at least as powerful as either one of them.

Uniform Complexity Classes

While the absence of uniformity constraints may be accommodating to certain proof methods, uniform parameterized complexity is more practically relevant. Luckily, we can also characterize the structure of parameterizations in relation to uniform parameterized complexity.

3.2.26. THEOREM. *Ordered by $\preccurlyeq$, the equivalence classes in $\mathscr{L}_{\mathbf{XP}}$ and $\mathscr{L}_{\mathbf{FPT}}$ form bounded distributive lattices.*

Proof:
Proving the current theorem requires attention to two aspects that did not play a role in proving Theorem 3.2.24. First, the criterion for membership in $\mathscr{L}_{\mathbf{XP}}$ or $\mathscr{L}_{\mathbf{FPT}}$ is more elaborate than that for membership in their nonuniform counterparts. Second, whenever a bound on the values of a gap function between two parameterizations is employed, that bound must now be computable. Despite these extra elements to consider, a proof of the current theorem may proceed largely along the same lines as the proof of Theorem 3.2.24. For starters, a proof concerning $\mathscr{L}_{\mathbf{XP}}$ is again subsumed in one concerning $\mathscr{L}_{\mathbf{FPT}}$ and we shall therefore present only the latter.

$\mathscr{L}_{\mathbf{FPT}}$ is bounded. The parameterization $(2^+)_{k\in 2^+}$ consisting of full slices is also a least element of $\mathscr{L}_{\mathbf{FPT}}$ with respect to $\preccurlyeq$. Any constant function acts as a computable upper bound on the gap from this parameterization to any other. As, for instance, the empty set is put in **FPT** by this parameterization, it is indeed a member of $\mathscr{L}_{\mathbf{FPT}}$.

In order for a parameterization to be a greatest element of $\mathscr{L}_{\mathbf{FPT}}$, it is no longer sufficient for its slices to be finite. Specifically, the construction of a computable upper bound on the gap function requires knowing the number of instances in each slice. With the length parameterization, $(\{x \mid |x| \leq \mathrm{asInt}(k)\})_{k\in 2^+}$, this extra requirement is satisfied. That it is a member of $\mathscr{L}_{\mathbf{FPT}}$ is, again, witnessed by the empty set, which is put in **FPT** by it. We claim that the gap from any parameterization to the length parameterization is computable. Observe that, for every parameterization η that corresponds to a direct parameterized procedure, the minimization function μ_η is computable. This is useful, because all parameterizations in $\mathscr{L}_{\mathbf{FPT}}$ correspond to direct parameterized procedures. As a result, the gap from a parameterization $\eta \in \mathscr{L}_{\mathbf{FPT}}$ to the length parameterization is also computable. For every argument, the gap is simply the maximum of a known finite set of computable values.

$\mathscr{L}_{\mathbf{FPT}}$ contains greatest lower bounds. A parameterization is in $\mathscr{L}_{\mathbf{FPT}}$ when it corresponds to a direct parameterized procedure that meets the requirements of Definition 3.2.12. A greatest lower bound for two parameterizations η and η' in $\mathscr{L}_{\mathbf{FPT}}$ can be constructed uniformly via such parameterized procedures. Let

ϕ and ϕ' be procedures associated with η and η'. Although ϕ and ϕ' need not converge to the same set, we can combine them in a parameterized procedure ψ defined by

$$\psi(x,k) = \begin{cases} \texttt{?} & \text{if } \phi(x,k) = \texttt{?} \text{ and } \phi'(x,k) = \texttt{?}, \\ \texttt{0} & \text{otherwise.} \end{cases}$$

This parameterized procedure converges to the empty set and witnesses that the empty set is in **FPT**. The parameterization corresponding to ψ is the greatest lower bound ζ as constructed in the proof of Theorem 3.2.24.

What remains is to show that the gap from ζ to η and η' can be bounded by a computable function. To this end, observe that, for any k, an instance is in η_k or η'_k only if it is in $\zeta_{\langle k,k\rangle}$. Combined with the bound $|\langle k,k\rangle| \leq 3 \cdot |k|$, it follows that, for every m, both $\mathrm{gap}_{\zeta,\eta}(m)$ and $\mathrm{gap}_{\zeta,\eta'}(m)$ are bounded by $3m$.

$\mathcal{L}_{\mathbf{FPT}}$ contains least upper bounds. For the presence of least upper bounds, the bounds on the gap in the proof of Theorem 3.2.24 are already computable. The only thing required for the proof to carry over to the uniform case is to show that the parameterization ζ as constructed is a member of $\mathcal{L}_{\mathbf{FPT}}$. Therefore, we turn, like before, to the parameterized procedures ϕ and ϕ' associated with parameterizations η and η' in $\mathcal{L}_{\mathbf{FPT}}$. The difference with the case for greatest lower bounds is that our derived procedure now returns ? if ϕ *or* ϕ' produces ?. Corresponding to this parameterized procedure, we find the parameterization ζ, which is hence a member of $\mathcal{L}_{\mathbf{FPT}}$.

$\mathcal{L}_{\mathbf{FPT}}$ is distributive. The proof of distributivity of the nonuniform lattice is concerned only with obtaining bounds on the gap function. As all these bounds in the proof of Theorem 3.2.24 are computable, the proof works equally well for distributivity of the uniform lattice. □

Also within the uniform lattices, filters are defined by sets. Contrary to the nonuniform case, the sets need to be decidable, for otherwise the induced collection of parameterizations is empty. This is implied by Theorem 3.1.1, which says that direct parameterized procedures can only converge to decidable sets.

3.2.27. THEOREM. *Let A be a decidable set. Ordered by $\preccurlyeq$, the collections $\mathcal{F}_{\boldsymbol{XP}}(A)$ and $\mathcal{F}_{\mathbf{FPT}}(A)$ are filters in $\mathcal{L}_{\boldsymbol{XP}}$ and $\mathcal{L}_{\mathbf{FPT}}$, respectively.*

Proof:
This proof too is presented for **FPT**, but works just as well for ***XP***.

$\mathcal{F}_{\mathbf{FPT}}(A)$ is nonempty. The direct parameterized procedure of which the existence is asserted by Theorem 3.1.3 witnesses that $\mathcal{F}_{\mathbf{FPT}}(A)$ is nonempty.

$\mathcal{F}_{\mathbf{FPT}}(A)$ **is upward closed.** Let η be a parameterization corresponding to a parameterized procedure ϕ that converges to A and meets the requirements of Definition 3.2.12. We shall show that all parameterizations in $\mathcal{L}_{\mathbf{FPT}}$ that are above η according to $\preccurlyeq$ are also in $\mathcal{F}_{\mathbf{FPT}}(A)$. Suppose a parameterization η' in $\mathcal{L}_{\mathbf{FPT}}$ and a computable function g are given such that, for all m, we have $\text{gap}_{\eta,\eta'}(m) \leq g(m)$. By definition of membership of η' in $\mathcal{L}_{\mathbf{FPT}}$, there exists a set B that is in **FPT** with η'. In turn, this means that there is some parameterized procedure ϕ' that meets the requirements of Definition 3.2.12 with respect to B and η'. We need to show that there is also a parameterized procedure that witnesses that A is in **FPT** with η'. Consider a parameterized procedure ψ that does the following on input (x, k').

```
1: If $\phi'(x, k')$ yields ?, we conclude that $x$ is not in $\eta'_{k'}$ and return ?.
2: Otherwise, we conclude that $x$ is in $\eta'_{k'}$ and we need to decide on membership
   of $x$ in $A$. There must be a parameter value $k$ of length at most $g(|k'|)$
   such that $x$ is in $\eta_k$. Therefore, we are sure to return $A(x)$ by proceeding
   as follows for all $k$ of length at most $g(|k'|)$.
    2.1: If $\phi(x, k)$ does not yield ?, return its output.
```

To prove that ψ meets the requirements of Definition 3.2.12, it suffices to show that step 2 does. Crucially, the parameter dependence of the running time of this loop should be bounded by a computable function. If f is a computable function bounding the parameter dependence of the running time of ϕ, the parameter dependence of the loop can be bounded by

$$\sum_{k \text{ with } |k| \leq g(|k'|)} f(k).$$

This bound is computable, and since ψ converges to A by construction, we find that η' puts A in **FPT**. As η' was an arbitrary parameterization above η in the uniform order on parameterizations, we conclude that $\mathcal{F}_{\mathbf{FPT}}(A)$ is upward closed.

$\mathcal{F}_{\mathbf{FPT}}(A)$ **contains greatest lower bounds.** Parameterizations in $\mathcal{F}_{\mathbf{FPT}}(A)$ correspond to direct parameterized procedures that converge to A and meet the requirements of Definition 3.2.12. Thus, parameterizations η and η' taken from $\mathcal{F}_{\mathbf{FPT}}(A)$ are associated to parameterized procedures ϕ and ϕ', both converging to A. These procedures can be combined like in the construction of a greatest lower bound in the proof of Theorem 3.2.26. However, instead of returning 0, we let the constructed procedure return the output of ϕ or ϕ' that is not ?. If both do not output ?, their outputs are the same, since both parameterized procedures converge to A. Thus we obtain a direct parameterized procedure that converges to A on a greatest lower bound of η and η', and meets the requirements of Definition 3.2.12. Because of this, we may conclude that $\mathcal{F}_{\mathbf{FPT}}(A)$ contains all greatest lower bounds. □

By the above theorem, the uniform order $\preccurlyeq$ counts as a ranking of how powerful parameterizations are. Like with the nonuniform order, a parameterization η is more powerful than a parameterization ζ if we have $\eta \preccurlyeq \zeta$. Recall from our discussion of the nonuniform case on page 100 that a parameterization is more powerful than another when its slices grow in bigger steps. Also in the uniform case, if we have $\eta \preccurlyeq \zeta$, we may say that the convergence behavior of η is an improvement over that of ζ. This improvement is of a different kind than the improvements made in typical algorithms races. These races seek improvements within a single parameterization [95, 54], striving for a reduced dependence of the running time on the parameter value. Instead, we suggest a search for improvements in the parameterization itself.

3.2.28. Example. We shall take a closer look at how it is that the order $\preccurlyeq$ relates to the speed of convergence of parameterized decision procedures. To do so, we consider convergence to a single set on different parameterizations. If one parameterization is not below the other, this has implications regarding the shortest parameter value of some instances. Note that if the parameterizations are *incomparable*, then such instances with a difference in parameter values occur both ways. They then show that either parameterization converges faster than the other on *some* instances. Of course, when the parameterizations are *comparable*, yet in different equivalence classes, the improvement holds only in one direction.

Let A be a set, η a parameterization in $\mathscr{F}_{\mathbf{FPT}}(A)$, and ϕ a direct parameterized procedure witnessing that A is in **FPT** with η. For some computable function f and polynomial p, the running time of ϕ on input (x, k) is at most $f(k) \cdot p(|x|)$. Suppose we are given a parameterization ζ that is not below η in the uniform order on parameterizations. We claim that, on infinitely many instances, the parameter dependence of ϕ is less than that of any parameterized procedure converging to A on ζ. If η and ζ are incomparable, there need not exist any parameterized procedure converging to A on ζ. We are only interested in the case where one does exist.

For a fixed parameterization, parameter values are a relative measure of complexity: If an instance x occurs in the parameterization only with a subset of the parameter values with which an instance y occurs in it, then x is harder than y. However, parameter values should not be compared between parameterizations. Indeed, they can differ wildly within an equivalence class of parameterizations. Therefore, we turn to the parameter dependence in the running times of parameterized procedures. With these, we have a more robust way to compare the complexity of instances between different parameterizations.

Suppose the running time of a parameterized procedure that converges to A on ζ is $f'(k) \cdot p'(|x|)$. Here, we may assume that f' is a computable function of which the value increases as the length of the parameter value, $|k|$, increases. Our claim comes down to the observation that there are infinitely many instances x with parameter values k that satisfy

- $x \in \eta_k$, and
- for all k' such that we have $x \in \zeta_{k'}$, we have $f(k) < f'(k')$.

If this were not the case, then for all but finitely many x and k with $x \in \eta_k$, there would be a k' such that we have $x \in \zeta_{k'}$ and $f'(k') \leq f(k)$. This would mean that f and f' could be used to construct a computable function bounding the function $\mathrm{gap}_{\zeta,\eta}$ from above. Such an upper bound cannot exist, because it would imply that we had $\zeta \preccurlyeq \eta$.

Note that while there are infinitely many instances that behave as in the above two items, these instances may be far apart. It is quite possible that for the vast majority of instances, the value of $f(k)$ as in the second item is far greater than that of $f'(k')$. Also, there may be a substantial difference in the polynomial factors, $p(|x|)$ and $p'(|x|)$. More typical algorithms races in the study of parameterized complexity may offer some relief here. These races mainly target the parameter dependence in the running time of direct parameterized procedures associated with a given parameterization.

In this light, the best parameterizations are those that are below all others. A set A admits an *optimal* parameterization with respect to, say, **FPT** if the filter $\mathcal{F}_{\mathbf{FPT}}(A)$ is principal. Indeed, by Definition 2.1.22, a filter is principal precisely when it contains an element that is below all others.

3.2.3 Optimal Nonuniform Parameterizations

We would like to identify the sets that admit optimal parameterizations with respect to any of our parameterized complexity classes. With respect to $\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}$, all sets admit optimal parameterizations.

3.2.29. Theorem. *For any set A, the filter $\mathcal{F}_{\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}}(A)$ is principal.*

Proof:
Let $S_1, S_2, S_3, \ldots$ be an enumeration of the polytime-segments of A. Recall from the proof of Theorem 3.2.4 that this enumeration is nonuniform. For the current theorem, that is acceptable. Consider the parameterization η given by $\eta_k = S_{\mathrm{asInt}(k)}$. By definition, A is in $\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}$ with η. As a consequence of Lemma 3.2.19, it is also below any other parameterization in $\mathcal{F}_{\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}}(A)$: Every slice of every other parameterization in $\mathcal{F}_{\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}}(A)$ is a polytime-segment of A and therefore included as a slice in η. Thus, η is a least element in $\mathcal{F}_{\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}}(A)$. □

We shall call a least element in the filter corresponding to some set a *principal parameterization* for that set. Strictly speaking, a principal parameterization refers to an equivalence class of parameterizations. Yet, a principal parameterization is unique up to $\preccurlyeq$-equivalence. Theorem 3.2.29 shows that all sets have principal parameterizations with respect to $\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}$. However, this is not a given for arbitrary parameterized complexity classes. When they exist, principal parameterizations provide insight into some of the computational complexity of a set. For instance, there is a one-to-one correspondence between the imix property of a principal parameterization with respect to $\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}$ and the levelability of a set. This correspondence arises as a consequence of the proof of Theorem 3.2.29. Specifically, precisely when a set A is P-semi-levelable do each of the sets S_i in the proof of Theorem 3.2.29 have infinite extensions.

3.2.30. Corollary. *A set A is* **P***-semi-levelable (respectively, almost* **P***-bi-immune) if and only if a principal parameterization in $\mathcal{F}_{\mathbf{XP}_{\mathbf{nu}}}(A)$ has (respectively, does not have) imix.*

Note that the filter induced by a **P**-bi-immune set consists of a single equivalence class of parameterizations. Namely it consists only of the class of parameterizations η where for every parameter value k the set η_k is finite. For filters with respect to $\mathbf{FPT}_{\mathbf{nu}}$, this is no different and the filters induced by almost **P**-bi-immune sets are again principal.

3.2.31. Theorem. *For any set A that is almost* **P***-bi-immune, $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$ is principal.*

Proof:
By definition of being almost **P**-bi-immune, A has a maximal polytime-segment S. For some polynomial p, this polytime-segment S is also an $\mathcal{O}(p)$-segment. Thus there exists a parameterization that has S as one of its slices and with which A is in $\mathbf{FPT}_{\mathbf{nu}}$. Indeed, such a parameterization can be constructed along the same lines as in the proof of Theorem 3.2.29. To wit, let $S_1, S_2, S_3, \ldots$ be an enumeration of the $\mathcal{O}(p)$-segments of A and consider the parameterization given by $\eta_k = S_{\mathrm{asInt}(k)}$. That η is indeed a parameterization follows from the fact that it not only contains S as one of its slices, but also all finite variations of S.

Suppose that there is a parameterization ζ in $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$ such that we do not have $\eta \preccurlyeq \zeta$. There would then be a slice of ζ that is an infinite extension of S. As this slice would be the domain of some polytime-approximation for A, we would find that S is not a maximal polytime-segment of A. This would contradict our choice of S, therefore η must be below ζ. Hence η is a principal parameterization for A. □

The key ingredient of the above proof is that, by definition, a polytime-segment of a set is maximal if no infinite extension of it is also a polytime-core. By the same token, a principal parameterization with respect to $\mathbf{FPT}_{\mathbf{nu}}$ for any almost **P**-bi-immune set does not have imix. On the other hand, if the filter with respect to $\mathbf{FPT_{nu}}$ were to be principal for any **P**-semi-levelable set, a principal parameterization has to have imix. The **P**-semi-levelable property is, however, indifferent to the degree of the polynomials involved in the polytime-segments of a set. This makes an investigation of filters with respect to $\mathbf{FPT_{nu}}$ induced by **P**-semi-levelable sets difficult. We need a variant that is sensitive to the degree of the polynomials related to polytime-segments of a set. Conceptually, we want polytime-segments that can be extended with infinitely many elements without increasing the degree of the associated polynomial. Whether or not a set can be approximated by such a sequence of segments that grows in infinitely large steps may depend on the degree of the polynomial. We should not confine ourselves to sets for which an approximation of this sort is available for all degrees. Otherwise, taken to its extreme, we consider only sets of which all $\mathcal{O}(1)$-segments can be extended with infinitely many elements into other $\mathcal{O}(1)$-segments. Therefore, we shall allow finitely many exceptions in our notion of levelability applied to fixed-parameter tractability. The following definition helps us, in Theorem 3.2.33, in pinpointing which sets have optimal parameterizations.

3.2.32. Definition. A set is **FPT**-*semi-levelable* if there is a constant c such that for all polynomials p of degree at least c, the set has no maximal $\mathcal{O}(p)$-segment.

Likewise, we could call a set almost **FPT**-bi-immune if it has a maximal $\mathcal{O}(p)$-segment for all polynomials p of sufficiently high degree. In contrast to the general, degree-independent definitions, these two classifications do not exhaust all sets. Sets may exist that are neither **FPT**-semi-levelable, nor almost **FPT**-bi-immune. Such sets would, however, be **P**-semi-levelable. Optimal parameterizations with respect to $\mathbf{FPT_{nu}}$ do not exist for sets that are **P**-semi-levelable, yet not **FPT**-semi-levelable.

3.2.33. Theorem. *For any set A that is* **P**-*semi-levelable and not* **FPT**-*semi-levelable, $\mathcal{F}_{\mathbf{FPT_{nu}}}(A)$ is nonprincipal.*

Proof:
In case A is **P**-semi-levelable and not **FPT**-semi-levelable, there is an infinite set of polynomials $\{p_1, p_2, p_3, ...\}$ such that, for all i,

- A has a maximal $\mathcal{O}(p_i)$-segment, and
- if S_i is a maximal $\mathcal{O}(p_i)$-segment of A and S_{i+1} is a maximal $\mathcal{O}(p_{i+1})$-segment of A, then $S_{i+1} \setminus S_i$ is infinite.

The first of these items follows from the fact that A is not **FPT**-semi-levelable. The second item is a consequence of A being **P**-semi-levelable.

Let η be any parameterization with which A is in $\mathbf{FPT_{nu}}$. Because the degrees of the polynomials $p_1, p_2, p_3, ...$ must be strictly increasing, there is some i such that the slices of η are all $\mathcal{O}(p_i)$-segments of A. As we have seen in the proof of Theorem 3.2.31, there is also a parameterization ζ in $\mathcal{F}_{\mathbf{FPT_{nu}}}(A)$ that has a maximal $\mathcal{O}(p_{i+1})$-segment of A as one of its slices. By the its maximality, this $\mathcal{O}(p_{i+1})$-segment contains all but at most finitely many elements of any $\mathcal{O}(p_i)$-segment of A. Therefore, we have $\zeta \preccurlyeq_{\mathrm{nu}} \eta$. Moreover, by the second item of the list above, we do *not* have $\eta \preccurlyeq_{\mathrm{nu}} \zeta$. This means that η cannot be a principal parameterization for A. As η was chosen arbitrarily, $\mathcal{F}_{\mathbf{FPT_{nu}}}(A)$ thus cannot be principal. □

Together, Theorem 3.2.31 and Theorem 3.2.33 provide insight in what sets have optimal parameterizations with respect to $\mathbf{FPT_{nu}}$. The resulting, incomplete, classification is depicted in Figure 3.3.

almost **P**-bi-immune	**P**-semi-levelable	
		FPT-semi-levelable
principal	*nonprincipal*	open

Figure 3.3: The universe of sets, represented by the horizontal line, can be divided according to levelability with respect to **P**. Additionally, a subclass of the **P**-semi-levelable sets is **FPT**-semi-levelable. Below the horizontal line, principality of the filter with respect to $\mathbf{FPT_{nu}}$ is indicated. For **FPT**-semi-levelable sets, no results are available.

Of course, Theorem 3.2.33 is only meaningful if there are **P**-semi-levelable sets that are not **FPT**-semi-levelable. This is the case.

3.2.34. THEOREM. *There are* **P***-semi-levelable sets that are not* **FPT***-semi-levelable.*

Proof:
We shall prove the theorem by presenting a **P**-semi-levelable set, A, that has a maximal $\mathcal{O}(p)$-segment for infinitely many polynomials p of distinct degrees. This set A is defined by a recursive procedure that decides on membership in A of an instance based on the membership in A of other instances. As the membership of no instance is, directly or indirectly, dependent on membership of itself, the procedure will terminate on all inputs. It is therefore a decision procedure for A. Alternatively, we could also say that the procedure defines the set A inductively.

Let $\phi_1, \phi_2, \phi_3, \ldots$ be an effective enumeration of all partial procedures and consider the procedure that, on input $\langle u, x\rangle$, procedes as follows.

```
1: We determine a set $I$ of indices of procedures that are consistent with an
   initial segment of $A$:

   1.1: Initialize $I$ to $\{1, 2, 3, \ldots, |\langle u, x\rangle|\}$.
```

```
    1.2: For each pair $\langle v,y\rangle$ that satisfies

             $\text{asInt}(v) \leq \text{asInt}(u)$  and  $|\langle v,y\rangle| \leq \log|\langle u,x\rangle|$

         we remove those indices from $I$ that are not consistent with $A(\langle v,y\rangle)$:
      1.2.1: Recursively compute $A(\langle v,y\rangle)$.
      1.2.2: For each index $i$ in $I$:
        1.2.2.1: Simulate up to $|\langle v,y\rangle|^{3\cdot\text{asInt}(u)}$ steps of $\phi_i$ on input $\langle v,y\rangle$.
        1.2.2.2: If $\phi_i$ was simulated to completion and was inconsistent with
                 $A(\langle v,y\rangle)$ in the sense that we have $\phi_i(\langle v,y\rangle) = 1 - A(\langle v,y\rangle)$,
                 remove $i$ from $I$.

2: We try to make an index in $I$ inconsistent with $A$:

    2.1: For each index $i$ in $I$:
      2.1.1: Simulate up to $|\langle u,x\rangle|^{3\cdot\text{asInt}(u)}$ steps of $\phi_i$ on input $\langle u,x\rangle$.
      2.1.2: If $\phi_i$ was simulated to completion and we have $\phi_i(\langle u,x\rangle) \in \{1,0\}$,
             return $1 - \phi_i(\langle u,x\rangle)$.
    2.2: Else, as no procedure could be made inconsistent, return 0.
```

The first stage of this procedure performs at most $|\langle u,x\rangle|^2$ simulations of computations, each of at most $\log(|\langle u,x\rangle|)^{3\cdot\text{asInt}(u)}$ steps. Besides these simulations, this stage computes an initial segment of A to test against. This segment is computed recursively and the recursion depth is bounded by the iterated logarithm of $|\langle u,x\rangle|$. By using dynamic programming, the time required to compute the segment becomes insignificant with respect to the total running time of the entire procedure.

The second stage of the procedure requires the simulation of at most $|\langle u,x\rangle|$ computations, each of at most $|\langle u,x\rangle|^{3\cdot\text{asInt}(u)}$ steps. Efficient simulation [13] limits the overhead of simulation to a logarithmic factor. As a function of the input $\langle u,x\rangle$, the number of steps spent in the second stage is thus in

$$\mathcal{O}(|\langle u,x\rangle| \cdot |\langle u,x\rangle|^{3\cdot\text{asInt}(u)} \cdot \log(|\langle u,x\rangle|^{3\cdot\text{asInt}(u)})).$$

This puts the number of steps taken by the entire procedure in $\mathcal{O}(|\langle u,x\rangle|^{3\cdot\text{asInt}(u)+2})$. Note that the running time of the procedure is not polynomial in the length of the input, $|\langle u,x\rangle|$, as u appears in the exponent.

Given a constant c, let p_c be the polynomial defined by $p(n) = n^{3c+2}$. For any fixed c, the set $\{\langle u,x\rangle \mid \text{asInt}(u) \leq c \text{ and } x \in 2^+\}$ is an $\mathcal{O}(p_c)$-segment of A. It is not a maximal polytime-segment, as for larger values of c infinitely many elements are introduced in the corresponding sets. However, we claim that it is a maximal $\mathcal{O}(p_c)$-segment of A and thus that A is **P**-semi-levelable, yet not **FPT**-semi-levelable. Suppose toward a contradiction that there is an infinite

$\mathcal{O}(p_c)$-segment $S \subseteq \{\langle w, x\rangle \mid \text{asInt}(w) > c \text{ and } x \in 2^+\}$ for A. Let i be an index of an $\mathcal{O}(p_c)$-approximation for A with domain S. Almost all $\langle w, x\rangle \in S$ will be so that $|\langle w, x\rangle| \geq i$. Therefore, for almost all inputs to our procedure, i will be included in I when the procedure enters its second stage. Because we have $\text{asInt}(w) > c$, we have $3 \cdot \text{asInt}(w) > 3 \cdot c + 2$. This means that any function in $\mathcal{O}(p_c)$ will eventually be dominated by the polynomial mapping n to $n^{3 \cdot \text{asInt}(w)}$, for any value of w that occurs in S. Indeed, this is the reason for the constant 3 in the exponent of the time bounds in our decision procedure for A. Now, for almost all elements of S, the second stage of our procedure does one of two things. Either it invalidates i as the index of an $\mathcal{O}(p_c)$-approximation for A, or it invalidates an index smaller than i. The latter of these possibilities can happen at most $i - 1$ times, so, since S was assumed to be infinite, eventually i must be invalidated. This contradicts our choice of i as the index of an $\mathcal{O}(p_c)$-approximation for A with domain S. We conclude that there is no infinite $\mathcal{O}(p_c)$-segment $S \subseteq \{\langle w, x\rangle \mid \text{asInt}(w) > c \text{ and } x \in 2^+\}$ for A. □

For completeness, we shall also show the existence of **FPT**-semi-levelable sets. Our proof revolves around length-increasing reductions. Our notion of a *reduction* is that of a membership-preserving polytime-computable function, in other words, that of a Karp reduction.

3.2.35. Theorem. *Every set outside* **P** *from which there is a linearly-length-increasing reduction to itself is* **FPT***-semi-levelable.*

Proof:
Let A be a set outside **P** and f a linearly-length-increasing reduction from A to itself. Suppose that A is not **FPT**-semi-levelable and, for some polynomial p of degree c, has a maximal $\mathcal{O}(p)$-segment S. We may assume that, for some polynomial q of degree $c - 1$, it is possible to compute f in time $\mathcal{O}(q)$. The sets

$$\begin{aligned} S' &= \{x \mid x \notin S \text{ and } f(x) \in S\}, \\ S_x &= \{x, f(x), f(f(x)), \ldots\} \end{aligned}$$

are, by nature of f, also $\mathcal{O}(p)$-segments of A. Furthermore, S' satisfies $S \cap S' = \emptyset$.

That S' is a $\mathcal{O}(p)$-segment of A requires the linear bound on the length of the output of f. A given string $x \in S'$ is a member of A precisely when $f(x)$ is. Because $f(x)$ is a member of S and S is a $\mathcal{O}(p)$-segment of A, membership of $f(x)$ in A can be decided by some $\mathcal{O}(p)$-approximation for A. Correspondingly, there is an $\mathcal{O}(p)$-approximation for A of which S' is the domain.

That S_x is a $\mathcal{O}(p)$-segment of A requires that f is length-increasing. For a fixed x, we can decide whether a given string y is a member of S_x by repeated application of f to x. If y is a member of S_x, then it is a member of A precisely when x is. Indeed, S_x is either a subset of A, or a subset of the complement of A.

By the assumed maximality of S, for every x there are only finitely many elements in the set $S \setminus S_x$. However, since A is not in $\mathbf{P}$, there are infinitely many x outside S and for each of these the set S_x intersects S'. Since f is length-increasing, there is no uniform upper bound on the length of the strings in these intersections for the infinitely many values of x. Hence S' is infinite, contradicting the maximality of S. □

The existence of **FPT**-semi-levelable sets now follows from the existence of sets outside **P** that have a linearly-length-increasing reduction to themselves.

3.2.36. Lemma. *There is a set outside* **P** *that has a linearly-length-increasing reduction to itself.*

Proof:
Let X be a set outside **P** and consider its cylindrification [see 17, Section 5.3]

$$A = \{\langle x, y\rangle \mid x \in X \text{ and } y \in \mathtt{2}^+\}.$$

Note that A too is not in **P**. Our pairing function is such that the function f defined by

$$f(\langle x, y\rangle) = \langle x, \mathtt{0}y\rangle$$

is a linearly-length-increasing reduction from A to itself. □

We remark that the cylindrification of a set is always a p-cylinder, because there exists a polytime-computable bijection between $\mathtt{2}^+$ and $\mathtt{2}^+ \times \mathtt{2}^+$.

3.2.4 Optimal Uniform Parameterizations

The proof of Theorem 3.2.29 fails for filters with respect to the uniform $\boldsymbol{X}$**P**. First and foremost, by Rice's theorem [125], the enumeration of polytime-segments is not effective. Therefore, the proof cannot guarantee the existence of a procedure that meets the requirements of the definition of **XP**, Definition 3.2.10. Additionally, being a principal parameterization with respect to a uniform complexity class requires a bound on the gap to any other parameterization to be computable.

We regain something akin to Theorem 3.2.29 by also taking into account *provability* of membership of a set in $\boldsymbol{X}$**P**. Employing provability when comparing computational complexities is a tactic that has also been used by Hartmanis [75]. To make our use of provability precise, let $\mathfrak{F}$ be a formal system capable of expressing statements about computation. A proof in $\mathfrak{F}$ is thus a syntactic derivation of a theorem on computation. For convenience, we stretch the notion of what can be proven in $\mathfrak{F}$ slightly.

3.2.37. Definition. Let $\hat{A}$ be a decision procedure for a set A. We say that a parameterization η *provably* puts $\hat{A}$ in $\boldsymbol{X}$**P** if there is a direct parameterized procedure ϕ and a computable function f satisfying

- η is the parameterization corresponding to ϕ, and
- for every parameter value k, with t_k mapping n to $f(k) \cdot n^{f(k)}$, there is a proof in $\mathfrak{F}$ of the following statement, expressed in the language of $\mathfrak{F}$.

  The partial application of ϕ to k yields a t_k-approximation for the set decided by $\hat{A}$.

Remark that there are many decision procedures for the set A and that $\hat{A}$ is just one of them. It is necessary to fix a decision procedure in the above definition in order to talk about A in the language of $\mathfrak{F}$. If we did not fix a decision procedure, it would not be possible to formulate the second of the required statements in the language of $\mathfrak{F}$. This would make it impossible for it to have a proof in $\mathfrak{F}$.

The proofs that are at play in the previous definition are not required to be computable from the parameter value k. Even without such a uniformity constraint, the following theorem, similar in spirit to Theorem 3.2.29, holds. In the following theorem, we assume that $\mathfrak{F}$ is sound and strong enough to express certain arguments. In particular, we assume that some of the reasoning performed in the proof of the theorem can be formalized in $\mathfrak{F}$. Precisely where these assumptions are made will be mentioned in the proof. Note that in this theorem, the underlying order on parameterizations is the uniform order, $\preccurlyeq$.

3.2.38. Theorem. *For any decision procedure $\hat{A}$ for a set A, there is a least parameterization among those provably putting $\hat{A}$ in* $\boldsymbol{X}\mathbf{P}$.

Proof:
A form of universal search, along the lines of Hutter [86] and Hartmanis [75], through polytime-approximations is possible in the parameterized setting. In order to perform such a search, let $\phi_1, \phi_2, \phi_3, \ldots$ be an effective enumeration of all procedures, where the procedures need not be total. Additionally, let $p_1, p_2, p_3, \ldots$ be an effective enumeration of all polynomials. Consider the parameterized procedure converging to A that, on input (x, k), does the following.

```
1: We construct a finite set $M$ of polytime-approximations for $A$:
   1.1: Initialize $M$ to the empty set.
   1.2: For each combination of a proof $\mathfrak{p}$ in $\mathfrak{F}$, an index $i$ of a parameterized
        procedure, and an index $j$ of a polynomial, all of length at most asInt($k$):
        1.2.1: If $\mathfrak{p}$ proves that $\phi_i$ is a $p_j$-approximation for $A$,
                 Add $i$ to $M$.
2: If any of the polytime-approximations in $M$ decides on membership of $x$,
   then so do we:
   2.1: For each index $i$ in $M$:
       2.1.1: Compute $\phi_i(x)$.
       2.1.2: If $\phi_i(x) \in \{1, 0\}$, return $\phi_i(x)$.
   2.2: Else, as none of the polytime-approximation decides on membership
        of $x$, return ?.
```

We shall first show that the parameterization corresponding to this procedure provably puts $\widehat{A}$ in $\boldsymbol{X}\mathbf{P}$. After that, we shall show that the gap from that parameterization to any other that provably puts $\widehat{A}$ in $\boldsymbol{X}\mathbf{P}$ can be bounded by a computable function.

For all parameter values k, the set M is finite throughout the execution of this parameterized procedure. As a result, the first stage of this procedure takes a finite number of steps and the second stage can be executed in polynomial time, for every fixed k. Moreover, we claim that the parameter dependence is computable. For the first stage, the parameter dependence can be computed by simply performing the prescribed computation and clocking the number of steps taken. This works as expected by our assumption that $\mathfrak{F}$ is sound. Observe that we can also keep track of all the polynomials p_j associated with indices i added to M in step 1.2.1. Doing so, we can upper bound the running time of the second stage by the sum of all these polynomials. Because this bound depends solely on the parameter value, we find that we can construct a computable function f as required by the definition of $\boldsymbol{X}\mathbf{P}$, Definition 3.2.10. Thus, A is in $\boldsymbol{X}\mathbf{P}$ with the parameterization corresponding to the parameterized procedure above. We assume that this argument can be expressed in $\mathfrak{F}$. More accurately, we assume that there is a proof in $\mathfrak{F}$ of the fact that for any fixed k, the above procedure is an $(f(k) \cdot n^{f(k)})$-approximation for the set decided by $\widehat{A}$. It then follows that the parameterization corresponding to the parameterized procedure above provably puts $\widehat{A}$ in $\boldsymbol{X}\mathbf{P}$.

Let ζ be a parameterization that provably puts $\widehat{A}$ in $\boldsymbol{X}\mathbf{P}$. Additionally, let ψ be a direct parameterized procedure and f a computable function witnessing that ζ provably puts $\widehat{A}$ in $\boldsymbol{X}\mathbf{P}$, as in Definition 3.2.37. Denote the parameterization corresponding to the parameterized procedure we constructed by η. We claim that there is a computable function g such that, for all k', slice $\zeta_{k'}$ is included in slice $\eta_{g(k')}$. This entails the desired relationship $\eta \preccurlyeq \zeta$. Given ψ and k' we can effectively come up with an index i such that ϕ_i corresponds to the partial application of ψ to k'. Correspondingly, using f we can come up with an index j such that ϕ_i is a p_j-approximation for A. By construction, there is a proof in $\mathfrak{F}$ of the fact that ϕ_i is a p_j-approximation for the set decided by $\widehat{A}$. Such a proof, $\mathfrak{p}$, can be found effectively by enumerating all proofs. Now, consider the function defined by

$$g(k') = \text{asStr}(\max\{|i|, |j|, |\mathfrak{p}|\}).$$

As i, j, and $\mathfrak{p}$ were derived effectively from ψ, f, and k', this is a computable function of k' when ψ and f are fixed. For any k', the first stage of the above parameterized procedure includes the corresponding i in the set M when we have $k = g(k')$. The second stage of the procedure then ensures that slice $\zeta_{k'}$ is included in slice $\eta_k = \eta_{g(k')}$, as we set out to prove. □

Thus, adding a provability requirement offsets the limitations we incurred by moving to a uniform setting. The provability requirement in Theorem 3.2.38 enforces the effectiveness that was not present in the proof of Theorem 3.2.29. With respect to **FPT**, the proof of Theorem 3.2.31 cannot be reused to show that almost **P**-bi-immune sets induce principal filters. Specifically, it is no longer sufficient for the gap from one parameterization to another to take on only finite values. Instead, there must be a uniformly computable bound on the value of the gap function. Of course, we *can* show that filters with respect to **FPT** are principal for sets in **P**.

3.2.39. Theorem. *For any set A that is in* **P**, *the filter $\mathscr{F}_{\mathbf{FPT}}(A)$ is principal.*

Proof:
The parameterization consisting of full slices, $(2^+)_{k\in 2^+}$, is one with which A is in **FPT**. Since the equivalence class of this parameterization is the least element of the encompassing lattice of parameterizations, the filter must be principal. □

Principal parameterizations with respect to **FPT** for sets in **P** do not have imix. More broadly, if an almost **P**-bi-immune set has a principal parameterization with respect to **FPT**, this parameterization does not have imix. This is the extent to which the proof of Theorem 3.2.31 can be applied to the uniform setting. More precisely, a principal parameterization must contain a maximal polytime-segment as one of its slices and can therefore not have imix. Conversely, if a set has a principal parameterization with respect to **FPT** that does have imix, the set must be **P**-semi-levelable.

3.2.40. Lemma. *A set that, with respect to* **FPT**, *has a principal parameterization without imix is almost* **P**-*bi-immune.*

Proof:
A parameterization η without imix has a slice η_k such that for all other slices $\eta_{k'}$ the difference $\eta_k \setminus \eta_{k'}$ is finite. Of any set A that is in **FPT** with η, this slice is a polytime-segment. If η is a principal parameterization for A, then η_k is even maximal up to finite variations for all slices of all parameterizations in $\mathscr{F}_{\mathbf{FPT}}(A)$. Because every polytime-segment of A can be turned into a slice of a parameterization with which A is in **FPT**, this means that A must be almost **P**-bi-immune. □

This observation is of use as we turn to principality of filters with respect to **FPT** for **P**-semi-levelable sets. From the point of view of applications, the **P**-semi-levelable sets are more interesting than the almost **P**-bi-immune sets. With almost

P-bi-immune sets, there is a clear limit to what can be achieved in polynomial time by any decision procedure. No such limit exists for **P**-semi-levelable sets. This is where parameterized decision procedures come into play, as they have the potential to combine better and better approximations. Interestingly, it turns out that there is no optimal parameterization with respect to **FPT** for a **P**-semi-levelable set. In that sense, each direct parameterized procedure that shows that a **P**-semi-levelable set is fixed-parameter tractable is restricted in its own way. For some **P**-semi-levelable sets, this follows already from the proof of Theorem 3.2.33, which works for the uniform setting as well. In the uniform setting, however, the reach of the result can be extended. We shall do so in a way that has a clear kinship to the diagonal argument used in the proof of the time hierarchy theorem of Hartmanis and Stearns [77]. However, where the time hierarchy theorem constitutes a hierarchy of *sets*, our theorem is about a hierarchy of *parameterized procedures.*

3.2.41. Theorem. *For any set A that is* **P***-semi-levelable, the filter $\mathscr{F}_{\mathbf{FPT}}(A)$ is nonprincipal.*

Proof:
We present a proof by contradiction, assuming η is a principal parameterization with respect to **FPT** for a given **P**-semi-levelable set A. A contradiction is arrived at by the construction of a parameterization ζ with which A is in **FPT** and for which we have $\eta \not\preceq \zeta$. Let ϕ be a direct parameterized procedure and p a polynomial witnessing that A is in **FPT** with η in accordance with Definition 3.2.12. Consider a direct parameterized procedure that converges to A and proceeds as follows on input (x, k).

```
1: Set a timeout so that at most $\mathrm{asInt}(k) \cdot |x|^2 \cdot p(|x|)$ steps are spent in total
   doing the following, for each $j \in \mathbb{N}$ in sequence:
    1.1: Compute $\phi(x, \mathrm{asStr}(j))$.
    1.2: If $\phi(x, \mathrm{asStr}(j)) \in \{1, 0\}$, return $\phi(x, \mathrm{asStr}(j))$.
2: Else, as no decision about $x$ could be reached in time, return ?.
```

Let ζ be the parameterization corresponding to this direct parameterized procedure. The dependence on the parameter value in the self-imposed running-time bound ensures that ζ is in fact a parameterization. Because the dependence on the length of the instance is a polynomial of which the degree does not depend on the parameter value, A is in **FPT** with ζ.

It remains to show that we have $\eta \not\preceq \zeta$. Observe that our procedure computes values of $\mathcal{O}(p)$-approximations for A until it encounters an approximation of which x is in the domain. By merit of the $|x|^2$ factor in the running-time bound, the number of approximations that our procedure can compute increases as a function of $|x|$. In fact, this number increases without a bound for any constant value of k. Since A is **P**-semi-levelable and therefore not almost **P**-bi-immune, it follows from Lemma 3.2.40 that η must have imix. Therefore, even for a fixed value of k, our procedure is able to decide membership of instances x for which $\mu_\eta(x)$ is arbitrarily high. Hence, the gap from η to ζ is the function that is infinite everywhere, proving $\eta \not\preccurlyeq \zeta$. □

As mentioned after Definition 3.2.8, the class of **P**-semi-levelable sets includes many natural sets [117]. The theorem tells us that these sets have no optimal parameterizations with respect to **FPT**. For sets that do have optimal parameterizations, the parameterizations are not very promising from an applications point of view. In light of Lemma 3.2.40, we get from the above theorem that no principal parameterization in a filter of the form $\mathcal{F}_{\mathbf{FPT}}(A)$ has imix. However, only parameterizations with imix are indicative of a successful parameterized attack on the complexity of a set. This was noted already on page 92, following Example 3.2.16. Therefore, only parameterized algorithms that converge on parameterizations with imix could be called *attractive.* Thus, we get the following.

3.2.42. Slogan. *No set for which a parameterized algorithm is attractive admits an optimal parameterization.*

We note that when, for a set A, the filter $\mathcal{F}_{\mathbf{FPT}}(A)$ is nonprincipal, there are truly infinitely many distinct parameterizations with which A is in **FPT**. Recall from Definition 2.1.21 that filters are closed under taking greatest lower bounds. Because of this, any finite number of parameterizations in $\mathcal{F}_{\mathbf{FPT}}(A)$ can be combined into one that is below all of them and still in $\mathcal{F}_{\mathbf{FPT}}(A)$. If $\mathcal{F}_{\mathbf{FPT}}(A)$ is principal, there is also a parameterization that is below any finite number of parameterizations, namely any principal parameterization. However, this is not the case if $\mathcal{F}_{\mathbf{FPT}}(A)$ is nonprincipal. Then, there is an infinite number of parameterizations that cannot be combined into one that is still in $\mathcal{F}_{\mathbf{FPT}}(A)$.

Often, parameterizations have a clear interpretation as a structural property of instances. This is for instance the case with graph-based parameterizations such as “maximum vertex degree”, “minimum vertex cover size”, or “treewidth”. From this interpretation, we get an alternative take on the above slogan. Suppose that a parameterized algorithm is attractive for a set A, and thus that $\mathcal{F}_{\mathbf{FPT}}(A)$ is nonprincipal. In that case, there are infinitely many structural properties that may make membership of an instance in A easy to decide. Consequently, certain structural properties, such as those corresponding to the graph-based parameterizations just mentioned, are always only part of the story. An analysis of the computational complexity of a set cannot be limited to any of these parameterizations.

3.2.43. Example. A consequence of Lemma 3.2.40 and Theorem 3.2.41 is that many popular parameterization such as “treewidth” cannot be optimal. If deciding membership in a set A is fixed-parameter tractable with respect to treewidth, there are a few possibilities.

One possibility is that the set A is in **P**. In this case an analysis of A in terms of fixed-parameter tractability is not necessary. Certainly, the treewidth parameterization is not optimal in this case. More broadly, A may be almost **P**-bi-immune. As we observed in the lead-up to Lemma 3.2.40, a principal parameterization for a almost **P**-bi-immune set does not have imix. There are, however, infinitely many graphs with any given treewidth, so the treewidth parameterization has imix. Thus, treewidth cannot be an optimal parameterization for the set A if A is almost **P**-bi-immune.

Another possibility is that A is **P**-semi-levelable. Then, the optimality of the treewidth parameterization for A is ruled out by Theorem 3.2.41. Any set is either almost **P**-bi-immune or **P**-semi-levelable, so we may conclude that the treewidth parameterization is *never* an optimal parameterization.

We now have a characterization of which filters with respect to **FPT** are principal and which are not. In Figure 3.4, this characterization, a combination of Theorem 3.2.39 and Theorem 3.2.41 is summarized visually.

almost **P**-bi-immune		**P**-semi-levelable
P		
principal	open	*nonprincipal*

Figure 3.4: The uniform counterpart to Figure 3.3. For almost **P**-bi-immune sets outside **P**, no results regarding the principality of filters with respect to **FPT** are available.

3.3 as Algorithmic Complexity

In the search for sources of computational complexity, the complexity cores of Lynch get no further than the identification of hard subsets. Because finite variations of a complexity core are complexity cores too, the notion does not lead to any useful formulation of single-instance complexity. With algorithmic complexity [100], it is possible to define a measure of computational complexity of individual instances. To overcome the difficulty of finite variations, this definition takes into account the sizes of decision procedures for a set [94, 119].

3.3.1. Definition. Given a function t, the t-bounded *instance complexity* of a string x relative to a set A is

$$\mathrm{ic}^t(x : A) = \min\{|\phi| \mid \phi \text{ is a } t\text{-approximation for } A \text{ and } x \in \mathrm{dom}(\phi)\}.$$

If there is no t-approximation for A that includes x in its domain, then $\mathrm{ic}^t(x : A)$ is undefined. As we shall soon see, this can only happen when t grows rather slowly.

Recall that the length of an approximation is only defined relative to a chosen encoding of procedures. However, as long as the encodings we choose from are all effective, each encoding can be converted to another. Therefore, the effect of the choice of an encoding on the instance complexity is limited to an additive constant [119, 100].

3.3.2. Example. So far, we have used the word "tractable" to indicate computation that does not take too long. However, the word "tractable" can also be applied to the implementation of algorithms, in which case it means something else. Informally speaking, an algorithm has a tractable implementation if it has an implementation that is not too unwieldy. The implementation should not use too many variables, has no excessive nesting of conditionals and loops, and performs relatively few operations in each loop. In other words, a tractable implementation of an algorithm is a short implementation of that algorithm.

Instance complexity alludes to this notion of complexity. It is concerned with algorithms that are approximations of a decision procedure for a set A. That is, algorithms that decide on membership of an instance either correctly or not at all. With the time bound, we restrict our attention to approximation algorithms that showcase limited computational complexity. Among such algorithms, the instance complexity selects the length of the shortest algorithm that is able to decide on membership of a given instance x. Note that instance complexity is uncomputable. This fact is closely related to the fact that the collection of approximation algorithms for a given set is undecidable.

We observe that there is are constants c_1 and c_2 such that we have, for all t and x, irrespective of A,

$$t(|x|) \geq c_1 \cdot |x| \implies \mathrm{ic}^t(x : A) \leq |x| + c_2. \tag{3.3}$$

In other words, if t is sufficiently large, the instance complexity is defined and can be bounded by the length of the instance plus an additive constant. This is true because for each instance x, an approximation tailored specifically for that x is available that meets the upper bound. This approximation contains a hardcoded copy of x and a record of whether or not x is a member of A. On an input y, it compares y to x and if they are equal, it outputs the hardcoded membership decision. If y and x differ, the approximation outputs ?. The time needed by this approximation is linear in $|x|$, and it can be represented in a number of bits equal to $|x|$ plus an additive constant.

A modification of the above argument allows us to show an upper bound tighter than (3.3). When t is sufficiently large, the instance complexity can be upper bounded by the Kolmogorov complexity. Instead of storing a hardcoded copy of the instance x in our approximation, we now store a procedure that reconstructs x. This may lower the length of the approximation, but increases its computation time, as it now also needs to reconstruct x. Indeed, what counts as "sufficiently large" for t depends on x.

We remind ourselves of the fact that approximations are *total* functions. Broadening the definition of instance complexity by allowing procedures that do not halt, we would obtain a weaker notion of instance complexity [98]. We shall not make use of this weaker notion and stick with the definition based on approximations. Even with this stronger notion, it is in general impossible to know whether a given procedure is an approximation for some specific set. It was for this reason that a provability requirement was necessary in Theorem 3.2.38. The other way around, we can start out with a known collection of approximations for a set. Our parameterized framework provides a way to analyze the computational hardness of instances in relation to such a collection of approximations. Of course, in our analysis these collections will be parameterizations.

Synopsis

In some ways, the study of parameterized computational complexity is a continuation of the study of instance complexity. Precisely how the two fields are related is examined in Section 3.3.1. We find that in the context of uniform fixed-parameter tractability, instance complexity can be used to lower-bound parameter values. Conversely, the minimization function of a parameterization can be used as an upper bound on instance complexity. The quality of this upper bound follows our order on parameterizations: A parameterization that is below another is associated with a tighter bound on instance complexity. Additionally, we find that there are infinitely many instances without parameter values smaller than the Kolmogorov complexity of the instance. Thus, the bound provided by a minimization function is not perfect.

We delve deeper into the interplay between algorithmic and computational complexity in Section 3.3.3. Before that, however, we look at how the parameterizations with which a set is fixed-parameter tractable typify the computational complexity of the set. Each parameterization represents a distribution of complexity. The collection of parameterizations with which a set is fixed-parameter tractable is thus a profile of the computational complexity of the set. In Section 3.3.2, we explore what notion of computational complexity these complexity profiles capture. Given the connections between minimization functions for parameterizations and instance complexity, our notion of complexity lives at the level of instances. We observe that the collection of parameterizations with which a set is fixed-parameter tractable is insensitive to polynomial-time postprocessing: Adding a polynomial-time postprocessing step to a decision procedure for a set A yields a decision procedure for a set with the same complexity profile as A. This inspires a comparison with the Berman–Hartmanis conjecture. That conjecture asserts that the class of **NP**-complete sets is closed under polytime-computable isomorphisms. We assert something similar, but using polynomial-time postprocessing instead of polytime-computable isomorphisms. We find that the anticipated closure property *does not* hold for the case of nonuniform fixed-parameter tractability. We leave it as a conjecture that it *does* hold for the case of uniform fixed-parameter tractability. In particular, we conjecture that the collection of parameterizations that make a given set fixed-parameter tractable determines the set up to polynomial-time postprocessing.

We point out that the last statement contains a perspective orthogonal to that of the previous section, Section 3.2. In that section, we looked at collections of parameterizations that make a set fixed-parameter tractable. In the current, we fix a collection of parameterizations and look at which sets are made fixed-parameter tractable by precisely those parameterizations.

Having established a measure of the distribution of computational complexity based on parameterizations, we can compare to other distributions of complexity. Especially, in Section 3.3.3, we compare computational complexity to algorithmic complexity. At the level of instances, a link between the running time of decision procedures and the length of their specification emerges. In one direction, this link is particularly strong, and we manage to show that algorithmically random instances are computationally hard.

3.3.1 Instance Complexity

The guiding principles behind instance complexity are similar to those behind parameterized computational complexity. Both are attempts to quantify the computational complexity of deciding on membership of specific instances in a set. This similarity can be made precise by linking instance complexity to *nonuniform* fixed-parameter tractability. Nonuniformity is appropriate here, because in the definition of instance complexity, Definition 3.3.1, a different approximation may be used for every instance. At the same time, our parameterized framework also has a uniform side and thus offers a uniform alternative for instance complexity.

The interface between instance complexity and parameterized complexity is provided by a parameterization that is defined in terms of instance complexity. Recall from Example 3.3.2 that there are polynomials p such that the p-bounded instance complexity relative to a set A is defined on all instances. In fact, very many polynomials satisfy the antecedent in (3.3), as it is a mild requirement.

3.3.3. Theorem. *Let p be a polynomial such that the p-bounded instance complexity is defined on all instances. Any set A is in $\mathbf{FPT_{nu}}$ with the parameterization*

$$\eta = (\{x \mid \mathrm{ic}^p(x : A) \le \mathrm{asInt}(k)\})_{k \in 2^+}.$$

Proof:
To begin with, we shall verify that η as defined in the theorem is indeed a parameterization. By construction, the slices of η satisfy

$$\forall k, k' \colon \ \mathrm{asInt}(k) \le \mathrm{asInt}(k') \implies \eta_k \subseteq \eta_{k'}.$$

Combined with the fact that, by assumption, each instance has a finite p-bounded instance complexity, it follows that η is a parameterization. In fact, it is even a point-cofinite parameterization.

For any value of k, there are only finitely many p-approximations for A of length at most $\mathrm{asInt}(k)$. These can be combined into a single $\mathcal{O}(p)$-approximation for A of which the domain is exactly η_k. While this is true for any value of k, the uncomputability of instance complexity prevents this combining process to be effective uniformly in k. However, it does show that A is in $\mathbf{FPT_{nu}}$ with parameterization η. □

The above theorem shows a relation between polynomially-bounded instance complexity and nonuniform fixed-parameter tractability. It tells us that each polynomially-bounded instance complexity relative to a set A is the minimization function of some parameterization in $\mathcal{F}_{\mathbf{FPT_{nu}}}(A)$. More generally, we may think of any minimization function of a parameterization, and, by extension, of any parameterization, as a measure of complexity.

For polynomials of different degrees, the resulting notion of polynomially-bounded instance complexity may differ. Indeed, each notion of polynomially-bounded instance complexity may correspond to a different parameterizations in the filter $\mathcal{F}_{\mathbf{FPT_{nu}}}(A)$. Consequently, there is no single parameterization in $\mathcal{F}_{\mathbf{FPT_{nu}}}(A)$ that represents 'the' polynomially-bounded instance complexity. The relation between polynomially-bounded instance complexity and nonuniform slicewise $\mathbf{P}$ is stronger. For $\boldsymbol{X}\mathbf{P_{nu}}$, we can link polynomially-bounded instance complexity to a *principal* parameterization. The following theorem is essentially an adaptation of Theorem 3.2.29.

3.3.4. Theorem. *For any set A, the parameterization*

$$\eta = (\{x \mid \exists p, \textit{ a polynomial: } \mathrm{ic}^p(x : A) \leq \mathrm{asInt}(k)\})_{k \in 2^+}$$

is a principal parameterization for A with respect to $\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}$.

Proof:
It may seem to make little sense to allow the polynomial p to depend on the instance x in the definition of η. However, this is perfectly fine for our purposes. Like in the proof of the previous theorem, for any value of k, there are only finitely many polytime-approximations for A of length at most $\mathrm{asInt}(k)$. Again, these can be combined into one polytime-approximation for A of which the domain is exactly η_k.

Observe that each polytime-approximation for A is taken into account in the combined approximation corresponding to some parameter value k. As a result of this, η is a least element in $\mathcal{F}_{\boldsymbol{X}\mathbf{P}_{\mathbf{nu}}}(A)$. □

The previous two theorems show that nonuniform parameterized complexity is closely related to instance complexity. Indeed, both notions are attempts at quantifying computational complexity at the level of individual instances. We cannot compare the two directly, since instance complexity is a function, whereas a parameterization is a family of sets. Recall that instance complexity is defined as the minimum of a set of lengths. This suggests that, given a polynomial p, there may be a parameterization η such that, for all instances x of a set A, we have, $\mathrm{ic}^p(x : A) = \mu_\eta(x)$. This would show that our parameterized framework incorporates instance complexity as a notion of complexity. The parameterization η of Theorem 3.3.3 comes close to achieving this, but does not correctly link parameter values to p-approximations for A. Instead, parameter values are linked to *the length of* p-approximations. Specifically, the parameterization η of Theorem 3.3.3 is so that we have $\mu_\eta(x) = |\mathrm{asStr}(\mathrm{ic}^p(x : A))|$. In other words, $\mu_\eta(x)$ is within one bit of $\log \mathrm{ic}^p(x : A)$. It would be nicer if we could choose η so that $\mu_\eta(x)$ is equal to $\mathrm{ic}^p(x : A)$. This can be achieved by switching to the parameterization

$$\zeta = (\{x \mid \mathrm{ic}^p(x : A) \leq |k|\})_{k \in 2^+}.$$

In this parameterization, it is *the length of* parameter values that is linked to *the length of* p-approximations. The parameterization ζ could be used in place of η in Theorem 3.3.3 since the two are equivalent in the uniform order on parameterizations. Notice that in our parameterized analysis of complexity, we are mostly interested in equivalence classes of parameterizations. The equivalence of η and ζ above therefore highlights a difference in focus between parameterized computational complexity and instance complexity. With parameterized complexity, the focus is more on the inclusion relation between slices than on the specific parameter values associated with slices. At its core, a parameterization is a way to group instances of comparable complexity. These groups of instances are the slices of the parameterization.

Nevertheless, we can use the above insights to define a notion of *polytime-bounded instance complexity* based on Theorem 3.3.4. Similarly to how we altered the parameterization central to Theorem 3.3.3, we can switch to the parameterization

$$\zeta = (\{x \mid \exists p, \text{ a polynomial}\colon\ \mathrm{ic}^p(x : A) \leq |k|\})_{k \in 2^+}$$

in Theorem 3.3.4. The theorem then allows us to define the polytime-bounded instance complexity of a string x relative to A as $\mu_\zeta(x)$. Note that if we expand the definition of ζ, we find

$$\begin{aligned}\zeta = (\{x \mid \exists \phi\colon\ & |\phi| \leq |k| \ \text{ and} \\ & \phi \text{ is a polytime-approximation for } A \ \text{ and} \\ & x \in \mathrm{dom}(\phi)\})_{k \in 2^+}.\end{aligned}$$

Thus, the polytime-bounded instance complexity of x is the length of the shortest polytime-approximation for A that decides on membership of x. Similarly, for a polynomial p, it is possible to define the $\mathcal{O}(p)$*-bounded instance complexity* of x. This would be the length of the shortest $\mathcal{O}(p)$-approximation for A that includes x in its domain.

Except relative to some rather trivial sets, the $\mathcal{O}(p)$-bounded instance complexity is not computable. Its practical usefulness is therefore limited. In our parameterized framework, we can look at the possibilities for a *computable* alternative. We do so by turning to the class of *uniformly* fixed-parameter tractable sets, **FPT**, and the corresponding class of parameterizations, $\mathcal{L}_{\mathbf{FPT}}$. Relative to a set A, the $\mathcal{O}(p)$-bounded instance complexity is the minimization function of

$$\zeta = (\{x \mid \exists q \in \mathcal{O}(p)\colon\ \mathrm{ic}^q(x : A) \leq |k|\})_{k \in 2^+}. \tag{3.4}$$

If the $\mathcal{O}(p)$-bounded instance complexity is not computable, then this parameterization ζ is not a member of $\mathcal{F}_{\mathbf{FPT}}(A)$, or even of $\mathcal{L}_{\mathbf{FPT}}$. Note, though, that a parameterization in $\mathcal{L}_{\mathbf{FPT}}$ may share any finite number of slices with ζ. This is so because while there may be no effective way to enumerate all $\mathcal{O}(p)$-approximations for A, any finite set of such approximations is decidable. In that sense, the behavior of μ_ζ, and thus of $\mathrm{ic}^{\mathcal{O}(p)}$ relative to A, can be approximated by the minimization functions of parameterizations in $\mathcal{L}_{\mathbf{FPT}}$. As the following theorem shows, this approximation is in essence an approximation from above.

3.3.5. Theorem. *For any set A that is in* **FPT** *with a parameterization η there is a polynomial p such that, as a function of x, we have*

$$\mathrm{ic}^{\mathcal{O}(p)}(x : A) \in \mathcal{O}(\mu_\eta(x)).$$

Proof:
It suffices to show that for some polynomial p and every parameter value k there is an $\mathcal{O}(p)$-approximation ϕ for A that satisfies

- $\operatorname{dom}(\phi) = \eta_k$, and
- $|\phi| \in \mathcal{O}(|k|)$, with the hidden constant depending only on A and η.

Let ψ be a direct parameterized procedure and p a polynomial witnessing that A is in **FPT** with η. By definition, for every k, the partial application of ψ to k yields an $\mathcal{O}(p)$-approximation for A with domain η_k. We claim that this $\mathcal{O}(p)$-approximation satisfies the requirements on ϕ listed above. The partial application can be computed from k and ψ. Thus, the length of the approximation can be kept in $\mathcal{O}(|\langle k, \psi\rangle|)$. Because ψ is fixed for all instances and is determined only by A and η, the theorem follows. □

This theorem has an interpretation in terms of optimal parameterizations. For a polynomial p and any set A, the parameterization ζ defined by (3.4) behaves like a principal parameterization with respect to $\boldsymbol{X}\mathbf{TIME}(p)$. However, as we observed already, this ζ need not be a member of $\mathscr{L}_{\mathbf{FPT}}$, let alone $\mathscr{L}_{\boldsymbol{X}\mathbf{TIME}(p)}$. In this interpretation, any parameterization η with which A is in **FPT** represents a measure of complexity. In particular, the function μ_η approximates from above the behavior of the $\mathcal{O}(p)$-bounded instance complexity relative to A. However, because η is necessarily decidable, μ_η is computable. In that sense, the minimization function serves as a form of uniform instance complexity.

Central to the study of instance complexity is the *instance complexity conjecture* [119]. This conjecture posits that infinitely many instances of a decision problem have no redundancy in their description that could aid in deciding on membership. The conjecture formalizes the idea that for infinitely many instances, a lookup in the style of Example 3.3.2 is essentially the best that can be done. In the example, we constructed a procedure that compares its input to a hardcoded string x. The bound on the instance complexity that this construction resulted in can be improved slightly. This is done by not hardcoding x, but instead including the specification of a procedure for reproducing x. The procedure should of course be able to produce x within the time bound we place on the instance complexity. In this way, it can be shown that the bounded instance complexity is bounded from above by the similarly bounded Kolmogorov complexity [119, 100]. For technical reasons, there is some slack required in the time bound. Let us make this known bound precise, denoting the t-bounded Kolmogorov complexity of a string x by $\mathrm{K}^t(x)$: There is a constant c such that, with $t'(n) = c \cdot t(n) \cdot \log t(n) + c$, for every set A and string x, we have $\mathrm{ic}^{t'}(x : A) \leq \mathrm{K}^t(x) + c$. The instance complexity conjecture states that this bound is tight infinitely often if there is no decision procedure for A with a running time bounded by t. Specifically, it states that for such a set A there is a constant c and infinitely many strings x such that we have $\mathrm{ic}^t(x : A) \geq \mathrm{K}^{t'}(x) - c$.

The instance complexity conjecture has been resolved for a variety of time bounds [59, 29]. For the unbounded version, which targets the semidecidable sets, the conjecture was proven wrong by Kummer [98]. We shall show that a time-unbounded but uniform statement similar in spirit to the instance complexity conjecture is true. For this uniform statement, we return to our earlier interpretation of minimization functions. We think of the minimization function with respect to a decidable parameterization as a form of uniform instance complexity. Note that even parameterizations that are equivalent according to the uniform order on parameterizations may give rise to different minimization functions. Therefore, it makes little sense to directly compare the minimization function to Kolmogorov complexity. Instead, we look at the behavior of the minimization function "in the limit". For this, we use an auxiliary function with an unbounded limit inferior, as defined on page 48.

3.3.6. Theorem. *Let η be a decidable parameterization that does not include 2^+, and let f be any computable function with an unbounded limit inferior. There are infinitely many instances x for which we have*

$$\mathrm{K}(x) \leq f(\mu_\eta(x)).$$

Proof:
The following pseudocode defines, uniformly in m, a procedure ϕ_m.

```
1: For each instance $x$ in $\{0, 1, 00, 01, ...\}$:
    1.1: If $f(\mu_\eta(x)) \geq m$, return $x$.
```

Because η does not include 2^+, the minimization function μ_η takes on arbitrarily large values. Additionally, as f has an unbounded limit inferior, so does the composite function of f after μ_η. This means that our procedure ϕ_m terminates, regardless of the value of m. Moreover, since η is decidable, μ_η is computable, so ϕ_m as a whole is, indeed, computable.

Observe that the set $S = \{x \mid \exists m \in \mathbb{N} \colon \phi_m \text{ returns } x\}$ is infinite. We claim that all but finitely many elements x of this set satisfy $\mathrm{K}(x) \leq f(\mu_\eta(x))$, thus proving the theorem. Our pseudocode was brief, and surely for some constant c we find that, for all m, we can realize $|\phi_m| \leq c \cdot |\mathrm{asStr}(m)|$. Therefore, if some ϕ_m returns x, we have

$$\mathrm{K}(x) \leq |\phi_m| \leq c \cdot |\mathrm{asStr}(m)|.$$

On the other hand, by construction we have $m \leq f(\mu_\eta(x))$. As $|\mathrm{asStr}(m)|$ is within one bit of $\log m$, we thus find that we have, for all x in S,

$$\mathrm{K}(x) \leq c \cdot \log f(\mu_\eta(x)).$$

Consequently, for all but finitely many x in S, we have $\mathrm{K}(x) \leq f(\mu_\eta(x))$. □

Note that given some threshold, there are only finitely many objects with a Kolmogorov complexity below that threshold. For this reason, the Kolmogorov complexity is unbounded on every infinite slice of a parameterization. Yet, we obtain from Theorem 3.3.6 that no nontrivial parameterization contains all instances of a bounded Kolmogorov complexity in its slices.

3.3.7. Corollary. *For any parameterization $\eta \in \mathcal{L}_{\mathbf{FPT}}$ that does not include 2^+ there are infinitely many strings x and k such that we have*

- $\mathrm{K}(x) \leq |k|$, *yet also*
- $x \notin \eta_k$.

In particular, the parameterization given by

$$(\{x \mid \mathrm{K}(x) \leq |k|\})_{k \in 2^+} \tag{3.5}$$

is not a member of $\mathcal{L}_{\mathbf{FPT}}$. This illustrates the remark in the proof of Theorem 3.2.26 that not every parameterization of which all slices are finite is a greatest element of $\mathcal{L}_{\mathbf{FPT}}$. More generally it shows that the Kolmogorov complexity cannot be obtained as the minimization function applied to some decidable parameterization. Of course, this is already immediate from the fact that Kolmogorov complexity is not computable. Nevertheless, the fact that the parameterization given by (3.5) is not a member of $\mathcal{L}_{\mathbf{FPT}}$ has a specific interpretation in our parameterized complexity theory. It conveys the same sentiment as the instance complexity conjecture, but in a more uniform setting. Conceptually, we find that, given a parameterization η in $\mathcal{L}_{\mathbf{FPT}}$, there is always a structural property of data that is not taken into account by η.

3.3.2 Equivalent Filters

Given a set A, every parameterization η in $\mathcal{F}_{\mathbf{FPT}}(A)$ gives a gradation of the computational complexity of the instances of A. The behavior of the complexity measure embodied by η and its minimization function, μ_η, can be somewhat intangible. When η is not a principal parameterization for A, there are, in a sense, better parameterizations for A. Correspondingly, the complexity measure associated with a parameterization that is not principal can be improved upon. As seen in Section 3.2.3 and Section 3.2.4, however, many sets do not allow for principal parameterizations at all. Furthermore, principal parameterizations are not unique as it is actually the equivalence class to which a parameterization belongs that is principal. Nevertheless, a sense of optimality is still reserved for the complexity *behavior* conveyed by principal parameterizations.

In the absence of principal parameterizations, we can instead look at the collection of all parameterizations that put a set A in a class like **FPT** or $\boldsymbol{X}$**P**. This collection, the filter with respect to the class of interest, embodies all possible gradations of computational complexity of the instances of A. Thus, it can function as a proxy for the distribution of complexity inside A. Anything that is true of all parameterizations in the filter is true of any individual parameterization in the filter. In other words, the filter is a profile of the computational complexity at the level of instances of A.

Looking at the entire filter makes a study of the distribution of complexity inside a set possible also when the filter is not principal. The approach is a continuation of an idea by Orponen [116], who represented the complexity characteristics of a set by the filter of its complexity cores. Orponen showed that this idea is fruitless when applied to *proper* polytime-cores, the complexity cores consisting only of members of the set under investigation. In comparison, our parameterized setting is promising. As profiles of the computational complexity of a set, filters convey information about a set. We shall try to characterize what aspects of a set are determined by the filter of parameterizations that make the set fixed-parameter tractable.

Nonuniform Filters and the Berman–Hartmanis Conjecture

With fixed-parameter tractability, we look at the parameter dependence of the running time of decision procedures. As every instance is associated with its own set of parameter values, our parameterized notion of complexity is thus one at the level of individual instances. Note that we do not look at the output of a decision procedure. In fact, we may allow for a second procedure that runs in polynomial time and decides whether or not the output of the decision procedure should be inverted. Our notion of complexity is insensitive to such polynomial-time postprocessing. This behavior is to be expected of a notion of computational complexity at the level of instances. More precisely, we perceive the distribution of complexity in a set A to be the same as that of the symmetric difference of A with any set in **P**. Recall that the symmetric difference of a set A and a set X is defined as $A \triangle X = (A \setminus X) \cup (X \setminus A)$. If, for a set A, we use the filter $\mathscr{F}_{\mathbf{FPT_{nu}}}(A)$ as a complexity profile, the insensitivity to polynomial-time postprocessing is formalized as follows.

3.3.8. Theorem. *For any set X in* **P** *and any set A we have*

$$\mathscr{F}_{\mathbf{FPT_{nu}}}(A) = \mathscr{F}_{\mathbf{FPT_{nu}}}(A \triangle X).$$

Proof:
Given X, any polytime-segment of A can be turned into a polytime-segment of $A \triangle X$ and vice versa. Let η be a parameterization with which A is in $\mathbf{FPT_{nu}}$

and let c be the degree in the running time of a polynomial-time decision procedure for X. The polytime-segments of A that are present in η have associated to them polynomials of a bounded degree. Let d be this degree. The degree of the constructed polytime-segments of $A \triangle X$ can be kept below $\max(c, d)$, hence $A \triangle X$ is also in $\mathbf{FPT_{nu}}$ with η. □

Intuitively, the above theorem states that taking the symmetric difference with a simple set does not alter the distribution of complexity in a set. If we want to be more precise, we should say that it is the complexity profile that stays the same. However, this means that no parameterization, representing a distribution of complexity, has been gained or lost by taking the symmetric difference. In this sense, the more intuitive interpretation of the theorem is correct.

Similarly, we find that the symmetric difference of two sets with the same complexity profile is no more complex than either of the initial sets: The filter corresponding to the symmetric difference includes that of the original sets. This signifies that instances only get easier, if their complexity changes at all.

3.3.9. Theorem. *For any two sets A, B satisfying $\mathcal{F}_{\mathbf{FPT_{nu}}}(A) = \mathcal{F}_{\mathbf{FPT_{nu}}}(B)$ we have*

$$\mathcal{F}_{\mathbf{FPT_{nu}}}(A) \subseteq \mathcal{F}_{\mathbf{FPT_{nu}}}(A \triangle B).$$

Proof:
Like in the proof of the previous theorem, we shall prove this theorem by combining polytime-approximations. For the current theorem, let ϕ and ψ be polytime-approximations for A and B respectively. It suffices to show that if their domains match, ϕ and ψ can be combined into a polytime-approximation for $A \triangle B$ with the same domain. The degree of the running time of this combined approximation should not be larger than the maximum of the degrees of the running times of ϕ and ψ. A procedure that, on input x, computes $\phi(x)$ and $\psi(x)$, and subsequently returns the exclusive disjunction of their outputs meets these requirements. From this, it follows that every parameterization that is in $\mathcal{F}_{\mathbf{FPT_{nu}}}(A)$ is also in $\mathcal{F}_{\mathbf{FPT_{nu}}}(A \triangle B)$. □

The above theorem does not guarantee that the symmetric difference of two sets that share all their parameterizations is in **P**. If this would be the case, a filter with respect to $\mathbf{FPT_{nu}}$ would uniquely define a set up to variations in **P**. This would lead to a nice alternative characterization of the notion of complexity embodied by filters with respect to $\mathbf{FPT_{nu}}$. Namely, we would have that all sets that share a complexity profile are related by a symmetric difference in **P**. Of comparable flavor is the *Berman–Hartmanis conjecture* [19]. This conjecture states that all **NP**-complete sets are related by polytime-computable isomorphisms. In other words, it states that completeness for **NP** uniquely defines a set up to isomorphisms computable in polynomial time.

Let us digress from our study of what characterizes sets that share a filter with respect to $\mathbf{FPT_{nu}}$, and look at the effect of isomorphisms on such filters. The existence of a polytime-computable isomorphism between two sets does not indicate that the sets have *the same* distribution of complexity. Instead, it indicates that distributions of complexity in the two sets are *comparable.* The observation that many of the known **NP**-complete sets are related by polytime-computable isomorphisms makes this viewpoint valuable [19, 69]. However appealing, the conjecture is generally believed to be untrue. This is because it is widely believed that *one-way functions*, polytime-computable functions with an inverse that is not computable in polynomial time, exist. Using particularly strong functions of this kind, it is possible to construct **NP**-complete sets between which no polytime-computable isomorphism exists [156, 99, 76, 5].

Nonetheless, we note that isomorphic sets have isomorphic filters with respect to $\mathbf{FPT_{nu}}$. To see why, first note that a permutation f of 2^+, a bijection from 2^+ to itself, can be lifted to a function on parameterizations via

$$f(\eta) = (\{f(x) \mid x \in \eta_k\})_{k \in 2^+}.$$

Defined this way, the function preserves minimization in the sense that, for all permutations f, parameterizations η, and instances x, we have

$$\mu_\eta(x) = \mu_{f(\eta)}(f(x)).$$

As a consequence, the nonuniform order on parameterizations is preserved as well. If the permutation is polytime-computable in both directions, it can be lifted even further, into an isomorphism of filters with respect to $\mathbf{FPT_{nu}}$. Given a set A, a permutation f that is polytime-computable in both directions is also a reduction from A to the set $B = \{f(x) \mid x \in A\}$. In fact, this f is a polytime-computable isomorphism between A and B. The filter that is isomorphic to $\mathcal{F}_{\mathbf{FPT_{nu}}}(A)$ via f is $\mathcal{F}_{\mathbf{FPT_{nu}}}(B)$.

Something interesting happens when, given a parameterization η and a permutation f of 2^+, the parameterization $f(\eta)$ is in the same equivalence class as η. When this is the case, f is an automorphism of the equivalence class of parameterizations to which η belongs. We could say that this equivalence class is closed under applications of f. Lifting f to the level of an isomorphism of filters, the equivalence class to which η belongs is a fixed point of f. Examples of such fixed points are the least and greatest element of $\mathcal{L}_{\mathbf{FPT_{nu}}}$. Any permutation of 2^+ is an automorphism on both these equivalence classes of parameterizations. Parameterizations in the least element include 2^+ as one of their slices. Since the minimization with respect to such a parameterization is bounded, these parameterizations are equivalent to those they are mapped to. Parameterizations in the greatest element of $\mathcal{L}_{\mathbf{FPT_{nu}}}$ have only finite slices. This property too is preserved by the mappings of parameterizations that stem from permutations of 2^+.

3.3.10. Example. A different example of an automorphism of an equivalence class of parameterizations is available with respect to the uniform order on parameterizations. Let η be the parameterization defined by (3.5). This parameterization is not decidable because Kolmogorov complexity is uncomputable. This makes that the equivalence class of the parameterization is not a greatest element in, say, $\mathcal{L}_{\mathbf{XP}}$ or $\mathcal{L}_{\mathbf{FPT}}$. Yet, we can show that, for any computable permutation f of 2^+, the parameterization $f(\eta)$ is in the same equivalence class as η. To wit, for every such f and all x, we have that $\mathrm{K}(f(x))$ is within an additive constant of $\mathrm{K}(x)$. This additive constant depends only on f. Therefore, $\mu_{f(\eta)}(x)$ and $\mu_{\eta}(x)$ are within an additive constant of each other and both $\mathrm{gap}_{f(\eta),\eta}$ and $\mathrm{gap}_{\eta,f(\eta)}$ are bounded.

Suppose the equivalence class of a parameterization η is closed under all permutations that are polytime-computable in both directions. The Berman–Hartmanis conjecture implies that if η puts any set that is complete for $\mathbf{NP}$ in $\mathbf{FPT}_{\mathbf{nu}}$, then it puts all such sets in $\mathbf{FPT}_{\mathbf{nu}}$. This parameterization η would be in the filter of parameterizations that put all $\mathbf{NP}$-complete sets in $\mathbf{FPT}_{\mathbf{nu}}$,

$$\bigcap_{S,\ \mathbf{NP}\text{-complete}} \mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(S).$$

Note that a parameterization that puts all $\mathbf{NP}$-complete sets in $\mathbf{FPT}_{\mathbf{nu}}$ also puts all $\mathbf{co}$-$\mathbf{NP}$-complete sets in $\mathbf{FPT}_{\mathbf{nu}}$. This is because $\mathbf{FPT}_{\mathbf{nu}}$ treats members and nonmembers of sets the same. Further, note that for any class of sets, finite or infinite, a similar intersection of filters yields the filter of parameterizations that put all sets in $\mathbf{FPT}_{\mathbf{nu}}$. That the intersections are indeed filters and cannot be empty is a consequence of the fact that $\mathcal{L}_{\mathbf{FPT}_{\mathbf{nu}}}$ is bounded. Every intersection of filters contains at least the greatest element of the lattice, so it is nonempty. The resulting filter may or may not be of the form $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$ for some set A.

We shall now return our attention to the characterization of sets that share a filter with respect to $\mathbf{FPT}_{\mathbf{nu}}$. Specifically, we shall look at the connections between filters that, for some set A, arise as filters of the form $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$. In doing so, we have abstracted from the complexity of instances fourfold. Our first level of abstraction is that of polytime-segments. Combined as slices, they constitute our second level, that of parameterizations. The structure of parameterizations with respect to parameterized complexity classes is that of a filter, which is our third level of abstraction. Theorem 3.3.8 and Theorem 3.3.9 are suggestive of an algebraic structure of filters.

Suppose the symmetric difference of any two sets of which the filters are the same would end up in $\mathbf{P}$. In that case, the symmetric difference would induce a commutative group structure on the filters of the form $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$, where A is a set. The elements of this group, the filters, are related by the group operation that maps filters $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A)$ and $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(B)$ to the filter $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(A \triangle B)$. In this group, the filter of any set in $\mathbf{P}$, which we could represent by $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}(\emptyset)$, serves as an identity element. Note that the sets A for which $\mathcal{F}_{\mathbf{FPT_{nu}}}(A)$ equals $\mathcal{F}_{\mathbf{FPT_{nu}}}(\emptyset)$ are precisely the sets in $\mathbf{P}$. More specifically, a set is in $\mathbf{P}$ precisely when its filter with regard to $\mathbf{FPT_{nu}}$ equals $\mathcal{L}_{\mathbf{FPT_{nu}}}$. Choosing $\emptyset$ as a representative set in $\mathbf{P}$ fits well with the fact that every element of the group is its own inverse. Indeed, we have $\mathcal{F}_{\mathbf{FPT_{nu}}}(A \bigtriangleup A) = \mathcal{F}_{\mathbf{FPT_{nu}}}(\emptyset)$.

We have seen in Theorem 3.3.8 that taking the symmetric difference with a set in $\mathbf{P}$ does not alter the distribution of complexity in a set. Ideally, the converse would be true as well and all sets that share a filter would be related by a symmetric difference in $\mathbf{P}$. Theorem 3.3.9 is not strong enough to conclude this alternative characterization of the notion of complexity embodied by filters with respect to $\mathbf{FPT_{nu}}$. In the nonuniform case, the algebraic structure of filters just described is not present, as sets with distinct computational complexity may share a filter.

3.3.11. Theorem. *There exist sets A and B satisfying*

- $A \bigtriangleup B \notin \mathbf{P}$, *and*
- $\mathcal{F}_{\mathbf{FPT_{nu}}}(A) = \mathcal{F}_{\mathbf{FPT_{nu}}}(B)$.

Proof:
Recall that the filter with respect to $\mathbf{FPT_{nu}}$ induced by a $\mathbf{P}$-bi-immune set consists only of the class of parameterizations of which all slices are finite. Conversely, if this is the filter induced by some set, then that set is $\mathbf{P}$-bi-immune. To prove the theorem it thus suffices to construct two $\mathbf{P}$-bi-immune sets with a symmetric difference outside $\mathbf{P}$.

A decidable $\mathbf{P}$-bi-immune set can be constructed using the finite extension method [46]. In this method, a set is constructed in such a way that it satisfies infinitely many requirements. Specifically, of every polytime-approximation it is required that either it is not an approximation for the set being built, or that its domain is finite [17]. By augmenting these requirements, we can ensure that the symmetric difference with some known decidable set is not in $\mathbf{P}$. We need to ensure that there is no set in $\mathbf{P}$ that equals the symmetric difference of the set we are constructing and the known decidable set.

Instead of spelling out all technicalities, we refer to a stronger result by Geske, Huynh, and Seiferas [64]. Using a similar approach, they derive that for every constant $c > 0$ there is a $\mathbf{TIME}(2^{cn})$-bi-immune set that is decidable in linear exponential time [see also 104]. Let A be such a set for $c = 1$ and let c_A be so that A is in $\mathbf{TIME}(2^{c_A n})$. Likewise, let B be such a set for $c = c_A + 1$. If $A \bigtriangleup B$ would be in $\mathbf{P}$, then $A \bigtriangleup (A \bigtriangleup B) = B$ would be in $\mathbf{TIME}(2^{(c_A+1)n})$. This contradicts the fact that B was constructed to be $\mathbf{TIME}(2^{(c_A+1)n})$-bi-immune, hence $A \bigtriangleup B$ cannot be in $\mathbf{P}$. $\square$

While filters with respect to $\mathbf{FPT_{nu}}$ are, by Theorem 3.3.8, indeed insensitive to polynomial-time postprocessing, they are insensitive to more than just that. The previous theorem showed that there are sets with intuitively different distributions of computational complexity that these filters cannot tell apart. This can be taken as an argument against the nonuniform notion of fixed-parameter tractability. By extension, it would then also be an argument against instance complexity, because of Theorem 3.3.3 and Theorem 3.3.4.

Uniform Filters and a Separation Conjecture

From a topology perspective, Theorem 3.3.11 represents the failure, in the setting of nonuniform fixed-parameter tractability, of a tantalizing separation axiom. We could say that a set in **P** *separates* two sets if it contains a polytime-core for precisely one of the two. Unfortunately, Theorem 3.3.11 holds that not every two sets of which the symmetric difference is outside **P** have such a separating set in **P**. A more intricate separation axiom may be available when uniformity constraints are added. We conjecture that, in the uniform case, the filter of the symmetric difference of two sets that share a filter collapses to that of a set in **P**.

3.3.12. CONJECTURE. *For any two sets A, B we have*

$$\mathscr{F}_{\mathbf{FPT}}(A) = \mathscr{F}_{\mathbf{FPT}}(B) \iff \mathscr{F}_{\mathbf{FPT}}(A \triangle B) = \mathscr{F}_{\mathbf{FPT}}(\emptyset).$$

In other words, we conjecture that when the filters of two sets A and B are equal, we have $A \triangle B \in \mathbf{P}$. Equivalently, we conjecture that, if both filters are the same, there is a set X in **P** such that we have $B = A \triangle X$. At the very least, we find that the converse is true since Theorem 3.3.8 has a uniform counterpart.

3.3.13. THEOREM. *For any set X in* **P** *and any set A we have*

$$\mathscr{F}_{\mathbf{FPT}}(A) = \mathscr{F}_{\mathbf{FPT}}(A \triangle X).$$

Proof:
We claim that any parameterization η with which some set A is in **FPT** also puts $A \triangle X$ in **FPT**. Because we have $(A \triangle X) \triangle X = A$, the theorem follows from this claim.

Let ϕ be a parameterized procedure witnessing that A is in **FPT** with η, and let ψ be a polynomial-time decision procedure for X. These procedures can be combined into a parameterized procedure that witnesses that $A \triangle X$ is in **FPT** with η as follows. First, given an instance x and parameter value k, the parameterized procedure simulates ϕ to completion on input (x, k). If $\phi(x, k)$ yielded ?, our procedure does so as well. Otherwise, it also computes $\psi(x)$ and outputs the exclusive disjunction of $\phi(x, k)$ and $\psi(x)$. The parameterized procedure thus defined meets the running-time requirements of the definition of **FPT** and converges to $A \triangle X$. Finally, the corresponding parameterization is η, as desired. □

Put yet another way, Conjecture 3.3.12 asserts that $\mathscr{F}_{\mathbf{FPT}}$ is a group homomorphism that maps $\mathbf{P}$ to the identity element. That is, $\mathbf{P}$ is the *kernel* of the group homomorphism $\mathscr{F}_{\mathbf{FPT}}$. Intuitively, the conjecture suggests that sets that map to the same filter with regard to **FPT** *have the same distribution of complexity.* More precisely, such sets are fixed-parameter tractable with exactly the same parameterizations, which represent distributions of complexity. If true, a filter in the range of $\mathscr{F}_{\mathbf{FPT}}$ would determine the input set up to a symmetric difference in $\mathbf{P}$. Thus, our conjecture is similar to the Berman–Hartmanis conjecture, but it targets postprocessing instead of isomorphism [see also 5]. More informally, our conjecture therefore reads as follows.

3.3.14. SLOGAN. *A set is determined up to polynomial-time postprocessing by the parameterizations that make it fixed-parameter tractable.*

Note that, unlike the other slogans in this thesis, this slogan does not correspond to a theorem but to a conjecture.

In support of our conjecture, we have a uniform counterpart to Theorem 3.3.9 too. While it does not show that the symmetric difference of two sets with the same filter is in $\mathbf{P}$, it does show that it is no more complex than either of the sets.

3.3.15. THEOREM. *For any two sets A, B satisfying $\mathscr{F}_{\mathbf{FPT}}(A) = \mathscr{F}_{\mathbf{FPT}}(B)$ we have*

$$\mathscr{F}_{\mathbf{FPT}}(A) \subseteq \mathscr{F}_{\mathbf{FPT}}(A \triangle B).$$

Proof:
The proof of Theorem 3.3.9 can easily be adapted for the uniform setting. The polytime-approximations at play in that proof can be obtained uniformly in the parameter. Rather than combining these approximations, we combine the parameterized procedures that produce them at once. Let ϕ and ψ be parameterized procedures putting A and B, respectively, in **FPT** with some shared parameterization. These procedures can be combined into a parameterized procedure that puts $A \triangle B$ in **FPT** with the same parameterization. In essence, the combined parameterized procedure returns, on input (x, k), the exclusive disjunction of $\phi(x, k)$ and $\psi(x, k)$. □

It is worth noting that uniformity constraints preclude a proof such as that of Theorem 3.3.11 for filters with respect to **FPT**. We are confronted with the fact that not every two parameterizations of which all slices are finite reside in the same uniform equivalence class. This aspect of filters with respect to **FPT** was encountered before in the proof of Theorem 3.2.26. On top of that, as observed below Corollary 3.3.7, a parameterization with finite slices need not even be a member of $\mathscr{L}_{\mathbf{FPT}}$. Equivalence of uniform filters can thus be seen as a refinement of equivalence of nonuniform filters. Interestingly, the usefulness of such a refinement was already hinted at by Orponen [116] in 1986.

3.3.3 Randomness and Hardness

The instance complexity conjecture brings together algorithmic complexity and computational complexity. This connection is interesting, because the two notions of complexity appear to be based on unrelated quantities. Algorithmic complexity is concerned with the lengths of specifications of procedures, whereas computational complexity is concerned with their running times. Instance complexity considers all decision procedures with some restricted running time for a given set at once. If the set adheres to the instance complexity conjecture, there are infinitely many instances on which no decision procedure can do better than a table lookup. To be fair, for each of these instances the lookup table we are comparing to would consist of a single, maximally compressed entry. The message of the conjecture is therefore a highly nonuniform one. In addition to that, the conjecture tells us very little about the nature of these infinitely many special instances.

The special instances on which a given decision problem is hard may be highly structured. For instance, VertexCover remains **NP**-complete when restricted to graphs where every vertex is connected to precisely three others [63]. At the same time, it is not unreasonable to expect that easy instances of an intractable decision problem necessarily have some structure. Some properties of an instance must be responsible for the fact that a reduced computation time suffices to arrive at a decision about membership. These properties correspond to recognizable redundancy in the encoding of instances. By means of lookup tables, however, a decision procedure can be made to run fast on any finite selection of instances. This hinders the application of our intuition to the instance complexity conjecture, as it looks at the collective of decision procedures for some set. Instead, we are interested in properties shared by each of the decision procedures individually.

High Complexity

Using parameterizations, we can analyze to what extent high algorithmic complexity implies high computational complexity. For algorithmic complexity, characterizing high complexity is routine [100, Theorem 3.3.1].

3.3.16. Definition. A set of strings X is *random* if there is a constant r such that, for all n and all $x \in X$ of length n, we have

$$\mathrm{K}(x) \geq n + \mathrm{K}(n) - r.$$

Note that the $\mathrm{K}(n)$ term is required because we are working with the prefix-free variant of Kolmogorov complexity. Other choices, ranging from a constant to $\log n$, are available, but for our purposes this definition, using $\mathrm{K}(n)$, works best. Furthermore, although Kolmogorov complexity is only defined up to an additive constant, our definition of a random set of strings is unambiguous. What sets count as random sets remains the same, even if we switch to a different encoding of procedures. Of course, the constant r that witnesses the randomness of a random set does depend on the encoding of procedures that is used.

Perhaps surprisingly, according to the above definition any finite set is random. Fortunately, there exist infinite random sets as well as infinite nonrandom sets. As a result, the interesting random sets are the infinite ones.

For computational complexity, a characterization of sets of instances that are of high complexity may not be immediately obvious. Identifying minimal computational complexity with polynomial-time decidability, complexity can be measured from parameterizations that put a set in $\boldsymbol{XP}$ or **FPT**. To do so, suppose we have a direct parameterized procedure ϕ that witnesses that some given set A is in one of these classes. Let η be the parameterization corresponding to ϕ and let f be the computable function upper bounding the parameter dependence of the running time of ϕ. We assume that whenever two parameter values k and k' are so that we have $\eta_k \subseteq \eta_{k'}$, we also have $f(k) \leq f(k')$. Intuitively, this expresses that deciding membership of fewer instances cannot be harder. Thus, instances of high computational complexity are those that only occur in slices that are high up in the inclusion order of a parameterization. For most natural encodings of parameters, this turns the minimization function of a parameterization into a measure of computational complexity of instances. We say that the encoding of a parameter is *compatible* with the inclusion order of a parameterization η if η satisfies

$$\forall k, k' : \eta_k \subseteq \eta_{k'} \implies |k| \leq |k'|. \tag{3.6}$$

Note that any parameterization has a subparameterization on which the encoding of the parameter is compatible with the inclusion order. This observation is similar to Lemma 3.1.19. Most parameterizations in the literature [e.g. 44, 57, 113, 37] meet the compatibility criterion in (3.6). Hence, for most common parameterization η that put some given set in **FPT**, a measure of the computational complexity of instances is provided by μ_η.

We have identified sets of high algorithmic complexity as those where the Kolmogorov complexity of members is within a constant of the highest value possible. Likewise, we wish to identify sets of high computational complexity as those where the members have a near-maximal computational complexity. Intractable instances attain high values under the minimization function with respect to a parameterization, thus the following notation is of use.

3.3.17. Definition. Given a parameterization η, we use $\mathrm{N}_\eta(n, k)$ for the number of elements in the set $2^n \cap \eta_k$ and further define

$$\mathrm{M}_\eta(n) = \max\{\mu_\eta(x) \mid x \in 2^n\}.$$

Observe that when a parameterization η is decidable, the functions N_η and M_η are computable. By our findings in Section 3.2.4, we should not expect optimal parameterizations with respect to **FPT** to exist for a given set. Therefore, a parameterized notion of hardness must be dependent on the choice of a parameterization. Hard instances are those where the running time of a direct parameterized procedure must contain a large parameter dependent factor. By our previous discussion, we expect that in this case the minimization function attains a high value. Note that the computable function upper bounding the parameter dependence of the running time is only an upper bound. Thus, strictly speaking, a high value of the minimization function does not express that an instance is computationally complex. It only expresses that we are unable to rule out that it is not. Nevertheless, we accept the minimization function as a measure of the computational complexity of instances.

3.3.18. Definition. With respect to a parameterization η, a set of strings X is *η-hard* if there is a constant h such that, for all n and all $x \in X$ of length n, we have

$$\mu_\eta(x) \geq \mathrm{M}_\eta(n) - h.$$

As with random sets, only infinite η-hard sets are of interest.

The connection between high algorithmic complexity, randomness, and high computational complexity, hardness, goes beyond the similarity of their definitions. Using an incompressibility argument, we can prove that randomness implies hardness with respect to certain parameterizations. The parameterizations for which we can do so are those that hold information about almost all strings. This requirement translates into an informativeness criterion which expresses that the slices of a parameterization can be used for compression. Compressing instances will be done in the proof of Theorem 3.3.20 by means of a specific encoding scheme that makes use of parameterizations. We have chosen the following definition in line with this encoding scheme. As a result, the following definition is visibly influenced by the details of the encoding scheme.

3.3.19. Definition. A decidable parameterization η is *informative* if there is a constant c such that for every n and k that satisfy $\mathrm{M}_\eta(n) - |k| \geq c$ we have

$$\mathrm{N}_\eta(n, k) \leq 2^{n+|k|+c-2\cdot\mathrm{M}_\eta(n)}.$$

Informally, this rather ad hoc informativeness criterion holds that the density of any fixed slice of an informative parameterization gets lower as n increases. To see why, observe that in the exponent on the right side of the above inequality the term $|k| + c$ must be less than or equal to $\mathrm{M}_\eta(n)$. Therefore, $\mathrm{N}_\eta(n, k)$ must also be bounded from above by $2^{n-\mathrm{M}_\eta(n)}$. If we assume that as n goes to infinity, so does $\mathrm{M}_\eta(n)$, then we thus have $\mathrm{N}_\eta(n, k) \in \mathrm{o}(2^n)$. In the words of Li and Vitányi [100, Example 2.2.6, attributed to Michael Sipser], this means that the slices of an informative parameterization are *meager*. Additionally, in an informative parameterization, the speed with which the density of a slice must decrease depends on the corresponding parameter value. For shorter parameter values, the density must decrease faster. Indeed, the difference between $|k| + c$ and $M_\eta(n)$ gets bigger as $|k|$ gets smaller.

Shortly, we shall see that informativeness of a parameterization is implied by a more practically useful property. For now, we are set to connect high algorithmic complexity to high computational complexity.

3.3.20. Theorem. *For any informative parameterization η, every random set is η-hard.*

In less technical terms, this amounts to the following slogan. This slogan assumes that we have agreed on a constant r such that an instance x of length n is random if it satisfies $K(x) \geq n + K(n) - r$. Only after agreeing on such a constant r is it meaningful to talk about random instances instead of about random sets. The above theorem tells us that the set of all random instances is a hard set in accordance with Definition 3.3.18.

3.3.21. Slogan. *Random instances are hard.*

Proof:
Let η be a decidable parameterization, x an arbitrary string of length n, and k a parameter value of length $\mu_\eta(x)$ such that we have $x \in \eta_k$. Denote by $K(\eta)$ the minimum length of a decision procedure for η. We shall consider a four-part encoding of x based on η. The first two parts of this encoding specify η and n, the last two specify k and the rank of x in $2^n \cap \eta_k$, respectively. Because η is decidable, the last two parts can be specified in $M_\eta(n) + 1$ and $\log N_\eta(n, k)$ bits respectively. By always using this many bits, we can concatenate the two specifications without the need for a separation marker to distinguish the two parts.

A few things are worth noting about this encoding scheme. The number of parameter values to distinguish is $2^{M_\eta(n)+1} - 1$, which is why our fixed-length encoding of parameter values uses $M_\eta(n) + 1$ bits. Additionally, we make use of the fact that, given η and n, we can compute $M_\eta(n)$. In turn, we can compute $N_\eta(n, k)$ once k is known too. Thus, we find

$$K(x) \leq K(\eta) + K(n) + M_\eta(n) + 1 + \log N_\eta(n, k).$$

Now, let X be a random set, the randomness of which is witnessed by a constant r, and let x be a string in X of length n. We can combine the defining equation for random sets, $n + K(n) - r \leq K(x)$, with the upper bound on $K(x)$ obtained above to get

$$n + K(n) - r \leq K(\eta) + K(n) + M_\eta(n) + 1 + \log N_\eta(n, k).$$

After some rearranging and flipping the inequality, we obtain

$$\log N_\eta(n, k) \geq n - M_\eta(n) - (K(\eta) + r + 1).$$

Lastly, suppose η is an informative parameterization and a constant c witnesses its informativeness. We may assume that we have $\mathrm{M}_\eta(n) - \mu_\eta(x) \geq c$, for if not, η-hardness of X is immediate. Thus, we may use the upper bound on $\mathrm{N}_\eta(n,k)$ provided by the informativeness criterion and get

$$n + |k| + c - 2 \cdot \mathrm{M}_\eta(n) \geq n - \mathrm{M}_\eta(n) - (\mathrm{K}(\eta) + r + 1),$$

which simplifies to

$$|k| \geq \mathrm{M}_\eta(n) - (\mathrm{K}(\eta) + r + c + 1).$$

In this last inequality, $\mathrm{K}(\eta)$, r, and c are independent of x. Because we have chosen k to be so that we have $|k| = \mu_\eta(x)$, this proves η-hardness of X. □

Of course, Theorem 3.3.20 was made possible by the way we defined being informative in Definition 3.3.19. The informativeness criterion may be somewhat elusive and it may not be straightforward to test whether a given parameterization is informative or not. In fact, at this point we have little reason to believe informative parameterizations exist at all. It is therefore useful to identify a class of parameterizations of which informativeness can be established easily. Central to this class will be a density requirement on the slices of parameterizations.

3.3.22. Definition. A parameterization η has *uniform exponential density* if there is a non-decreasing function f, and there are two constants ε and γ, both strictly between 0 and 1, such that, for all n and all k with $|k| \leq \mathrm{M}_\eta(n)$, we have

$$2^{\varepsilon f(|k|) n^\gamma} \leq \mathrm{N}_\eta(n,k) \leq 2^{f(|k|) n^\gamma}.$$

Parameterizations of which the slices have exponential density have, from a parameterized tractability point of view, a nice robustness property. The slices of a parameterization are polytime-segments of every set that the parameterization puts in ***XP*** or **FPT**. With polytime-segments, we have looked at the approximations with a running time that was bounded by a polynomial as a function of the length of the input. It may make sense to also consider the running time of an approximation as a function of the rank of an element in the domain of the approximation. If the domain of a polytime-approximation is of subexponential density, the running time may become superpolynomial as a function of the rank of inputs in the domain. This could be indicative of the existence of a more frugal encoding of objects with respect to the approximated set. In particular, an encoding similar to the multi-part encoding used in the proof of Theorem 3.3.20 may be viable and even invertible in polynomial time. This will be discussed in more detail in Section 3.4.5. Having exponentially dense slices is a guarantee that no recoding based on the parameterization is going to influence the fixed-parameter tractability of a set.

For parameterizations that have uniform exponential density, informativeness follows from a simple property.

3.3.23. Lemma. *Let η be a decidable parameterization having uniform exponential density. If we have $\mathrm{M}_\eta(n) \in \mathcal{O}(\log n)$, then η is informative.*

Proof:
Let f, ε, and γ witness the uniform exponential density of η. As, for all k, the value of $\mathrm{N}_\eta(n,k)$ cannot exceed 2^n, and the left-hand side of the density criterion is at most $2^{\varepsilon f(\mathrm{M}_\eta(n)) n^\gamma}$, we find $2^{\varepsilon f(\mathrm{M}_\eta(n)) n^\gamma} \leq 2^n$. Equivalently, this means that we have

$$f(\mathrm{M}_\eta(n)) \leq \frac{n}{\varepsilon n^\gamma} = \frac{n^{1-\gamma}}{\varepsilon}.$$

Because of the bound on M_η, there is a constant δ such that, for every n, we have $\mathrm{M}_\eta(2n) \leq \mathrm{M}_\eta(n) + \delta$. Since f is non-decreasing, we can combine this with the previous inequality to obtain

$$\begin{aligned} f(\mathrm{M}_\eta(n) - \delta) &\leq f(\mathrm{M}_\eta(n/2)) \\ &\leq \frac{(n/2)^{1-\gamma}}{\varepsilon} \\ &= \frac{n^{1-\gamma}}{\varepsilon 2^{1-\gamma}}. \end{aligned}$$

We can plug this bound on f into the right-hand side of the density criterion to find, for all k of length at most $\mathrm{M}_\eta(n) - \delta$,

$$\begin{aligned} \mathrm{N}_\eta(n,k) &\leq 2^{f(\mathrm{M}_\eta(n)-\delta) n^\gamma} \\ &\leq 2^{\frac{n^{1-\gamma}}{\varepsilon 2^{1-\gamma}} n^\gamma} \\ &= 2^{\frac{n}{\varepsilon 2^{1-\gamma}}}. \end{aligned}$$

More generally, for any constant c we find that, for all k of length at most $\mathrm{M}_\eta(n) - c \cdot \delta$, we have

$$\mathrm{N}_\eta(n,k) \leq 2^{\frac{n}{\varepsilon 2^{c(1-\gamma)}}}.$$

If we choose c large enough, then the denominator in the exponent, $\varepsilon 2^{c(1-\gamma)}$, will become larger than 1. Because we have $\mathrm{M}_\eta(n) \in \mathcal{O}(\log n)$, choosing c larger still we can make sure that we have $\frac{n}{\varepsilon 2^{c(1-\gamma)}} \leq n + c \cdot \delta - 2 \cdot \mathrm{M}_\eta(n)$. From this, because we have

$$\mathrm{N}_\eta(n,k) \leq 2^{n + c\cdot\delta - 2\cdot \mathrm{M}_\eta(n)} \leq 2^{n + |k| + c\cdot\delta - 2\cdot\mathrm{M}_\eta(n)},$$

it follows that η is informative. □

A parameterization of which informativeness can be shown using this lemma is the vertex cover parameterization.

3.3.24. Example. To be technically correct, we should distinguish between the **NP**-complete [63] set

$$\textsc{VertexCover} = \{(G, l) \mid \text{graph } G \text{ has a vertex cover of size at most } l\},$$

and the parameterization

$$\eta = (\{G \mid \text{the graph encoded by } G \\ \text{has a vertex cover of size at most asInt}(k)\})_{k \in 2^+}.$$

It is well known that VertexCover is in **FPT** when parameterized by the parameterization that bounds the size of the solution, $(\{(G, l) \mid l \leq \text{asInt}(k)\})_{k \in 2^+}$. Whether this last parameterization is informative depends on the pairing function that is used to combine G and l into a single string. Note that the parameterization that bounds the size of the solution also counts as a parameterization in the framework of Flum and Grohe. This is not the case for η, unless **P** equals **NP** and the minimum vertex cover size can be computed in polynomial time.

The parameterization η is emblematic of having uniform exponential density. Consider the adjacency matrix encoding of graphs and let n represent the number of cells in an adjacency matrix. Remark that the number of vertices in a graph of which the adjacency matrix has n cells is roughly $\sqrt{n}$. For any parameter value k, there are roughly $2^{\text{asInt}(k)\sqrt{n}}$ graphs of size n that have a fixed vertex cover of size at most asInt(k). This is so because any of the $\sqrt{n}$ vertices of the graph can only be connected to any of the asInt(k) elements of the vertex cover.

From Lemma 3.3.23, it now follows that the vertex cover parameterization is informative, because besides that it has uniform exponential density, we also have

$$\mathrm{M}_\eta(n) = |\text{asStr}(\sqrt{n})| \in \mathcal{O}(\log n).$$

That is, the largest minimum vertex cover has about as many vertices as there are available in the graph. The size of a minimum vertex cover can thus be specified in a number of bits that is logarithmic in the number of vertices in the graph.

Low Complexity

Turning to the other end of the complexity spectrum, we may ask whether low algorithmic complexity in turn implies low computational complexity. A formalization of the notion of low computational complexity is readily available in our parameterized framework. With respect to a parameterization η, a set is η-easy if on that set the minimization with respect to η is bounded by a constant. Using that a parameterization is directed, this leads to a pleasantly succinct definition.

3.3.25. Definition. With respect to a parameterization η, a set of strings X is *η-easy* if there is a parameter value k such that we have $X \subseteq \eta_k$.

As with our definition of hard sets, this definition is most meaningful in the context of a set that is in $\boldsymbol{X}\mathbf{P}$ or **FPT** with the particular parameterization. Then, a set is easy when it is a subset of a polytime-segment. This leads to a connection to algorithmic complexity related to Theorem 3.3.3. The key observation in the proof of that theorem is that sets of which the instance complexity with respect to a polynomial is bounded are polytime-segments. Since our notion of easiness is concerned with one slice only, we can remain in the realm of uniform parameterized complexity theory. We remark that any set that is covered by finitely many slices of a parameterization η is included in a slice of η, because parameterizations are directed.

3.3.26. THEOREM. *The following statements about a decidable set A and an arbitrary set X are equivalent.*

1. *There is a parameterization η with which A is in* **FPT** *such that X is η-easy.*
2. *There is a polynomial p and a constant c such that for all $x \in X$ we have $\mathrm{ic}^p(x : A) \leq c$.*

Proof:
$1 \implies 2$. This statement is similar to Theorem 3.3.5. Any slice of a parameterization with which A is in **FPT** is a polytime-segment of A. Thus, there is a polynomial p such that X is included in a p-segment of A. Consequently, there is a p-approximation for A that includes all members of X in its domain. By Definition 3.3.1, this means that the p-bounded instance complexity of the members of X can be bounded from above by a constant.

$2 \implies 1$. All p-approximations for A with a length of at most c can be combined into a single $\mathcal{O}(p)$-approximation for A. As a consequence, we get that X is the subset of some $\mathcal{O}(p)$-segment, say S, of A. Consider the parameterization of finite extensions of S given by

$$\eta = (S \cup \{x \mid |x| \leq \mathrm{asInt}(k)\})_{k \in 2^+}.$$

Essentially, this parameterization adds the set S to each slice of the length parameterization. By our observations on page 92, immediately below Example 3.2.16, we know that A is in **FPT** with η. Moreover, by construction, X is η-easy. □

This theorem lines up a notion of low computational complexity with a notion of low algorithmic complexity. As such it complements Theorem 3.3.20. However, we feel that bounded instance complexity with respect to a polynomial is not an adequate formalization of the opposite of randomness. Polynomially-bounded instance complexity depends on both a time bound and a reference set, whereas our definition of random sets depends on neither. In that regard, polynomially-bounded instance complexity is already very much a notion of computational complexity.

Looking at Definition 3.3.16, an obvious formalization of the opposite of randomness would be to require the Kolmogorov complexity to be bounded. A definition along these lines, though, would face two problems. Firstly, a set on which the Kolmogorov complexity is bounded from above by a constant is necessarily finite. Since the range of any function on a finite set is bounded, the proposed definition would be too demanding to be of use. Secondly, Kolmogorov complexity disregards the computational cost of producing a particular string. With random sets, the time required for decoding a compressed representation was of no concern. The Kolmogorov complexity of a member of a random set can be realized by a bit-for-bit specification, and such a specification can be decoded in linear time. The same is not true of strings with substantial redundancy in their descriptions. This is a drawback of the proposed definition as compressibility can only be taken advantage of if the redundant part of a string can be generated quickly.

A measure of algorithmic complexity that is not relative to a reference set and takes into account how fast a string can be produced was proposed by Hartmanis [74]. According to this measure, the opposite of a random set is a set of small generalized Kolmogorov complexity [15, 8].

3.3.27. Definition. A set of strings X has *small generalized Kolmogorov complexity* if there is a constant c such that, for all n and all $x \in X$ of length n, there is a procedure ϕ that satisfies

- $|\phi| \leq c \log n$, and
- ϕ outputs x within n^c steps.

Note that for any n and any fixed c the number of procedures of length at most $c \log n$ is polynomial in n. Deciding whether a given x of length n is produced by any of these procedures within n^c steps is therefore possible in a time bounded polynomially in n. Yet, this does not mean that every set with small generalized Kolmogorov complexity is in **P**. Every subset of a set with small generalized Kolmogorov complexity, no matter how difficult to decide, has small generalized Kolmogorov complexity. To add a sense of uniformity, we may want to think only of the sets in **P** with small generalized Kolmogorov complexity as nonrandom sets. As it turns out, this gets us another class of sets of highly structured strings. The class we get was first identified by Hartmanis and Yesha [78].

3.3.28. Definition. A set of strings X is *p-printable* if there is a polynomial p and a procedure ϕ that, on any input $\text{asStr}(n)$, lists all members of X with a length of at most n within $p(n)$ steps.

The equivalence of the class of p-printable sets and the class of sets in **P** with small generalized Kolmogorov complexity was shown independently by Balcázar and Book [15], and Rubinstein [132]. Shortly after, Allender and Rubinstein [8] identified two more ways of characterizing the p-printable sets. They observed that a set is p-printable precisely when it is polynomial-time isomorphic to some *tally set*, any subset of $\{\mathtt{0}\}^+$, in **P**. Consequently, they showed that a set is polynomial-time isomorphic to a tally set in **P** precisely when it is sparse and p-rankable [see also 69]. Whether or not all sparse sets in **P** are p-rankable and thus p-printable is an open problem. Yet, oracles are known relative to which not all sparse sets in **P** are p-printable [74].

With Theorem 3.3.20, we have found that random sets are hard in relation to any informative parameterization. For the converse, we may expect every set with small generalized Kolmogorov complexity to be easy in relation to some parameterization. While we shall leave this as an open problem, it has been shown that such a theorem would be at odds with the Berman–Hartmanis conjecture.

3.3.29. Theorem (Hartmanis [74, Theorem 14]). *Let f be a function that grows faster than any polynomial, and X the set of strings such that for all n and all $x \in X$ of length n, there is a procedure ϕ that satisfies*

- $|\phi| \leq \log n$, *and*
- ϕ *outputs x within $f(n)$ steps.*

If X is easy for some parameterization with which an **NP***-complete set is in* **FPT***, then there are nonisomorphic* **NP***-complete sets.*

Notice that the set X in the theorem above is not actually a set with small generalized Kolmogorov complexity, as the time bound f is not polynomial. Still, the theorem puts a point on the map for the study of the parameterized tractability of **NP**-complete sets in relation to the Berman–Hartmanis conjecture.

3.4 as Model Classes

To a deity that has an unbounded amount of computation time at its disposal, all that can be known about an object x can be conveyed in $\mathrm{K}(x)$ bits. In that sense, Kolmogorov complexity is a measure of the information in an object. In practice, however, part of the information thus measured may be out of reach. Next to computational reasons, this may be because of the expectations an observer has about an object. As an observer may not be receptive to all information, it may be possible to convey a sufficient part of the information in an object x in under $\mathrm{K}(x)$ bits. This is what lossy compression aims for.

3.4.1. Example. To get a feel for the amount of information in an object that is relevant, we take a brief look at image compression. The Kolmogorov complexity of an image can be approximated from above using a lossless graphics file format such as Portable Network Graphics, PNG [133]. In turn, the information that is actually relevant to a human observer can be approximated using a lossy file format such as JPEG [133]. Looking at the Kodak True Color Image Suite [61], a standard test set, we find the following mean file sizes [139].

PNG	*JPEG (Q=90)*
641 kilobyte	136 kilobyte

Thus, the approximate mean Kolmogorov complexity of the images in the test suite is 641 kilobyte, and their approximate mean useful information is 136 kilobyte. This puts the fraction of the information in an image that is essential to its qualities as an image at around 21%.

In the presentation of an object, a certain amount of redundancy is desirable. Hardly ever do we want an object to be displayed as a string witnessing just its useful information, or even its Kolmogorov complexity. When we derive a length notion from a more natural presentation, this length can be divided into three parts. These parts are: the *useful information*, the *noise*, and the *redundancy*, as shown in Figure 3.5. Here, noise is the difference between the Kolmogorov complexity and the useful information.

3.4.2. Example (continued). The most natural presentation of an image is as its intended graphical display. Via the bitmap image file format, BMP, we get a length measure that is, at least in spirit, derived from this means of presentation. The images in the Kodak image suite all contain $768 \cdot 512 = 393216$ pixels, each recorded with 3 bytes of color depth. Thus, the BMP size of each image is 1180 kilobyte. In relation to this, the images contain, on average, at most 11% useful information, around 43% noise, and at least 46% redundant information.

That not all of some object's natural length is allotted to useful information can also be seen from information-hiding techniques. Homoglyphs are text characters

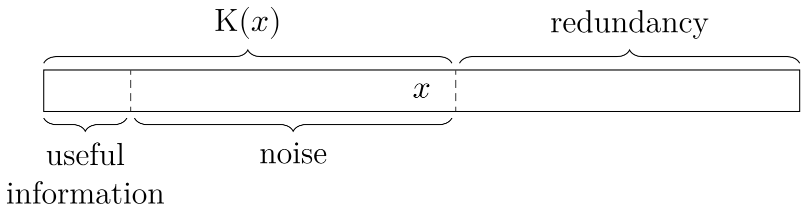


Figure 3.5: The length of a string x, decomposed into the Kolmogorov complexity of x and the redundancy in x. The Kolmogorov complexity is further decomposed into useful information and useless noise. Relative sizes are depicted in accordance with Example 3.4.2.

that are not perceived to be different, yet have different encodings. By selectively replacing characters in a text message by their homoglyphs, information can be injected into a message without ostensibly altering it. Arguably, the information in a text message is not even altered when words or phrases in it are replaced by synonyms. This, too, can be used for information hiding, or, more technically, for *steganography* [84].

While the useful information is left unaltered by steganography, this does not mean that the presence of hidden information cannot be detected. Such detection is the goal of *steganalysis*. For instance, it is not unreasonable to expect that the use of homoglyphs reduces the redundancy in a message. Thus, a carrier message *without* hidden information may be more compressible than a message *with* hidden information. Applied to images, techniques are developed to restrict the action of steganographic information hiding to the noise in an image. These techniques evade detection by simple compression tests. An overview of steganography methods for images is given by Cheddad et al. [32].

Synopsis

Quantifying the useful information in an object is the domain of *algorithmic statistics*. This is a formulation of statistics centered on model selection for individual data samples. Algorithmic statistics will be introduced in more detail in Section 3.4.1. The associated formalization of useful information, *sophistication*, is the topic of Section 3.4.2. Algorithmic statistics includes a notion of goodness-of-fit grounded in Kolmogorov complexity, where computational resource usage is not considered. Consequently, sophistication targets a universal observer with unbounded computational powers. This approach is complicated by the fact that, unlike Kolmogorov complexity, sophistication differs in an unbounded way among universal observers. That is, sophistication is sensitive to the encoding used for specifying procedures. Additionally, if we consider reasoning to be a form of computation, then it makes sense to include computational complexity in a measure of useful information. This should make it possible to target a human observer instead of a universal observer. With parameterizations, we can formalize useful information in a context-sensitive way. Moreover, since parameter values can provide a measure of computational complexity, a parameterized sophistication can include computational considerations. Doing so tightens the relationship between algorithmic complexity and computational complexity.

Previously, we looked at the relationship between parameter values and the algorithmic complexity of individual strings. Here, we shall continue that investigation. We shall see that parameterizations provide an alternative to the use of Kolmogorov complexity in algorithmic statistics. The parameterized take on algorithmic statistics includes the traditional version as a special case. This enables us to use traditional algorithmic statistics as a running example in the section on sophistication, Section 3.4.2.

Many theorems and questions that are central to algorithmic statistics are about the existence and prevalence of objects that lack simple statistical models. In Section 3.4.3, we continue our generalization of algorithmic statistics so that parameterized counterparts to such theorems and questions can be stated. We observe that the idea of simplicity of a model for an object requires a notion of length to be available for the object. Throughout our treatment of parameterized algorithmic statistics, we make a distinction between an object and a string representation of that object. Therefore, a notion of length is only available for objects after we have decided on a way to encode objects. This lets us conclude that an *object* can only be incompressible if it has a truly canonical encoding, which is only the case if the object is already a string.

In Section 3.4.4, we get to see how the inclusion of parameterizations makes the theory of algorithmic statistics more fine-grained. The section places sophistication in the context of three well-known strategies for model selection in statistics: Occam's razor, the maximum likelihood principle, and the minimum description length principle. Additionally, in this section we find that most strings have simple models, also in parameterized algorithmic statistics. These simple models would even pass as good simple models in traditional algorithmic statistics for infinitely many objects. Thus, for infinitely many objects, the useful information reflected by a parameterization is really all the useful information there is.

Questions surrounding the prevalence of objects with simple models can also be turned around in parameterized algorithmic statistics. We do so in Section 3.4.5, where we focus on parameterizations corresponding to parameterized algorithms that are successfully applied in practice. From the success of these algorithms, we can infer what parameter values we are most likely to encounter. To wit, we know that the expected parameter value is small, like we have seen all the way back in Example 1.2.1. For a fixed parameterization, this gives us information about what a good statistical model of the real world should look like. In turn, this knowledge can be used as a guiding principle in the design of suitable data structures for dealing with real-world data. Section 3.4.6 explores the properties and possibilities of data structures rooted in parameterizations.

3.4.1 An Overview of Algorithmic Statistics

The crucial insight that makes algorithmic statistics possible is that the Kolmogorov complexity of an object can always be realized by a two-part code. Every object can be described by two strings of which the combined length equals, up to an independent additive constant, the Kolmogorov complexity of the object. The two parts are commonly called *model* and *data-to-model code*. The second part is so named because it describes the object relative to the model. In the pairing of the two parts it must be clear where one part ends and the other begins.

In traditional applications of the minimum description length principle [128, 127, 150, 72] the word 'model' is used to indicate a functional form. The second part of the two-part code is then used to both instantiate a single function *and* specify the data with respect to it. In algorithmic statistics, however, the two-part code should separate the meaningful information in an object from the random noise in that object. Algorithmic statistics is not concerned with any structure inside the second part of the two-part code. Thus, the word 'model' is used for what is known as a *point hypothesis* or *exact hypothesis* in inferential statistics. More concretely, in algorithmic statistics, we interpret models as probability distributions. In turn, data-to-model codes are interpreted as entropy encodings in accordance with those probability distributions [148, 36].

A measure of the informativeness, the *sophistication*, of an object is available in the form of the minimum complexity of a model for the object. This minimization is restricted to those probability distributions with which the two-part code for the object realizes the Kolmogorov complexity of the object. Unfortunately, there is no clear-cut way to define the complexity of a probability distribution. If we would allow the universal distribution [100] to have a finite complexity, then it would realize the Kolmogorov complexity of every object. The finite complexity of the universal distribution would present a constant additive overhead to the Kolmogorov complexity. However, Kolmogorov complexity is only defined up to an additive constant. Likewise, the universal distribution is only defined up to a multiplicative constant. Hence, this additive overhead is unavoidable to begin with. In that sense, the universal distribution would act as a universal model [149, 21]. Disregarding additive constants for a moment, we see that such a universal model would place a universal bound on the informativeness of any object. This renders our measure of informativeness useless. To which probability distributions we assign finite complexity is hence a central question in the development of algorithmic statistics [147]. The class of probability distributions considered is called the *model class*. While the choice of a model class is usually an ad hoc choice in applications of the minimum description length principle [150, 72], algorithmic statistics calls for a single generic model class. The lack of a universally best model class, according to some motivation from outside the theory, is a major obstacle for algorithmic statistics.

Many model classes have been considered in the existing literature. When interpreted as classes of probability distributions, there is significant variation in the requirements that have been put on the computability of the probabilities. Regardless of this variation, the studied model classes can be characterized by a classification of the *support* of the probability distributions they contain. Least restrictive are the model classes where the support of a probability distribution can be any semidecidable set. Such model classes are present in the works of Rissanen [127], Koppel [96], Gács, Tromp, and Vitányi [62] (as implicit probabilistic models), Vitányi [149], and Antunes and Fortnow [10]. A model class where exactly the decidable sets occur as the support of the probability distributions is hinted at by Gács, Tromp, and Vitányi [62] (as explicit probabilistic models). However, their definitions are not strong enough to enforce decidability. The most fruitful model class has been that of uniform probability distributions with arbitrary, yet necessarily finite, support. This is the model class originally proposed by Kolmogorov and most prominent in the works of Gács, Tromp, and Vitányi [62] and Vereshchagin and Vitányi [148]. A slightly more general model class where the probability distributions have finite support, yet are allowed to take on different rational-valued probabilities on their support, was considered by Vereshchagin and Shen [147].

All of the model classes studied by the aforementioned authors exclude universal models. Yet, none are presented with a guarantee against putting meaningful information in the data-to-model code, thus *underfitting* the data. As argued by Bloem, De Rooij, and Adriaans [21], it is likely that these model classes do in fact contain models that are “too universal”. That is, some models are universal enough to bound the informativeness of some objects by a value less than the informativeness we ascribe to them intuitively. Such model classes misrepresent the informativeness of certain objects that we think of as containing meaningful information. At the same time, Vereshchagin [146], Bloem, De Rooij, and Adriaans [21], and Antunes et al. [11] show that the choice of the complexity measure used to measure the complexity of models matters. Sophistication, which is described in more detail in the next section, in particular is sensitive to this choice. While Kolmogorov complexity is invariant under the choice of a universal machine, sophistication, despite being based on Kolmogorov complexity, is not. This is true regardless of how we adapt Kolmogorov complexity for measuring the complexity of probability distributions, which can be done in many ways [62]. Slight variations in the model complexity can change a model from realizing the Kolmogorov complexity of an object to not realizing it and vice versa. Hence the minimum complexity of a probability distribution with which the two-part description of an object realizes the Kolmogorov complexity of that object can change significantly with a change of the reference universal machine. As a result, some choices of a reference machine may lead to widespread *overfitting* [21]. When performing maximum likelihood estimation, this form of overfitting can be mitigated by working with complexity-constrained subclasses of the model class [148]. However, that brings us no closer to a definitive measure of the useful information in an object. Alternatively, one can adjust the complexity measure for models so that the selection of overfitting models is penalized. This is done by Rissanen [127] and Antunes and Fortnow [10], but it is not proven to be adequate and may lead to underfitting instead [21].

We propose to include the model class and its associated complexity measure as a degree of freedom in the framework of algorithmic statistics. Indeed, we shall use parameterizations as model classes with an inherent complexity measure. Besides addressing underfitting and overfitting, doing so also makes it possible to examine complexity measures other than the Kolmogorov complexity. The uncomputability of Kolmogorov complexity has been identified as an impediment to applications of algorithmic statistics [127, 147]. Remarkably, the existing literature on algorithmic statistics has so far only explored in detail variations of Kolmogorov complexity as complexity measures. At the same time, the potential for restricted model classes has been identified [22, 147]. By introducing an explicit dependence on a model class, we accept that a notion of useful information may be subject to one's expectations. There appears to be no universally best model class, or, in other words, no universal prior for algorithmic statistics. Because of that, the dependence of a notion of useful information on a model class is likely unavoidable.

3.4.3. SLOGAN. *Useful information is context-dependent.*

Accepting that a notion of useful information is dependent on a model class justifies the addition of a model class variable to the framework.

3.4.2 Sophistication

Traditionally, algorithmic statistics looks at describing a string x with the help of a probability distribution P that assigns nonzero probability to x. The probability mass function P is required, at the least, to be rational-valued and computable [147]. Knowing P, a code that is optimal with respect to P uses $-\log P(x)$ bits for describing x in prefix-free fashion [36]. Now, let us use $|P|$ for the length of a description of P in some fixed prefix-free encoding of rational-valued computable probability mass functions. We find an upper bound on the Kolmogorov complexity of x in the form of a two-part description of x,

$$\mathrm{K}(x) \leq |P| - \log P(x).$$

How tight this bound can get depends on the chosen method of encoding probability mass functions. It is customary to focus on methods that are as efficient as possible. An extreme case is provided by the probability distributions P of which the support is a singleton, $\{x\}$, and we have $P(x) = 1$. For such a probability distribution P, the length $|P|$ can be made to match, up to an additive constant, the Kolmogorov complexity of x. With encodings that accomplish this, the above equation can become an equality.

Since we are describing the string x using two-part codes, we may investigate the many ways in which a description of x can be composed of two parts. In particular, we are interested in the ways the Kolmogorov complexity of x can be split in two by probability mass functions. Naturally, when we restrict our model class, we limit the number of ways to describe x. However, some model classes nearly cover all possible ways to balance the lengths of the two parts. One such model class is that of *uniform* distributions. The support of a uniform probability mass function is a finite set. For a finite set A, let $\langle A\rangle$ denote a listing of all elements of A and notice that the number of elements of A can be recovered from $\langle A\rangle$.

3.4.4. Theorem (Vereshchagin and Shen [147, Remark 1 & Proposition 6]). *Let P be a rational-valued computable probability mass function and x a string to which P assigns nonzero probability. There exists a finite set A of m elements including x such that we have*

$$\mathrm{K}(\langle A\rangle) \leq |P| + \mathcal{O}(\log |x|) \quad \textit{and} \quad -\log\frac{1}{m} \leq -\log P(x) + \mathcal{O}(1).$$

In other words, up to terms logarithmic in the length of a string x, we may restrict our attention to uniform distributions. Uniform distributions effectively maximize the minimum probability in a distribution with a given finite support. For distributions with a given infinite support, there is no clear-cut way of maximizing the minimum probability. Therefore, it is interesting to note that there is a variant of Theorem 3.4.4 for a model class where distributions have potentially infinite support. Note that the $-\log\frac{1}{m}$ term in Theorem 3.4.4 represents the fact that $\log m$ bits suffice to tell the elements of the finite set A apart. With Elias delta coding as defined in Section 2.1.1, we can single out the rth element of any decidable set using $\log r + 2\log\log r$ bits. Prefix-free codes such as the Elias delta coding are related to probability distributions by the *Kraft inequality* [36, 100]. An element that is encoded using ℓ bits is accordingly assigned a probability of $2^{-\ell}$. The probability distributions stemming from Elias delta coding via the Kraft inequality can have infinite support. It is the model class of such distributions that has a property similar to that codified in Theorem 3.4.4 for the model class of uniform distributions. For a decidable set A, let $\mathrm{K}(A)$ denote the least length among the lengths of decision procedures for A.

3.4.5. Theorem. *Let P be a rational-valued computable probability mass function and x a string to which P assigns nonzero probability. There exists a decidable set A including x such that, with $r = \operatorname{rank}(x : A)$, we have*

$$\mathrm{K}(A) \leq |P| + \mathcal{O}(\log |x|)$$

and

$$\log r + 2\log\log r \leq -\log P(x) + \mathcal{O}(1).$$

Proof:
Let S be the support of P and let r_S be the rank of x in this set. In other words, we define S as $\{y \mid P(y) > 0\}$ and we define r_S as $\operatorname{rank}(x : S)$. Furthermore, set i to the unique value for which we have

$$\frac{i^2\, 2^i}{(\log r_S)^2\, r_S} < P(x) \leq \frac{(i+1)^2\, 2^{i+1}}{(\log r_S)^2\, r_S}.$$

Observe that i must be less than $\log r_S$, because $P(x)$ is at most 1. Therefore, we have $i \in \mathcal{O}(|x|)$ and the number of bits needed to specify i is in $\mathcal{O}(\log |x|)$.

Now, consider the set

$$H = \left\{ y \;\middle|\; P(y) > \frac{i^2\, 2^i}{(\log \operatorname{rank}(y : S))^2\, \operatorname{rank}(y : S)} \right\}.$$

We find that x is included in H. Observe that all members of H preceding x must have a probability of at least

$$\frac{i^2\, 2^i}{(\log r_S)^2\, r_S}.$$

Therefore the rank of x in H is at most

$$\frac{(\log r_S)^2\, r_S}{i^2\, 2^i}, \tag{3.7}$$

as otherwise the sum of the probabilities would exceed 1. The rank of x in H can thus be upper bounded by $\frac{1}{P(x)}$, which means H almost satisfies the requirements on the set A of the theorem. To complete the proof, all that remains is to move some complexity from the data-to-model code into the model. The set H can be partitioned into $|x|^2$ subsets $H_1, H_2, H_3, \dots, H_{|x|^2}$ via

$$H_j = \{y \in H \mid \operatorname{rank}(y : H) \equiv j \pmod{|x|^2}\}.$$

For exactly one j, the set H_j contains x. Let A be this set H_j. That is, A is a thinned-out version of H that includes x. By construction, A contains at most one out of every $|x|^2$ elements of H. Following (3.7), the rank of x in A is therefore at most

$$\frac{(\log r_S)^2\, r_S}{i^2\, 2^i} / |x|^2,$$

rounded upward if needed. Hence, the rank of x in A is at most $\frac{r_S}{i^2\, 2^i} + 1$ and if we let r denote this rank, we find

$$\begin{aligned}
\log r + 2 \log\log r &\leq \log \frac{r_S}{i^2\, 2^i} + 2 \log\log \frac{r_S}{i^2\, 2^i} + \mathcal{O}(1) \\
&\leq \log r_S - 2\log i - i + 2\log\log r_S + \mathcal{O}(1) \\
&\leq -\log \frac{(i+1)^2\, 2^{i+1}}{(\log r_S)^2\, r_S} + \mathcal{O}(1) \\
&\leq -\log P(x) + \mathcal{O}(1),
\end{aligned}$$

as desired. Additionally, A was constructed from P, i, and j, so we also have $\mathrm{K}(A) \leq |P| + \mathcal{O}(\log|x|)$. $\square$

The message of Theorem 3.4.5 is that, up to terms logarithmic in the length of a string x, we may restrict our attention to distributions in accordance with Elias delta coding. This allows us to focus on decidable sets as models, and on parameterizations with decidable slices as model classes.

3.4.6. Definition. A *parameterized model* is a slice of a parameterization, represented by its corresponding parameter value.

We call a parameterization η a *parameterized model class*, and a parameter value k an *η-model* for a string x if we have $x \in \eta_k$. By using the same coding method across all slices, across all parameterizations even, we are able to isolate the useful information about an object. We are unable to express any properties other than the rank of a string x in a slice η_k if our data-to-model code is confined to Elias delta coding. Therefore, useful information can only accumulate in a specification of the model, thus in the parameter value k.

One way to think of a parameterization in this context is as a sort of alternative computability universe. In such an alternative universe, the slices of the parameterization take the role of "decidable" sets. Parameter values then encode the equivalent of the corresponding decision procedures. Of course, when not all slices of the parameterization are decidable, this alternative universe is different from the universe we inhabit. However, even when all slices are decidable, the parameterization may still challenge our notion of how computability behaves. The parameter value associated with a slice may not be related to anything we would reasonably want to call a decision procedure for the slice. On the other hand, when we can effectively map the parameter values to decision procedures for the respective slices, we are in a fairly restricted setting.

3.4.7. Example. When we think of ways to encode procedures, we normally only think of so-called *acceptable* encodings [131]. Loosely speaking, there must be an effective way to interpret an encoded procedure and perform the computations it represents. All acceptable encodings lead to the same notion of Kolmogorov complexity. Our choice of an encoding has been somewhat implicit, but it plays a role in the parameterization given by

$$(\{x \mid k \text{ encodes a decision procedure that accepts } x\})_{k\in 2^+}. \tag{3.8}$$

In this parameterization, there are two kinds of slices. For a parameter value k that encodes a decision procedure, the corresponding slice is the set decided by that decision procedure. In other words, the decision procedure encoded by k outputs `1` on the members of slice k and it outputs `0` on the nonmembers. However, if k does not encode a decision procedure, for instance because k does not encode a total function, then the corresponding slice is empty. Unfortunately, by Rice's theorem [125], this means we cannot effectively map a parameter value to a decision procedure for the corresponding slice.

Suppose it is possible to map parameter values to decision procedures for the corresponding slices of a parameterization effectively. Then, the minimum length of a decision procedure for a slice is, up to an additive constant, bounded from above by the length of the corresponding parameter value. The additive constant only depends on the algorithmic complexity of the function mapping parameter values to decision procedures. Because this constant is fixed for the parameterization, we may ignore it and use the length of a parameter value as the length of a decision procedure. For more unusual parameterizations such a constant may not exist. Yet, we may still use the length of a parameter value as a measure of the complexity of a slice. In general, a nice starting point for measuring the algorithmic complexity of a string with respect to a parameterization is the following.

3.4.8. Definition. The *parameterized complexity* of a string x with respect to a parameterization η is

$$\mathrm{pc}_{\eta}(x) = \min\{|\langle k, \mathrm{rank}(x : \eta_k)\rangle| \mid k \in 2^{+} \text{ such that we have } x \in \eta_k\}.$$

Note that, strictly speaking, the above definition should look at pairs of a parameter value and a *string representation* of the rank. To improve readability, we shall not explicitly state the conversion of the rank, which is a number, to a string when dealing with such pairs.

As parameterizations are covers of 2^{+}, the parameterized complexity with respect to a parameterization is defined for every string. The choice of a pairing function may have an effect on the parameterized complexity. It may affect not only its value, but also the parameter value with which this value, as a minimum of multiple candidates, is realized.

3.4.9. Example (continued from Example 3.4.7). Our pairing function, Definition 2.1.2, uses a prefix-free code, the Elias delta code, for its second component. As a result, the number of bits used to encode the rank in the definition of parameterized complexity aligns nicely with Theorem 3.4.5. The pairing function does not use a prefix-free code for its first component. However, when encoding procedures, it is natural to use a prefix-free code to begin with [100]. Suppose we use a prefix-free encoding of decision procedures in the parameterization given by (3.8). In that case, parameterized complexity with respect to the parameterization given by (3.8) is essentially Kolmogorov complexity.

To see that it is at most the Kolmogorov complexity, observe that for every string x, there is a set $\{x\}$ with a decision procedure of length $\mathrm{K}(x) + \mathcal{O}(1)$. In the set $\{x\}$, the rank of x is 1, hence the parameterized complexity of x

is at most $\mathrm{K}(x) + \mathcal{O}(1)$. At the same time, the Kolmogorov complexity of x cannot be substantially higher than its parameterized complexity. Consider a procedure that, given a string k and a number r, simulates the decision procedure encoded by k and finds the rth string accepted by it. This procedure recovers x from any $\langle k, \mathrm{rank}(x : \eta_k)\rangle$, whenever x is a member of η_k. When we hardcode the pair $\langle k, \mathrm{rank}(x : \eta_k)\rangle$, we obtain a new procedure of which the length is $|\langle k, \mathrm{rank}(x : \eta_k)\rangle| + \mathcal{O}(1)$. Hence the parameterized complexity of x is not only at most $\mathrm{K}(x) + \mathcal{O}(1)$, but also at least $\mathrm{K}(x) - \mathcal{O}(1)$.

The example above demonstrates how parameterized complexity generalizes Kolmogorov complexity. It showed that for some parameterization, the two notions of complexity coincide on all strings, up to an independent additive constant. Other parameterizations may show the same behavior, or show it only for some strings. We say that a parameterization η *captures* the (Kolmogorov) complexity of a string x if, up to a fixed additive constant independent of x, we have $\mathrm{pc}_\eta(x) = \mathrm{K}(x)$. Note that in this definition, several hidden constants are at work. To be precise, we should first choose a constant c and fix a reference encoding of procedures that we use to define Kolmogorov complexity. Having done so, a parameterization η captures the complexity of those strings x for which we have

$$\mathrm{K}(x) - c \leq \mathrm{pc}_\eta(x) \leq \mathrm{K}(x) + c.$$

One benefit of parameterized complexity over Kolmogorov complexity is that it can be computable, even if it captures the complexity of some strings.

3.4.10. Lemma. *If η is a decidable parameterization, then pc_η is a computable function.*

Proof:
A procedure computing the parameterized complexity of a string x could proceed as follows.

```
1: For each parameter value $k$, in order of increasing length:

    1.1: If $\eta_k$ includes $x$,
         break out of the current loop and continue at step 2.

2: For each parameter value $k'$ of length at most $|\langle k, \mathrm{rank}(x : \eta_k)\rangle|$:

    2.1: If $\eta_{k'}$ includes $x$ and $|\langle k', \mathrm{rank}(x : \eta_{k'})\rangle|$ is less than $|\langle k, \mathrm{rank}(x : \eta_k)\rangle|$,
         set $k$ to $k'$.

3: Return $|\langle k, \mathrm{rank}(x : \eta_k)\rangle|$.
```

The unbounded loop at the first step of this algorithm is guaranteed to be exited from, because η is a cover of 2^+. The bounded loop at the second step makes sure that the returned value is indeed the minimum length of all candidate lengths. Thus, the returned length is the parameterized complexity of x with respect to η. Note that the algorithm is allowed to test whether a slice of η includes x because η is assumed to be decidable. □

For a given string x, there may be more than one parameterized model that witnesses the minimum in the definition of parameterized complexity. Those that do are particularly interesting parameterized models of x. Because they realize the parameterized complexity of x, these models contain all information about x that is available in the parameterization. This is a form of *sufficiency*, as used in parametric probabilistic statistics [see also 36, Section 2.9].

3.4.11. Definition. A parameter value k, with respect to a parameterization η, constitutes a *sufficient parameterized statistic* for a string $x \in \eta_k$ if we have

$$|\langle k, \operatorname{rank}(x : \eta_k)\rangle| = \operatorname{pc}_\eta(x).$$

In other words, an η-model k for x is a sufficient parameterized statistic if $|\langle k, \operatorname{rank}(x : \eta_k)\rangle|$ is minimal.

3.4.12. Example. In parametric probabilistic statistics, we are dealing with model classes indexed by a parameter, say, θ. A *statistic* is a function of a data sample into some arbitrary set and it is *sufficient* if it preserves the information in the sample about the parameter. The identity function is always a sufficient statistic, but more interesting sufficient statistics may be available. The textbook example is a binomial distribution with a fixed number of trials, n, and an unknown success probability for each trial, p. The model class is then the class of binomial distributions with parameters n and p. As n is fixed, this class is indexed by p.

Given a data sample of the outcomes of n trials, we can make a prediction about the value of p that was used in generating the sample. The only property of the sample that is relevant for our judgment turns out to be the number of successes. If there are i successes in our sample, our best guess for the parameter p is $\frac{i}{n}$. Indeed, the mapping from the sample to the number of successes is a sufficient statistic.

The relation to our parameterized sufficiency criterion is most visible with respect to the parameterization given by (3.8). With that parameterization, a parameter value k that encodes a decision procedure for a set S is a sufficient statistic for a string x if we have

$$|\langle k, \operatorname{rank}(x : S)\rangle| = \mathrm{K}(x).$$

The right side of this equation can be interpreted as "all the information in x". The equation thus states that when we replace x by its rank in S, we have not lost any of the information in x. In other words, k preserves the information in x about decidable sets that contain x. It is in this sense that a sufficient parameterized statistic is related to a sufficient probabilistic statistic.

Looking at the sufficient parameterized statistics for a string x, we can see what complexity we should expect, at least, of a parameterized model for x. To use this minimum model complexity as a measure of the informativeness of x was suggested by Koppel [96]. Conceptually, however, it is a notion of simplicity needed for a formalization of Occam's razor. While the razor has received lots of criticism, it has proven its worth many times, in both theory and applications [72].

3.4.13. Definition. The *parameterized sophistication* of a string x with respect to a parameterization η is

$$\operatorname{soph}_\eta(x) = \min\{|k| \mid k \text{ is a sufficient parameterized statistic for } x\}.$$

Notice that, for every parameterization η and string x, the sophistication $\operatorname{soph}_\eta(x)$ is lower bounded by $\mu_\eta(x)$,

$$\mu_\eta(x) \leq \operatorname{soph}_\eta(x).$$

Indeed, the parameterized sophistication soph_η need not equal the minimization function μ_η. For example, the minimization with respect to the parameterization given by (3.8) is bounded from above by the length of a decision procedure for 2^+. However, the sophistication with respect to that parameterization can get arbitrarily large. Still, when η satisfies

$$\forall k, k' \colon |k| \leq |k'| \implies \eta_k \subseteq \eta_{k'}, \tag{3.9}$$

the rank of a string in a slice only increases with the length of its η-models. In other words, when the inclusion order of η is compatible with the encoding of the parameter, a stronger version of (3.6), the functions soph_η and μ_η *do* coincide. Note that when η is decidable, soph_η is a computable function. This result is similar to the computability of pc_η and can also be seen in the proof of Lemma 3.4.10.

3.4.14. Example (continued from Example 3.4.9). The traditional notion of sophistication by Koppel [96] is essentially sophistication with respect to the parameterization given by (3.8). Conceptually, sophistication is an algorithmic minimal sufficient statistic [62, 148]. The most common model class with respect to which sophistication is computed is the model class of finite sets [21]. Using Theorem 3.4.4, it is then argued that, in effect, the entire class of rational-valued computable probability mass functions is taken into account. Similarly, our definition with respect to the parameterization given by (3.8) pertains to this broad class by Theorem 3.4.5.

One thing to note is that traditional sophistication is defined with reference to a traditional notion of an algorithmic sufficient statistic. The benchmark for this traditional notion is Kolmogorov complexity, whereas the notion of a sufficient parameterized statistic hinges on parameterized complexity. The Kolmogorov complexity of a string depends on a chosen encoding of procedures, and is therefore only defined up to an additive constant. As a result, traditional sophistication is dependent on the choice of a constant. For parameterized sophistication, the choice of a constant is replaced by the choice of a parameterization.

Whatever parameterization η we are working with, each string x is assigned a finite sophistication $\mathrm{soph}_{\eta}(x)$. This is a consequence of the fact that parameterizations are covers of 2^{+}. In particular, for every parameterized model class η and every string x, there is an η-model for x that is a sufficient parameterized statistic for x. Thus, in a probability theory derived from η, we are able to carry out statistical inference on samples of only a single object, the string x. In this probability theory, we can assign a prior probability to each slice of η as a function of the length of the corresponding parameter value. For instance, we may do so by fixing a prefix-free encoding of parameter values and using the Kraft inequality. This highlights a similarity between two-part codes and Bayesian statistical inference [72]. The probability of the event is then similarly assigned using the Kraft inequality and the length of the rank in the slice, encoded in a prefix-free way. Using our pairing function, Definition 2.1.2, the parameterized complexity, Definition 3.4.8 depends on an Elias delta encoding of the rank. We may use the same encoding for parameter values. If we use $\ell(z)$ to denote the length of the Elias delta encoding of the string z, we can be a bit more precise about our probability distributions. The prior probability of a slice corresponding to a parameter value k becomes $2^{-\ell(k)}$. Likewise, given an η-model k for a string x, the probability of x in this model becomes $2^{-\ell(\mathrm{asStr}(\mathrm{rank}(x:\eta_k)))}$. These two probability distributions enable statistical inference.

Apart from being a cover of 2^{+}, parameterizations, as families of sets, are also directed. Consequently, not only singleton samples are events, but all finite samples are events. For every parameterized model class η and every finite set of strings X, there is an η-model k such that X is included in η_k. An investigation of algorithmic statistics in light of finite events of more than one string was carried out by Milovanov [107]. The model class used by Milovanov is that of finite sets. To enable similar investigations in more general model classes, a few conditions to model classes are outlined by Vereshchagin and Shen [147, Section 6.1]. By being directed, parameterizations meet their second condition, which requires that for every n, some model must contain all strings of length n. This positions parameterizations as an alternative to the model classes of restricted type of Vereshchagin and Shen [147]. However, their model classes are also required to be decidable. They place this requirement so that the emanating notions of complexity are bounded in one way or another by those derived from their most general model class.

Let us denote the traditional sophistication, that with respect to the parameterization given by (3.8), of a string x by $\operatorname{soph}(x)$. Suppose η is a decidable parameterization and denote by $\mathrm{K}(\eta)$ the minimum length of a procedure that, given some x and k, decides whether x is in η_k. Then, we find, up to an additive constant,

$$\operatorname{soph}(x) \leq \operatorname{soph}_{\eta}(x) + \mathrm{K}(\eta). \tag{3.10}$$

This holds because every η-model k for x has a Kolmogorov complexity bounded, up to an additive constant, by $|k| + \mathrm{K}(\eta)$. Even when η captures the complexity of x, the above comparison of sophistications may be a strict inequality. For example, the traditional sophistication of a string x of which the Kolmogorov complexity is roughly $|x| + 2 \cdot \log |x|$ is bounded from above by a constant. This is because the set of all strings, $\mathbb{2}^+$, is a sufficient model for such a string. However, $\mathbb{2}^+$ need not occur as a slice of η, even when η captures the complexity of x. Therefore, the parameterized sophistication of such strings need not be bounded by a constant and the two notions of sophistication can differ.

Outside of decidability requirements, similar parameterizations may lead to different measures of sophistication too. As different ways of encoding decision procedures lead to different length measures, the length of a decision procedure is only defined up to an additive constant. This additive constant is the bane of algorithmic statistics. Surely, the various parameterizations defined by (3.8) for different encodings of decision procedures are all equivalent in the sense of Definition 3.2.20. However, as observed by Vereshchagin [146] and Bloem, De Rooij, and Adriaans [21], the universality of Kolmogorov complexity does not guarantee the invariance of sophistication up to an additive constant. Which parameterized models count as sufficient parameterized statistics is sensitive to the chosen encoding of decision procedures. Specifically, for technical reasons, the sufficiency criterion in traditional algorithmic statistics involves an additive slack-constant. A change in the encoding of decision procedures can make some models sufficient statistics that were not sufficient statistics before, and vice versa. Thus, the encoding of decision procedures affects the ensuing measure of sophistication.

3.4.3 Stochasticity

If a string x is random, then all bits of x contribute to the information in x. By a straightforward counting argument, we know that random strings exist [100]. A related question that is not as easily answered, is whether or not strings exist of which all bits contribute to *useful* information. It was shown by Shen [137] that for each n, there is a string of length n of which the traditional sophistication is at least $\frac{n}{2} - \mathcal{O}(\log n)$. Later, Gács, Tromp, and Vitányi [62] improved this bound to $n - \mathcal{O}(\log n)$. Thus, in traditional algorithmic statistics it is indeed possible for most of the n bits to count toward useful information. This is fairly surprising, because the traditional sophistication of a random string can be bounded from above by a constant. For every random string, the set $\mathbb{2}^+$ is a sufficient statistic with respect to the parameterization defined by (3.8).

A difference between parameterized algorithmic statistics and the traditional form can be seen in the treatment of random strings. While the traditional sophistication of a random string is very low, the parameterized sophistication with respect to an informative parameterization η is not. This is a consequence of Theorem 3.3.20 and the fact that the minimization function μ_η lower bounds the sophistication soph_η. In other words, what is traditionally considered noise, may become information in the context of a parameterization. We take this as further evidence supporting Slogan 3.4.3: useful information is context-dependent.

That random strings may contain a substantial amount of useful information violates one of the original intuitions about what useful information is. A common perception is that neither strings of minimum Kolmogorov complexity, nor strings of maximum Kolmogorov complexity can be rich in useful information [2]. Useful information, it is often thought, should strike a balance between these two extremes [149, 2]. Challenging this intuition is the goal of lossy data compression, which is to extract from a string its useful information. When the length of a string is divided into useful information, noise, and redundancy, as in Figure 3.5 on page 145, we observe that useful information is necessarily incompressible. In that sense, useful information is random. Therefore, the output of a lossy compression routine is ideally both high in useful information and high in Kolmogorov complexity. Of course, lossy compression is specific to a certain domain, such as images. Without knowing this domain, a compressed string may not appear particularly informative. More generally, contrary to Kolmogorov complexity, a notion of length is wholly determined by the chosen method of encoding. Since randomness revolves around a comparison between Kolmogorov complexity and length, randomness too is dependent on the chosen method of encoding. This matters, because if there is any structure to the objects in a given domain, there is no longer a unique way of encoding those objects.

3.4.15. Slogan. *There is no such thing as a random object unless the object is a string and nothing but a string.*

Sophistication focuses on sufficient statistics. Sufficiency of a parameterized model is a robust concept and no hidden additive constants are involved. Still, we may want to take parameterized models that are *nearly* sufficient into account in our analysis of the information in a string. For this, we measure how far away a parameterized model is from being a sufficient parameterized statistic [147].

3.4.16. Definition. The *parameterized optimality deficiency* of a string x in a parameterized model k with respect to a parameterization η is

$$\delta_\eta(x,k) = |\langle k, \mathrm{rank}(x : \eta_k)\rangle| - \mathrm{pc}_\eta(x).$$

When k is not a parameterized model for x, we set $\delta_\eta(x, k) = \infty$.

Note that, for every parameterized model k, most strings in η_k have a parameterized optimality deficiency not much greater than $|k|$. More precisely, for all constants n and c, there are roughly 2^n strings x with $|\langle k, \operatorname{rank}(x : \eta_k)\rangle| = |k| + n$, while there are at most 2^{n-c+1} that satisfy $\operatorname{pc}_\eta(x) \leq n - c$. In other words, at most a 2^{-c+1}-fraction of strings satisfy $\delta_\eta(x, k) \leq |k| + c$.

Additionally, observe that the parameterized optimality deficiency of x is 0 precisely when k is a sufficient parameterized statistic for x. On the basis of optimality deficiency, we can express whether a string has parameterized models of a given length that meet a required level of sufficiency.

3.4.17. Definition. Let α and β be natural numbers and η a parameterization. A string x is *(α, β)-η-stochastic* if there is an η-model k for x satisfying

$$|k| \leq \alpha \quad \text{and} \quad \delta_\eta(x, k) \leq \beta.$$

The purpose of the numbers α and β is perhaps best explained through an example.

3.4.18. Example. The number 1729 enjoys some fame for being very much not a dull number. It is known as the *Hardy–Ramanujan number*, after a visit from Godfrey H. Hardy to Srinivasa Ramanujan in which the number was discussed. At first, it may seem that the number does not have any striking features. It did so to Hardy, anyway. To make precise what this means, we need something like a parameterized model class η of number theoretic properties. We may expect that, when k_{odd} is the η-model of odd numbers, the number 1729 is $(|k_{\text{odd}}|, \beta)$-$\eta$-stochastic for some small value of β. That is, we may expect that there is not much more to 1729 than it being odd. However, 1729 is the 261st product of three distinct primes. While we have not pinned down η exactly, we may expect that being the product of three distinct primes is a fairly simple property. As 1729 is only the 865th odd number, the parameterized optimality deficiency of 1729 in k_{odd} may not be so small after all. Instead, we are better off modeling 1729 with the η-model of products of three distinct primes, $k_{\text{3-primes}}$. Still, we may not have that 1729 is $(|k_{\text{3-primes}}|, \beta)$-$\eta$-stochastic for any too small value of β. This is because 1729 is also the first number that is the sum of two cubes of positive numbers in two different ways. Thus 1729 is the first member of the model of numbers that are the sum of two cubes of positive numbers in multiple ways, $k_{\text{2-cubes}}$. While this last model is the most complex, it is quite likely that it determines the sophistication of 1729. In summary, we have three η-models for 1729, satisfying

$$|k_{\text{odd}}| \leq |k_{\text{3-primes}}| \leq |k_{\text{2-cubes}}|,$$

and

$$\delta_\eta(1729, k_{\text{2-cubes}}) \leq \delta_\eta(1729, k_{\text{3-primes}}) \leq \delta_\eta(1729, k_{\text{odd}}).$$

Traditionally, stochasticity is not defined using optimality deficiency, but using a slightly different deficiency measure known as *randomness deficiency* [137, 100]. These two deficiency measures can differ. For traditional algorithmic statistics, Vereshchagin and Shen [147, Theorem 2] have shown that both measures lead to essentially the same notion of stochasticity. The same holds true for restricted model classes, thanks to a result of Vereshchagin and Vitányi [145] [see also 147]. Because it fits our framework better, we therefore prefer optimality deficiency over randomness deficiency.

Let us take a look at the stochasticity of a string x in relation to a parameterization η. There is a frontier of values of α and β beyond which x is (α, β)-η-stochastic [148, 62]. Suppose that α and β are so that x is (α, β)-η-stochastic. There is then an η-model of complexity at most α that is within β of being a sufficient parameterized statistic for x. Sometimes, relaxing the sufficiency requirement slightly means that simple models become available. Thus, a string x may be (α, β)-η-stochastic for values of α and β that are reasonably small in relation to its length $|x|$. Such a string x is informally said to be *stochastic* [147]. Indeed, a small value of α reflects a high prior probability of the model in accordance with our discussion on page 157. Also, as noted following the definition of parameterized optimality deficiency, Definition 3.4.16, a small value of β suffices for most strings.

3.4.19. Example. In Example 3.4.1, we encountered the JPEG lossy image compression format. The JPEG compression procedure contains a tuning parameter that determines the amount of degradation in quality that is considered admissible. This tuning parameter can range in value from 1 to 100, where 100 represents near-perfect reproduction of the original image. Unfortunately, there is no unique minimal value of this parameter for which the degradation in image quality is unnoticeable to some reference observer. In other words, there is no clear-cut quality value beyond which a JPEG compressed image is a sufficient statistic for the original image. For Example 3.4.1, a quality value of 90 was used.

With a lower quality setting, the JPEG compression procedure is able to produce smaller files. The compressed file size can be thought of as the size of a model for the original image. Thus, the α component of stochasticity acts like a bound on the file size. Because there is no proper sufficiency criterion for JPEG compression, the quality setting cannot directly be related to the β component of stochasticity. However, there is a conceptual connection between lowering the quality setting and increasing the value of β.

We are used to seeing a substantial decrease in file size when we allow the image quality to degrade only slightly. In Figure 3.6, this can indeed be observed. At high values of the quality parameter, small changes have a large impact on the resulting file size.

The traditional notion of stochasticity is a special case of our parameterized one. A string is called (α, β)-*stochastic*, without mention of any η, if it is stochastic

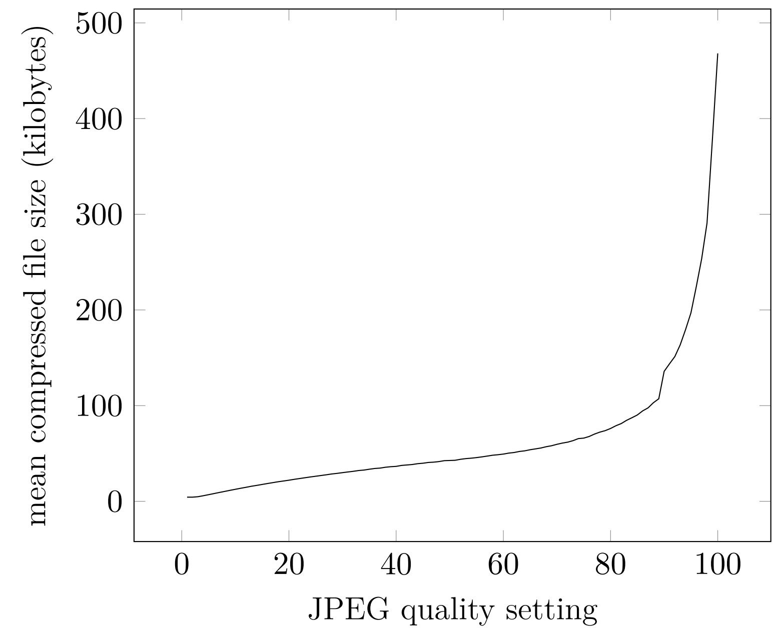


Figure 3.6: The mean file size of the images in the Kodak True Color Image Suite [61] after JPEG compression at different quality settings.

in relation to the parameterization given by (3.8). Not all strings are stochastic. In fact, some of the strings of which the sophistication is close to the length are not stochastic at all. Extending a similar result by Shen [137], Gács, Tromp, and Vitányi [62] showed that some strings are only (α, β)-stochastic when $\alpha + \beta$ nearly exhausts the length of the string. More specifically, for each n, there is a string of length n that is only (α, β)-stochastic when $\alpha+\beta$ is at least $n-\mathcal{O}(\log n)$. This result about the existence of *non-stochastic* strings can be extended to the parameterized form of stochasticity. If, given a parameterization η, a string of length n is only (α, β)-η-stochastic when $\alpha + \beta$ is at least $n - \mathcal{O}(\log n)$, then we say that the string is *non-η-stochastic*. The following theorem relates the parameterized notion of stochasticity to the traditional one.

3.4.20. Theorem. *Let x be a string and η a decidable parameterization. Denote by $\mathrm{K}(\eta)$ the minimum length of a decision procedure for η. Furthermore, define $\gamma = \mathrm{pc}_\eta(x) - \mathrm{K}(x)$, meaning that η captures the complexity of x if γ is equal to 0. If x is (α, β)-η-stochastic, then it is $(\alpha + \mathrm{K}(\eta), \beta + \gamma + \mathrm{K}(\eta))$-stochastic.*

Proof:
Let k be an η-model for x witnessing that x is (α, β)-η-stochastic and let j be the encoding of a decision procedure for η_k of minimum length. Recall from Example 3.4.9 that parameterized complexity in the traditional setting equals Kolmogorov complexity and observe that we have

$$|j| \leq |k| + \mathrm{K}(\eta) \leq \alpha + \mathrm{K}(\eta).$$

It remains to show that the optimality deficiency of x in k with respect to η is at most $\beta + \gamma + \mathrm{K}(\eta)$. The first step to this end follows the same reasoning as we have just encountered,

$$\begin{aligned}\delta(x, j) &= |\langle j, \mathrm{rank}(x : \eta_k)\rangle| - \mathrm{K}(x)\\ &\leq |\langle k, \mathrm{rank}(x : \eta_k)\rangle| + \mathrm{K}(\eta) - \mathrm{K}(x)\end{aligned}$$

which can then be expanded as

$$\begin{aligned}&= |\langle k, \mathrm{rank}(x : \eta_k)\rangle| + \mathrm{K}(\eta) - \mathrm{pc}_\eta(x) + \gamma\\ &= \delta_\eta(x, k) + \gamma + \mathrm{K}(\eta)\\ &\leq \beta + \gamma + \mathrm{K}(\eta).\end{aligned}$$

Thus, η_k witnesses that x is $(\alpha + \mathrm{K}(\eta), \beta + \gamma + \mathrm{K}(\eta))$-stochastic. □

When η is held fixed, $\mathrm{K}(\eta)$ is constant and the theorem above can be seen as a translation from η-stochasticity to traditional stochasticity. The constant γ depends on x and only disappears when η captures the complexity of x. If the complexity of a non-stochastic string x is captured by a parameterization η, then x is also non-η-stochastic. This follows from the contrapositive formulation of Theorem 3.4.20.

3.4.21. Corollary. *Let x be a string and η a decidable parameterization. Furthermore, let $\gamma = \mathrm{pc}_\eta(x) - \mathrm{K}(x)$ be the amount by which the complexity of x escapes η. If x is not (α, β)-stochastic, then it is not $(\alpha - \mathrm{K}(\eta), \beta - \gamma - \mathrm{K}(\eta))$-$\eta$-stochastic either.*

The parameterization defined in (3.8) includes 2^+ as a model. Therefore, the minimization function with respect to that parameterization is bounded from above by a constant. The same is not true of the sophistication. Put differently, there is a constant α such that for every string x there is a constant β_x such that x is (α, β_x)-stochastic. This constant, β_x, cannot be bounded from above independently of x. The same cannot be said for stochasticity with respect to an arbitrary parameterization η. A string may fail to be η-stochastic not only because simple η-models are not good enough, but also because the string has no simple η-models. For all $\alpha < \mu_\eta(x)$, regardless of β, the string x is not (α, β)-η-stochastic. The sophistication of x with respect to η may even be bigger than its length, $\mathrm{soph}_\eta(x) > |x|$. For such x and η, there are values of β with which x is not even $(|x|, \beta)$-η-stochastic. Consequently, η does not capture the complexity of x and the non-η-stochasticity of x is not a consequence of Corollary 3.4.21. Indeed, there would be no reason to assume that x is non-stochastic in the traditional sense.

3.4.4 Parameter Estimation

In Lemma 3.4.10, we found that the parameterized complexity with respect to a decidable parameterization is computable. Not only does this result extend to the computability of sophistication, it also entails that optimality deficiency is computable. Therefore, for a fixed α and β, and a decidable η, the set of (α, β)-η-stochastic strings is decidable. Moreover, for an (α, β)-η-stochastic string x, a witness, k, of its stochasticity can be computed. In particular, for the case where β equals 0, we can compute an η-model k for x such that we have $|k| = \mathrm{soph}_{\eta}(x)$. Let us take a closer look at such mappings from a string x to an η-model for x. As the η-models are represented by parameter values, we can think of the model class conveyed by η as a model class indexed by parameter values. In that sense, the model class conveyed by η is a parametric model class as known from probabilistic statistics. Likewise, a mapping from strings to η-models is also a mapping from strings to parameter values. In other words, such a mapping is an *estimator*. This probabilistic interpretation of concepts in algorithmic statistics has been recognized early on [127, 150, 62]. An estimator that realizes the sophistication of its input is particularly exciting, as its output is a sufficient statistic that is as simple as can be. While computable with respect to decidable parameterizations, the computation of such an estimator may be prohibitively resource intensive. For this reason, alternative estimators are of interest.

So far, underlying our investigation has been a two-part code for describing a string x. If k is an η-model for x, then, knowing η, we can recover x from

$$\langle k, \mathrm{rank}(x : \eta_k) \rangle. \tag{3.11}$$

We can minimize several quantities related to this two-part specification and each defines a different estimator.

The simplest model for x is one that minimizes the length of the first part of the specification, $|k|$. The corresponding k can be called the *Occam's razor* estimate, and its length equals $\mu_{\eta}(x)$. In traditional algorithmic statistics, this estimator is of no interest because the complexity of the simplest model for a string cannot exceed that of 2^{+}. However, in parameterized algorithmic statistics, 2^{+} need not occur as a slice in any specific parameterization.

Rather than minimizing the length of the first part of the specification, we could also minimize the length of the second part, $|\mathrm{rank}(x : \eta_k)|$. The corresponding η-model k is one that contains as much information about x as possible, and is known as the *maximum likelihood* estimate [148]. In traditional algorithmic statistics, this estimator would yield a model, a finite set, of which the input string is the first element. This is a form of overfitting, since all information about x now resides in the model part of the code, and none is left for the data-to-model part. Again, in parameterized algorithmic statistics, a model like this need not occur as a slice in any given parameterization.

Minimizing the total length of the two-part specification of x, we recover the *minimum description length* estimator. Equivalently, this estimator minimizes the sum of the lengths of both parts and as such balances the previous two estimators [148, 127]. A parameter value obtained by this estimator, an estimate, is a sufficient statistic. When we restrict to the simplest such models, we find that the length of a minimum description length estimate k is the sophistication $\mathrm{soph}_\eta(x)$.

In summary, some natural estimators derived from the two-part description of a string, and their minimization objectives, are the following.

estimator	*minimizes*	*length*
Occam's razor (OR)	first part	μ_η
maximum likelihood (ML)	second part	[no name]
minimum description length (MDL)	sum	soph_η

The parameterization defined by (3.8) is not decidable and traditional algorithmic statistics pays attention to the computability of estimators. One approach to the computability of estimators is to place an upper bound on the model complexity and consider each estimator as a function of this bound [62, 148] [in the presence of resource bounds, 108]. Specifically, we consider ML and MDL relative to a selection of the slices of (3.8). The selection is obtained by including only those models of which the corresponding parameter value has a length shorter than the anticipated bound. As mentioned before, OR is not interesting in the traditional context because it can be bounded from above by a constant. On the other hand, ML and MDL are interesting, but, while the length of their estimates can be approximated from above, they are not computable [148]. Of the two estimators, MDL is easier to approximate and therefore more convenient to work with in practice [148, Section V.B]. Additionally, as a function of a bound on the model complexity, MDL selects a sufficient statistic precisely when ML does [148, Lemma IV.2]. With respect to decidable parameterizations, OR, ML, and MDL would all be computable as a function of a bound on the length of parameter values. Note, though, that without such a bound ML need not be computable. Unfortunately, in the parameterized setting MDL cannot simply be used as a substitute for ML. The tight correspondence between the two in traditional algorithmic statistics is not a given in the parameterized setting.

There are many more estimators than the three we have looked at so far, OR, ML, and MDL. For a parameterization η, we may look at those estimators that are in some sense consistent with η.

3.4.22. Definition. A function $\kappa\colon 2^+ \to 2^+$ is a *parameter estimator* for a parameterization η if it is consistent with η in the sense that, for all strings x, we have $x \in \eta_{\kappa(x)}$

3.4.23. Example. Suppose κ is a parameterization in the Flum and Grohe framework for parameterized complexity theory. On page 90, we associated with this function the parameterization

$$\eta = (\{x \mid \operatorname{asInt}(\kappa(x)) \leq \operatorname{asInt}(k)\})_{k\in 2^+}.$$

For this parameterization, κ is a polytime-computable parameter estimator.

We have seen that when η is decidable, the parameter estimators OR and MDL are computable. Yet, even when η is not decidable, a parameter estimator for η may be computable. For instance, if κ maps all strings to an encoding of a decision procedure for 2^+, it is a computable parameter estimator for (3.8).

3.4.24. Example (continued from Example 3.2.13). Let A be a p-cylinder and $g\colon 2^+ \to 2^+ \times 2^+$ a corresponding isomorphism. Recall that we denote by g_1 the first component of the image of g and that the parameterization based on g was defined as

$$\eta = (\{x \mid \operatorname{asInt}(g_1(x)) \leq \operatorname{asInt}(k)\})_{k\in 2^+}.$$

For this parameterization, g_1 is a parameter estimator. As g_1 is computable in polynomial time, it is also a parameterization in the sense of Flum and Grohe. Moreover, we have, for all x,

$$\mu_\eta(x) = \operatorname{soph}_\eta(x) = |g_1(x)|.$$

Note that (3.9) holds for η, so we find that OR, ML, MDL, and g_1 are all the same parameter estimator for η.

As we have seen, a parameter estimator need not produce a sufficient parameterized statistic for its input. However, we can get a pretty decent upper bound on the parameterized optimality deficiency of the estimates produced by a computable parameter estimator. For this, we require two bounds involving the Kolmogorov complexity of a string x. The first of these is a straightforward extension of (3.10) to parameterized complexity,

$$\mathrm{K}(x) \leq \operatorname{pc}_\eta(x) + \mathrm{K}(\eta). \tag{3.12}$$

If we combine this inequality with a lower bound on the Kolmogorov complexity of x, we get a lower bound on the parameterized complexity.

A lower bound on the Kolmogorov complexity of a string can be found using a computable parameter estimator κ. Since κ is computable, for every parameter value k, the inverse $\kappa^{-1}(k) = \{x \mid \kappa(x) = k\}$ is a decidable set. Therefore, letting $\mathrm{K}(\kappa)$ denote the minimum length of a procedure for computing κ, we have, for every string x and with $k = \kappa(x)$,

$$\mathrm{K}(\langle k, \operatorname{rank}(x : \kappa^{-1}(k))\rangle) \leq \mathrm{K}(x) + \mathrm{K}(\kappa). \tag{3.13}$$

Specifically, using κ we can compute k and $\operatorname{rank}(x : \kappa^{-1}(k))$ from x, so the complexity of those values together is at most that of κ plus that of x.

These two inequalities, (3.12) and (3.13), allow us to upper bound the parameterized optimality deficiency of estimates produced by a computable parameter estimator. Before we go into more detail, it should be noted that $\kappa^{-1}(k)$ need not be a slice of the parameterization η. However, this is mainly a technical concern as we can extend η to include these inverses for all parameter values. Given η and a parameter estimator κ for η, let η' be the parameterization defined by

$$\eta'_{k'} = \begin{cases} \eta_k & \text{if } k' = \mathtt{0}k, \\ \kappa^{-1}(k) & \text{if } k' = \mathtt{1}k. \end{cases}$$

If η is decidable and κ is computable, then η' is decidable too. Moreover, it is in the same equivalence class as η in the uniform order on parameterizations defined in Definition 3.2.20. For completeness, we lift the parameter estimator κ to a parameter estimator κ' defined by $\kappa'(x) = \mathtt{1}\kappa(x)$. This turns a computable parameter estimator κ for η into a computable parameter estimator κ' for η' in a way that gives us, for any string x,

$$\eta'_{\kappa'(x)} = \kappa'^{-1}(\kappa'(x)).$$

Let us say that a parameter estimator κ for a parameterization η that, for all x, satisfies $\eta_{\kappa(x)} = \kappa^{-1}(\kappa(x))$ is a *slice-estimator.* We then get the following theorem that allows us to bound the parameterized optimality deficiency of the estimates provided by a computable slice-estimator. More broadly, this theorem can be used to identify, though not by any effective means, η-stochastic strings. Incidentally, we obtain an improvement on our observation, following Definition 3.4.16, regarding the density of strings with a high optimality deficiency.

3.4.25. Theorem. *Let η be a decidable parameterization, and κ a computable slice-estimator for η. There is a constant c such that, for every string x, when we set*

$$\begin{aligned} k &= \kappa(x), \\ i &= \operatorname{rank}(x : \kappa^{-1}(k)), \\ \beta &= |\langle k, i\rangle| - \mathrm{K}(\langle k, i\rangle) + c, \end{aligned}$$

we find that x is $(|k|, \beta)$-η-stochastic.

Proof:
The claimed parameterized stochasticity is witnessed by the slice $\eta_k = \kappa^{-1}(k)$. To see that the parameterized optimality deficiency of x with respect to η is indeed at most β, consider

$$\begin{aligned} \delta_\eta(x, k) &= |\langle k, \operatorname{rank}(x : \eta_k)\rangle| - \mathrm{pc}_\eta(x) \\ &= |\langle k, i\rangle| - \mathrm{pc}_\eta(x) \end{aligned}$$

which, by (3.12) and (3.13), satisfies

$$\begin{aligned}&\leq |\langle k,i\rangle| - (\mathrm{K}(x) - \mathrm{K}(\eta))\\ &\leq |\langle k,i\rangle| - \mathrm{K}(\langle k,i\rangle) + \mathrm{K}(\eta) + \mathrm{K}(\kappa).\end{aligned}$$

As $\mathrm{K}(\eta)$ and $\mathrm{K}(\kappa)$ are independent of x, we may set $c = \mathrm{K}(\eta) + \mathrm{K}(\kappa)$, with which the last line becomes equal to β and the bound is proven. □

Of course, a string should only be called η-stochastic if it has a simple model with a low optimality deficiency. To see that Theorem 3.4.25 can truly be used to identify η-stochastic strings, we first take a closer look at the optimality deficiency. Observe that $|\langle k,i\rangle|$ can be arbitrarily large, while $\mathrm{K}(\langle k,i\rangle)$ is still small. More importantly, though, is the observation that the two quantities can also be close to each other. In fact, for any constant c and any sufficiently large β, there are infinitely many values of k and i such that we have

$$|\langle k,i\rangle| - \mathrm{K}(\langle k,i\rangle) + c \leq \beta.$$

The incompressibility theorem for prefix complexity [100, Theorem 3.3.1(ii)] tells us that the above inequality must hold for a large fraction of possible values of k and i. Specifically, for each fixed constant r, of all $\langle k,i\rangle$ of length n, at most $2^{n-r+\mathcal{O}(1)}$ satisfy $\mathrm{K}(\langle k,i\rangle) \leq |\langle k,i\rangle| - r$, or, equivalently, $r \leq |\langle k,i\rangle| - \mathrm{K}(\langle k,i\rangle)$. Moreover, this is still true of all the possible values of i after we fix a value of k [100, Theorem 3.9.1]. Thus, we can turn Theorem 3.4.25 into a statement about the prevalence of stochastic strings. A similar statement regarding the traditional notion of stochasticity was proven by Shen [137, Theorem 4].

3.4.26. Corollary. *Let k be an η-model in the range of κ such that η_k is an infinite set. For all β, among the first m elements of η_k at least $(1-2^{-c})m$ are $(|k|,\beta)$-η-stochastic, where c equals $\beta - \mathcal{O}(1)$ with the $\mathcal{O}(1)$-term depending on η, κ, and k.*

Note that, for a fixed value of β, the dependence on k of the hidden constant is more specifically a dependence on $|k| - \mathrm{K}(k)$. Because of that, the corollary is especially powerful when the range of κ contains an infinite random set of parameter values belonging to infinite η-models. Here, randomness of the set is meant as defined in Definition 3.3.16. In that case, the same density of strings with an optimality deficiency bounded by β is achieved on infinitely many η-models. Informally, disregarding the density considerations, the corollary can then be phrased in a way that somewhat resembles the instance complexity conjecture: Infinitely many η-models are a near-sufficient parameterized statistic infinitely often. Sufficiency is a way of saying that all information in a model is useful information and none of the information is not useful. We thus get a more informal way of phrasing the corollary.

3.4.27. Slogan. *Infinitely many parameter values reflect precisely all useful information of infinitely many strings.*

This is true for any η for which there is a computable slice-estimator of which the range includes an infinite random set of infinite η-models. Because of Theorem 3.4.20, the statement remains true if we replace our parameterized notion of sufficiency with the traditional one. This traditional notion involves Kolmogorov complexity, therefore Corollary 3.4.26 complements Corollary 3.3.7.

Moreover, a closer look at the proof of Theorem 3.4.25 directs us to a set of strings of which η captures the complexity. Let η, x, k, and i be as in the statement of Theorem 3.4.25. By definition of parameterized complexity, we find $\mathrm{pc}_\eta(x) \leq |\langle k, i\rangle|$. Now, if $\langle k, i\rangle$ is a random string in the sense that, up to an additive constant, we have $|\langle k, i\rangle| = \mathrm{K}(\langle k, i\rangle)$, then we get $\mathrm{pc}_\eta(x) \leq \mathrm{K}(\langle k, i\rangle)$. By (3.13), we then also get $\mathrm{pc}_\eta(x) \leq \mathrm{K}(x) + \mathrm{K}(\kappa)$. Finally, combined with (3.12), we get

$$\mathrm{K}(x) - \mathrm{K}(\eta) \leq \mathrm{pc}_\eta(x) \leq \mathrm{K}(x) + \mathrm{K}(\kappa),$$

showing that η captures the complexity of x. As noted leading up to Corollary 3.4.26, there are infinitely many random strings $\langle k, i\rangle$ and the corresponding strings x are η-stochastic. We point out that if η is informative in the sense of Definition 3.3.19, then our set of strings of which η captures the complexity is not a random set. Likewise, if the slices of η are sparse, then most of these strings are highly non-random.

3.4.28. Example (continued from Example 3.4.24). By capturing the complexity of a string, a parameterization isolates the redundancy in the description of a string into the second part of a two-part code. Originally, however, parameterized complexity theory did not target algorithmic complexity. Instead, it revolved around isolating not descriptive redundancy, but computational redundancy associated with a string in light of a decision problem. It is worthwhile to note that a parameterization can achieve both these feats simultaneously.

Let A be a p-cylinder, $g\colon 2^+ \to 2^+ \times 2^+$ a corresponding isomorphism and η the parameterization based on g. All slices of η are infinite and the range of g_1 is the entire set 2^+. Because of this, it follows from Corollary 3.4.26 that infinitely many slices of η contain infinitely many strings of which η captures the complexity. For those strings, η acts as a measure of both algorithmic and computational complexity.

A parameterization can witness that computational and algorithmic complexity may coincide infinitely often. However, this does not provide us with a dual to Theorem 3.3.20, which states that random instances are hard. While some strings of low algorithmic complexity may appear in a parameterization with short parameter values, this need not be true of all such strings. In order to control the expected minimum length of a parameter value for a given string, we need to tinker with the distribution we use to sample the strings.

3.4.5 Encodings and Distributions

At the end of Section 3.4.3, several possibilities were discussed for why a string may fail to be stochastic with respect to a parameterization η. While any of these may be used to prove that non-η-stochastic strings exist, they tell us very little about how likely we are to encounter such strings. The existence of non-stochastic strings in the traditional sense is described by Vereshchagin and Shen as a “mathematical existence result; it does not say anything about the possibility to observe non-stochastic objects in the ‘real world’ ”. In support of this sentiment, they show that any effective way of sampling from 2^+ may produce a non-stochastic string only with a negligible probability [109, 147]. This result makes use of the universal probability distribution, which is the distribution related to Kolmogorov complexity via the Kraft inequality. A similar result revolving around the cardinality of the set of non-stochastic strings goes back to Shen [137, Theorem 3].

For parameterized algorithmic statistics, a bound on the cardinality of the set of non-η-stochastic strings is available in the form of Corollary 3.4.26. However, we are not so much interested in how many non-η-stochastic strings there are, as in how likely we are to encounter such strings in the ‘real world’. For this, we need a reference distribution on the strings and the universal distribution may not be our best option. In our framework, we use the slices of parameterizations as a substitute for the decidable sets. Hence, our reference measure of algorithmic complexity should be parameterized complexity instead of Kolmogorov complexity. Of course, parameterized complexity can be equal to Kolmogorov complexity, such as is the case with respect to the parameterization given by (3.5). In that case, the resulting distribution equals the universal distribution. Since this parameterization is not in $\mathscr{L}_{\mathbf{FPT}}$, there may be more realistic distributions to work with. In line with Slogan 3.4.15, this implies that the method of encoding objects matters, as only the universal distribution is encoding-invariant. We shall explore distributions derived from parameterized complexity, and their effect on the parameter values we may expect for strings of a given length.

The Uniform Distribution

For a start, if we assume all strings of length n are equally likely, the effects of Corollary 3.4.26 are undone by Theorem 3.3.20. The random strings alone push the expected minimum length of a parameter value to within a multiplicative constant of its maximum value. Note that there is a slight abuse of notation in the following theorem. To turn the uniform distribution on strings of a given length into a distribution on all strings, we need a prior distribution on the lengths of strings. However, this prior distribution is canceled out again when we take an expected value conditional on the length of a string. For this reason, we can leave the prior distribution implicit and still speak of a ‘uniform’ distribution.

In actuality, we are dealing with a distribution on strings that is only uniform conditional on the length of a string.

3.4.29. Theorem. *Let η be a parameterization that is informative in the sense of Definition 3.3.19, and consider the expected value,* $\mathrm{E}_{\mathrm{unif}}(\mu_\eta(x) \mid |x|)$*, of* $\mu_\eta(x)$*, conditional on the length of x, under the uniform distribution on strings of a given length. There is a constant c such that, for all n, we have*

$$\underset{\mathrm{unif}}{\mathrm{E}}(\mu_\eta(x) \mid |x| = n) \geq \frac{\mathrm{M}_\eta(n)}{c} - c.$$

Proof:
Pick a constant r, fix an arbitrary value for n, and let X be the set

$$\{x \in 2^n \mid \mathrm{K}(x) \geq n + \mathrm{K}(n) - r\}$$

that is random in accordance with Definition 3.3.16. By Theorem 3.3.20, there is a constant h, independent of n, such that all $x \in X$ satisfy $\mu_\eta(x) \geq \mathrm{M}_\eta(n) - h$. Further, the incompressibility theorem [100] tells us that X contains at least $2^n(1 - 2^{-r+\mathcal{O}(1)})$ strings, where we may assume that r is larger than the constant represented by $\mathcal{O}(1)$. Together, these observations make the following derivation possible.

$$\begin{aligned}
\underset{\mathrm{unif}}{\mathrm{E}}(\mu_\eta(x) \mid |x| = n) &= \sum_{x \in 2^n} \mu_\eta(x) \cdot 2^{-n} \\
&\geq \sum_{x \in X} \mu_\eta(x) \cdot 2^{-n} \\
&\geq \sum_{x \in X} (\mathrm{M}_\eta(n) - h) \cdot 2^{-n} \\
&\geq 2^n \cdot (1 - 2^{-r+\mathcal{O}(1)}) \cdot (\mathrm{M}_\eta(n) - h) \cdot 2^{-n} \\
&\geq (1 - 2^{-r+\mathcal{O}(1)}) \cdot \mathrm{M}_\eta(n) - h.
\end{aligned}$$

Now, choosing $c = \max\left\{\frac{1}{1-2^{-r+\mathcal{O}(1)}}, h\right\}$, we get the statement of the theorem. □

For parameterizations encountered in application-oriented parameterized complexity theory, we often have $\mathrm{M}_\eta(n) \in \mathcal{O}(\log n)$. This is so because commonly the least numeric parameter value $\mathrm{asInt}(k)$ with which a string x is in η_k can be upper bounded by a polynomial in $n = |x|$. The vertex cover parameterization considered in Example 3.3.24 is an example of such a parameterization. Moreover, this bound on M_η is usually tight, meaning that there are constants c_1 and c_2 such that, for all n, we have $c_1 \log n \leq \mathrm{M}_\eta(n) \leq c_2 \log n$. Theorem 3.4.29 then tells us that the same is true of the expected value of $\mu_\eta(x)$. This translates to the fact that $|x|$ can be bounded from above by a polynomial in the expected value of $\mathrm{asInt}(k)$. Now, suppose ϕ is a parameterized procedure with a running time of

$f(\text{asInt}(k))|x|^{\mathcal{O}(1)}$, for some convex function f. This is the kind of running time associated with fixed-parameter tractability. By Jensen's inequality [36], we find $\mathrm{E}(f(\text{asInt}(k)) \mid |x|) \geq f(\mathrm{E}(\text{asInt}(k) \mid |x|))$. Consequently, the expected running time of ϕ, assuming a uniform distribution on the strings of a given length, is comparable to the worst-case running time. For fixed-parameter tractable problems outside $\mathbf{P}$, this would mean that, with the uniform distribution, we should expect instances to be intractable. However, fixed-parameter tractable problems are found to be tractable in practice. From this, we conclude that the uniform distribution is not a good model of the 'real world'. Hence, we should take parameter values into account in the distributions we consider.

The Random–Hard Distribution

Increasing the role of the parameter value, we can use the encoding that we encountered in the proof of Theorem 3.3.20 as the basis of a distribution. As we shall only be concerned with the distribution for a fixed parameterization η, we omit the specification of η from the encoding. This leaves us with a prefix-free multi-part encoding that, for a string x of length n, consists of the following parts, in order:

1. $\mathrm{K}(n)$ bits specifying n,
2. $\mathrm{M}_\eta(n) + 1$ bits specifying a parameter value k such that we have $x \in \eta_k$,
3. $\log \mathrm{N}_\eta(n, k)$ bits specifying $\text{rank}(x : 2^n \cap \eta_k)$.

Observe that this encoding is prefix-free because the length of each part is determined by the values encoded in the parts preceding it. Indeed, the first part uses a prefix-free encoding to begin with. Effectively, by our encoding of the first part, we assume a universal prior distribution on the length of the strings we encode. Because our multi-part encoding is prefix-free, we can use the Kraft inequality to derive a probability distribution from it. In light of Theorem 3.3.20, we shall refer to this probability distribution as the *random–hard* distribution.

The above multi-part encoding may allow for several different encodings of a single string x. This is because there may be multiple parameter values k, each of length at most $\mathrm{M}_\eta(|x|)$, such that x is included in η_k. Later on, we shall also look at distributions that allow for arbitrary parameter values, not just those with a length of at most $\mathrm{M}_\eta(|x|)$. Here, we shall take all possible encodings in accordance with the above encoding into account. Specifically, we define our distribution so that the probability of a string x is the same as that of arriving at a description of x by tossing a fair coin. For any string x, let S_x be the set of possible parameter values with which x can be encoded, namely

$$S_x = \{k \mid |k| \leq \mathrm{M}_\eta(x) \ \text{ and } \ x \in \eta_k\}.$$

Using this set, the probability of a string x in accordance with our encoding can be expressed as

$$\begin{aligned}\underset{\text{r-h}}{\mathrm{P}}(x) &= \sum_{k \in S_x} 2^{-(\mathrm{K}(|x|)+\mathrm{M}_\eta(|x|)+1+\log \mathrm{N}_\eta(|x|,k))} \\ &= \frac{1}{2^{\mathrm{K}(|x|)+\mathrm{M}_\eta(|x|)+1}} \sum_{k \in S_x} \frac{1}{\mathrm{N}_\eta(|x|,k)}\end{aligned}$$

We obtain the probability conditional on the length of the string by simply dropping the specification of the length, $\mathrm{K}(|x|)$. Thus, for all n and all x of length n, we have

$$\underset{\text{r-h}}{\mathrm{P}}(x \mid |x| = n) = \frac{1}{2^{\mathrm{M}_\eta(n)+1}} \sum_{k \in S_x} \frac{1}{\mathrm{N}_\eta(n,k)}$$

The terms of the sum on the right side of the equation above represent uniform distributions on the sets $2^n \cap \eta_k$. These uniform distributions are combined for different parameter values, where each parameter value k with $|k| \le \mathrm{M}_\eta(n)$ is assigned a probability of $\frac{1}{2^{\mathrm{M}_\eta(n)+1}}$. The random–hard probability distribution is thus the marginal distribution of the strings x as part of a combined distribution that includes parameter values. For every n, all strings of length n get a nonzero probability, because parameter values up to a length of $\mathrm{M}_\eta(n)$ are taken into account. Our distribution assigns a higher probability to strings that occur in more slices as well as to strings that occur in slices that contain fewer elements.

In Theorem 3.4.29, we computed the expected minimum length of a parameter value, $\mu_\eta(x)$, under the uniform distribution. Because the random–hard distribution is a marginal distribution, it is not straightforward to compute the expected value of $\mu_\eta(x)$ under this distribution. Therefore, we shall initially consider the expected length of a parameter value, $|k|$. Unfortunately, the expected parameter value length is essentially its maximum value. This suggests that the random–hard distribution is also not a good model of the 'real world'.

3.4.30. Theorem. *Let η be a parameterization and consider the expected length of a parameter value $k \in S_x$, conditional on the length of x, under the random–hard probability distribution. For all n, we have*

$$\underset{\text{r-h}}{\mathrm{E}}(|k| \mid |x| = n) \ge \mathrm{M}_\eta(n) - 1.$$

Proof:
By changing the summation order in the computation of the expected value, we

get rid of anything specific to the string x,

$$\begin{aligned}
\mathop{\mathrm{E}}_{\text{r-h}}(|k| \mid |x| = n) &= \sum_{x \in 2^n} \sum_{k \in S_x} |k| \frac{1}{2^{\mathrm{M}_\eta(n)+1}\, \mathrm{N}_\eta(n,k)} \\
&= \sum_{k \text{ with } |k| \leq \mathrm{M}_\eta(n)} \; \sum_{x \in 2^n \cap \eta_k} |k| \frac{1}{2^{\mathrm{M}_\eta(n)+1}\, \mathrm{N}_\eta(n,k)} \\
&= \sum_{k,\ |k| \leq \mathrm{M}_\eta(n)} |k| \frac{1}{2^{\mathrm{M}_\eta(n)+1}\, \mathrm{N}_\eta(n,k)} \sum_{x \in 2^n \cap \eta_k} 1 \\
&= \sum_{k,\ |k| \leq \mathrm{M}_\eta(n)} |k| \frac{1}{2^{\mathrm{M}_\eta(n)+1}\, \mathrm{N}_\eta(n,k)} \mathrm{N}_\eta(n) \\
&= \frac{1}{2^{\mathrm{M}_\eta(n)+1}} \sum_{k,\ |k| \leq \mathrm{M}_\eta(n)} |k|.
\end{aligned}$$

In the last expression, we recognize the expected length of a string of length at most $\mathrm{M}_\eta(n)$, where all candidate strings are equally likely. To be exact, the full expression equals $\mathrm{M}_\eta(n) - 1 + \frac{1}{2^{\mathrm{M}_\eta(n)}}$. □

It is possible to only consider encodings of a string x with respect to a specific witness of $\mu_\eta(x)$. The previous theorem would then become a statement about the expected value of $\mu_\eta(x)$. For this, consider the boundary of a parameterization η defined by

$$\eta^\partial_k = \eta_k \setminus \bigcup_{j < \mathrm{asInt}(k)} \eta_{\mathrm{asStr}(j)}.$$

A string is included in a slice k of this boundary of η only if η_k is the first slice of η to include x. Thus, the only parameter value with which a string is in the boundary of a parameterization is a shortest parameter value. Because of this, the expected value of $|k|$ coincides with the expected value of $\mu_{\eta^\partial}(x) = \mu_\eta(x)$. The boundary of a parameterization is not itself a parameterization, as it is no longer directed. However, if a parameterization η is decidable, then so is its boundary η^∂. We can use the boundary of a parameterization to define a distribution along the lines of the random–hard distribution discussed previously. With respect to this distribution, the expected value of $\mu_\eta(x)$ is thus at least $\mathrm{M}_\eta(n) - 1$, by Theorem 3.4.30. Note that we only needed to change the distribution and not the reference parameterization η, since $\mathrm{M}_{\eta^\partial}$ is identical to M_η.

Parameterized Complexity Distributions

With the distributions considered so far, non-η-stochastic strings are more prevalent than what we observe in practice. Let us turn to the two-part code (3.11) underlying parameterized complexity. This two-part code can be used as the basis of a probability distribution on strings, which we shall refer to as the *parameterized* *complexity* distribution. We do so via the Kraft inequality, which requires us to encode the two parts of (3.11) into a prefix-free set. Guided by fixed-parameter tractability, we shall characterize suitable pairing functions.

We assume that the two parts are encoded independently, each using their own prefix-free encoding. Thus, for all k and i, the number of bits required for encoding the pair $\langle k, i\rangle$ can be expressed as the sum of the lengths of the encodings of k and i. Let ℓ_1 and ℓ_2 express these lengths, so that the pair is encoded using $\ell_1(k) + \ell_2(i)$ bits. Further, let A be a set that is in **FPT** with a parameterization η and let f witness the parameter-dependence of the corresponding running time. We are interested in the expected value of this running time as we know from experience that the expected running time should be a tractable one. The general form of this expected value with respect to the parameterized complexity distribution, before specifying a specific pairing function, is

$$\begin{aligned}\mathop{\mathrm{E}}_{\mathrm{pc}}(f(k) \mid |x| = n) &= \sum_{k\in 2^+} \sum_{x\in 2^n\cap\eta_k} f(k)\cdot 2^{-|\langle k,\mathrm{rank}(x:\eta_k)\rangle|} \Big/ \sum_{x\in 2^n} \mathop{\mathrm{P}}_{\mathrm{pc}}(x)\\ &= \sum_{k\in 2^+} f(k)\cdot 2^{-\ell_1(k)} \sum_{x\in 2^n\cap\eta_k} 2^{-\ell_2(\mathrm{rank}(x:\eta_k))} \Big/ \sum_{x\in 2^n} \mathop{\mathrm{P}}_{\mathrm{pc}}(x).\end{aligned}$$

Here, the division takes care of the normalization needed because we compute the expected value conditional on the length n. We shall assume that ℓ_1, ℓ_2, and η are such that the distribution of n follows a power law. Under this assumption, $\mathrm{E}_{\mathrm{pc}}(f(k) \mid |x| = n)$ can be bounded by a polynomial in n if and only if

$$\sum_{k\in 2^+} f(k)\cdot 2^{-\ell_1(k)} \sum_{x\in 2^n\cap\eta_k} 2^{-\ell_2(\mathrm{rank}(x:\eta_k))}$$

can be bounded by a polynomial in n. In this expression, the final sum is a sum of the probabilities of the individual strings of length n in η_k, as obtained via the Kraft inequality using ℓ_2. As such, that sum is at most 1, and, up to factors polynomial in n, we find

$$\mathop{\mathrm{E}}_{\mathrm{pc}}(f(k) \mid |x| = n) \le \sum_{k\in 2^+} f(k)\cdot 2^{-\ell_1(k)}.$$

Observe that the bounding value on the right is not sensitive to the length n. Therefore, only two things can happen as a result of our choice of a pairing function. Either the above sum diverges, or it reduces to a constant independent of n. In the latter case, the expected running time of a decision procedure for A is a tractable one, as the expected value of $f(k)$ is bounded by a polynomial in n. One possibility for our pairing function is to use

$$\ell_1(k) = \mathrm{asInt}(k) + \log f(k) - \mathcal{O}(1) \tag{3.14}$$

bits, with the $\mathcal{O}(1)$-term used for normalization, for encoding a parameter value k. Depending on η, the distribution of n may still follow a power law, and we would then find, up to factors polynomial in n,

$$\mathop{\mathrm{E}}_{\mathrm{pc}}(f(k) \mid |x| = n) \leq \sum_{k \in 2^+} 2^{-\operatorname{asInt}(k)} \in \mathcal{O}(1).$$

If, for some $\varepsilon < 1$ and all but finitely many strings k, we have $\log f(k) \leq \varepsilon \operatorname{asInt}(k)$, then even a simple unary encoding of k, satisfying

$$\ell_1(k) = \operatorname{asInt}(k),$$

would suffice. Note that, in practice, the function f commonly looks something like $f(k) = 2^{\mathcal{O}(1)\cdot \operatorname{asInt}(k)}$. For example, this is the case for the parameterized procedures for Clique and VertexCover discussed in Section 1.3. With such a function f, the length prescribed by (3.14) would not be too dissimilar to a unary code.

The best-known two-part encoding that uses a unary code for its first part is Golomb coding [71, 133]. Crucially, Golomb coding differs from the coding of strings introduced here in that our coding method does not assign a unique code to every string. Similarly to Golomb coding, our coding method is suggestive of a geometric distribution on the occurrence of strings from successive η-models.

At the same time, a unary encoding of parameter values suggests that the expected value of the length, $|k|$, of an η-model k is bounded by a constant. Most importantly, this constant does not depend on the length bound $|x| = n$ conditional on which the expected value is computed. Thus, the probability of encountering string x that has no η-models of a short length in comparison to $|x|$ is vanishingly small. In that sense, the probability of observing a non-η-stochastic string is negligible if η is associated with empirical tractability.

3.4.31. Slogan. *The expected parameter value is small.*

In applications, knowledge of the distribution governing the data you are working with is important. From the success of fixed-parameter tractability in practice, we argue that in such distributions $\mathrm{E}(f(k) \mid |x| = n)$ can be bounded by a polynomial in n. Of course, this assumes the thesis by Cobham and Edmonds that feasible computation is computation with a polynomially bounded running time. We have seen that, for suitable pairing functions, the parameterized complexity probability distribution satisfies this criterion. However, recall that in practice, by Theorem 3.2.41, there is no optimal parameterization with which a set is in **FPT**. Hence, there is no best-fitting probability distribution with respect to a fixed-parameter tractable set outside **P**. Nevertheless, each parameterization with which a set is in **FPT** gives rise to a model of reality. In particular, such distributions can be used for benchmarking, and even for researching likely properties of relevant data. In this regard, distributions based on parameterized complexity offer an alternative for generic distributions of objects that are not inherently strings. For instance, the Erdős–Rényi model [51, 65, 40] outlines a distribution on graphs that is not specific to any application. This distribution may not be the most suitable when your application involves a parameterized procedure that is found to perform well in practice. The distribution associated with the parameterization underlying this procedure may be of greater interest.

3.4.32. Example. In Example 3.3.24, we found that the largest minimum vertex cover among those of graphs that have n vertices contains about n vertices. Combining Theorem 3.3.20 with the incompressibility theorem [100], this can be linked to the universal distribution. We find that, under the universal distribution, the expected size of a minimum vertex cover of an n-vertex graph is linear in n. Similarly, the expected size of a minimum vertex cover of an n-vertex graph is linear in n in the Erdős–Rényi model [152, Theorem 23].

We have seen examples of graph problems that are fixed-parameter tractable once parameterized by the minimum vertex cover size in Section 1.3. If such a parameterized approach is successful in practice, then, in line with Slogan 3.4.31, we encounter mostly graphs with small vertex covers. Thus, in that case, neither the universal distribution, nor that of the Erdős–Rényi model is a good model of our data. The parameterized complexity distribution with a unary encoding of parameter values may be a better model.

In light of the above example, the parameterized complexity probability distribution provides a distinct notion of a random, or *typical*, graph. Such a notion may be more meaningful to the application at hand than the notion provided by the Erdős–Rényi model.

3.4.6 Data Structures

Sampling from a parameterized complexity probability distribution for a parameterization η is straightforward because of the underlying two-part code, (3.11). In short, we generate a string of bits by tossing a fair coin until we arrive at a description of a pair $\langle k, i\rangle$, using the appropriate pairing function. This pair is then mapped to the ith string in η_k. By definition of the parameterized complexity distribution, this procedure generates strings according to that distribution. The procedure can alternatively be described as a random initialization of a two-part data structure. This data structure can be used instead of the "raw" representation of strings. Of course, procedures cannot make use of this data structure directly, but must be adapted. Before going into the adaptations needed, let us take a look at the benefits the two-part data structure offers.

Let η be a parameterization and x a string. At its most efficient, the two-part data structure realizes the parameterized complexity of x with respect to η. When it does, the data structure decomposes the length of x similarly to what is depicted in Figure 3.5 on page 145. Instead of the traditional measures of algorithmic complexity, however, the prevailing measure is that of algorithmic complexity in the context of η. Thus, the Kolmogorov complexity $\mathrm{K}(x)$ is replaced by $\mathrm{pc}_\eta(x)$ and the useful information is replaced by $\mathrm{soph}_\eta(x)$. As $\mathrm{pc}_\eta(x)$ could be bigger than $|x|$, we may not always identify any algorithmic redundancy. Note, though, that the difference between $\mathrm{pc}_\eta(x)$ and $|x|$ can be bounded by a function of $\mathrm{M}_\eta(|x|)$. The specifics of this bound depend on the chosen pairing function used in the definition of parameterized complexity.

For any set A, a data structure derived from a parameterization in $\mathscr{F}_{\mathbf{FPT}}(A)$ may have an especially desirable characteristic. Consider in relation to A a parameterization $\eta \in \mathscr{F}_{\mathbf{FPT}}(A)$ and a string x such that $\mu_\eta(x)$ is substantially smaller than $\mathrm{M}_\eta(|x|)$. That is, membership in A of this string x is easy to decide for the parameterized decision procedure corresponding to η. Depending on how informative η is, the data structure that corresponds to it will manage to compress x. We shall use a variant of the pairing function of Definition 2.1.2 where we reverse the components. If we let $\ell(k)$ denote the length of the Elias delta encoding of the string k, this pairing function is so that, for all k and i, we have $|\langle k, i\rangle| = \ell(k) + |i|$.

3.4.33. Theorem. *Let η be a parameterization and k a parameter value. If η satisfies the mild informativeness criterion $\mathrm{N}_\eta(n,k) \in \mathrm{o}(2^n)$ then, for every constant d, all but finitely many strings $x \in \eta_k$ satisfy*

$$|\langle k, \mathrm{rank}(x : \eta_k)\rangle| \leq |x| - d.$$

Proof:
The inequality of the theorem can equivalently be put as

$$\ell(k) + d \leq |x| - |\mathrm{asStr}(\mathrm{rank}(x : \eta_k))|.$$

As $\ell(k)$ and d are independent of x, it suffices to show that the right-hand side, $|x| - |\mathrm{rank}(x : \eta_k)|$, is almost always greater than any fixed constant. In other words, it should have an unbounded limit inferior, implying that η_k is meager. Note that the rank of any x in η_k is at most

$$\sum_{n \leq |x|} \mathrm{N}_\eta(n,k),$$

because this is the total number of strings in η_k with a length of at most that of x. As $|\mathrm{rank}(x : \eta_k)|$ is roughly equal to $\log \mathrm{rank}(x : \eta_k)$, it suffices to show that

$$|x| - \log \sum_{n \leq |x|} \mathrm{N}_\eta(n,k)$$

has an unbounded limit inferior.

Let f be a function with an unbounded limit inferior that satisfies, for all n,

$$\mathrm{N}_\eta(n,k) \leq \frac{2^n}{f(n)}.$$

Such a function exists because we have $\mathrm{N}_\eta(n,k) \in \mathrm{o}(2^n)$. We may assume that f does not grow too fast and also satisfies $f(n) \leq \frac{3}{2}f(n-1)$. We claim that we have, for all x,

$$\sum_{n\leq|x|} \mathrm{N}_\eta(n,k) \leq 4\frac{2^{|x|}}{f(|x|)}. \tag{3.15}$$

Assuming this claim, we find

$$\begin{aligned}|x| - \log \sum_{n\leq|x|} \mathrm{N}_\eta(n,k) &\geq |x| - \log 4\frac{2^{|x|}}{f(|x|)} \\ &= \log f(|x|) - 2.\end{aligned}$$

Because f has an unbounded limit inferior, so does the function that maps n to $\log f(n) - 2$, thus the equation above proves the theorem.

We shall prove (3.15) by induction on the length of x. The base case, $|x| = 1$, is immediate. Assume now that (3.15) holds for all n less than $|x|$, which gives us

$$\begin{aligned}\sum_{n\leq|x|} \mathrm{N}_\eta(n,k) &= \mathrm{N}_\eta(|x|,k) + \sum_{n\leq|x|-1} \mathrm{N}_\eta(n,k) \\ &\leq \frac{2^{|x|}}{f(|x|)} + 4\frac{2^{|x|-1}}{f(|x|-1)}\end{aligned}$$

By the limited growth rate of f, we then get

$$\begin{aligned}\sum_{n\leq|x|} \mathrm{N}_\eta(n,k) &\leq \frac{2^{|x|}}{f(|x|)} + 2\frac{2^{|x|}}{\frac{2}{3}f(|x|)} \\ &\leq \left(1 + \frac{2}{\frac{2}{3}}\right)\frac{2^{|x|}}{f(|x|)},\end{aligned}$$

proving (3.15). □

This result can be extended to other pairing functions as well. Suppose we have a pairing function and two functions, ℓ_1 and ℓ_2, such that the length of the pair consisting of strings k and i is $\ell_1(k) + \ell_2(i)$. In that case, compression should be considered with respect to ℓ_2 and the key inequality of the theorem should be changed to

$$|\langle k, \mathrm{rank}(x : \eta_k)\rangle| \leq \ell_2(x) - d.$$

The above theorem augments our observation in Example 3.4.28 that η may act as a measure of both algorithmic and computational complexity. A data structure conforming to a parameterization in $\mathscr{F}_{\mathbf{FPT}}(A)$ best compresses those instances for which deciding membership in A is easy. This is especially pleasant because the success of fixed-parameter tractability hinges on the observation that deciding membership of real-world data is easy. Note that this compression is lossless.

The lossy compression obtained by dropping the second part of the two-part data structure may be of its own interest. We note that the lossy compression of a string x that has a good lossless compression need not be particularly good. With this, we mean that it is possible that few other strings of length $|x|$ have as good of a lossless compression, yet most have a better lossy compression. This is because having high sophistication may in fact be a compressing property.

3.4.34. Example. Suppose η is a parameterization for which μ_η and soph_η coincide, and for which for all n we have that $\mathrm{M}_\eta(n)$ is roughly $\log n$. Now, choose some string x and a parameter value k such that we have $|k| = \frac{|x|}{2}$. We alter η by removing x from all slices $\eta_{k'}$ of η for which we have $|k'| < |k|$, and replacing slice η_k by the singleton $\{x\}$. With respect to this modified parameterization, the sophistication and minimization still coincide. The sophistication of x is $\frac{|x|}{2}$, and so is the minimum length of its lossy compression. Additionally, the parameterized complexity of x, representing the length of the lossless compression, is also more or less equal to the sophistication of x. This is so because the rank of x in slice k is 1, thus the length of the second part of the two-part code for x is negligible.

Let n be the length of x. A lossless compression of length $\frac{n}{2}$ is possible for at most a $2^{-n/2}$-fraction of the strings of length n. Therefore, the lossless compression available for x is pretty good. By contrast, with a length of $\frac{n}{2}$, the lossy compression available for x is comparatively poor. Indeed, because $\mathrm{M}_\eta(n)$ is roughly $\log n$, all other strings have a lossy compression of roughly length $\log n$.

In applications, compression performance is often measured by the compression ratio or, for data streams, the compressed data rate [133, 139] (see also Example 3.4.1). Thus, compression performance is assumed to be constant throughout all input lengths. Contrary to this, a typical parameterization η may be so that M_η is logarithmic as a function of the length of strings. We shall not go into this discrepancy here and leave it to be explored in future research.

Technically, Theorem 3.4.33 above is not much more than a loose contrapositive statement of Theorem 3.3.20. Of course, the context is different and with it the interpretation. Here, the focus is on the two-part data structure. If we are to use this data structure as the input to a procedure, we should keep an eye on the way we express the computational complexity of the procedure. Rather than as a function of the length of the original string, the running time should be measured as a function of the length of the data structure. Luckily, if a parameterization η has uniform exponential density, then for all k and $x \in \eta_k$, the quantities $|x|$ and $|\mathrm{rank}(x : \eta_k)|$ are related by polynomials. Thus, provided the data structure can be decoded fast, if a set is in **FPT** with some parameterization, the recoded version is also in **FPT**.

This finally brings us to the adaptations of procedures needed in order to make use of the two-part data structure. Simply put, a procedure can be made to act on the two-part data structure by having it reconstruct the original string before proceeding as usual. Specifically, we are thus interested in the computational cost of decoding the two-part data structure. Let η be a parameterization and k a parameter value. We are interested in the situation where, for every string $x \in \eta_k$, we can compute x from $\langle k, \mathrm{rank}(x : \eta_k)\rangle$ in time polynomial in $|x|$. As just mentioned, if η has uniform exponential density, the running time of this procedure would also be polynomial in $|\mathrm{rank}(x : \eta_k)|$. Likewise, it would be polynomial in $|\langle k, \mathrm{rank}(x : \eta_k)\rangle|$. Using binary search, we find that if x can be computed from its rank in polynomial time, then its rank itself can be computed in polynomial time [79, Theorem 6.1]. Hence, there is a neat characterization of the parameterizations of interest.

3.4.35. Definition. A parameterization is *fixed-parameter p-rankable* if there is a parameterized procedure ϕ, a computable function f, and a polynomial p satisfying

- for every parameter value k, the partial application of ϕ to k yields a computation for $\mathrm{rank}(\cdot : \eta_k)$ that runs in $f(k) \cdot p$ time.

3.4.36. Example. The parameterizations related to certain p-cylinders are examples of fixed-parameter p-rankable parameterizations. Let $g \colon 2^+ \to 2^+ \times 2^+$ be a polytime-isomorphism that is increasing in its second component. This means that, for all strings z_1, z_2, and y, if we have $\mathrm{asInt}(z_1) < \mathrm{asInt}(z_2)$, then we also have $\mathrm{asInt}(g^{-1}(y, z_1)) < \mathrm{asInt}(g^{-1}(y, z_2))$. Further, let η be the parameterization related to g as described in Example 3.2.13. We shall show that the second component of the image of g, can be used to compute the rank of a string in a slice of η.

Recall that we defined the rank of a string x in a set X as the number of elements in the set

$$\{y \in X \mid \mathrm{asInt}(y) \leq \mathrm{asInt}(x)\}.$$

In particular, the rank is defined also for a string x that is not a member of X. Therefore, we can express the rank of a string in a slice of η as the sum of ranks in sets defined by a constant first component in the image of g. If we denote by $g_1^{-1}(y)$ the set of strings $\{g^{-1}(y, z) \mid z \in 2^+\}$, then, for every k and x, we have

$$\mathrm{rank}(x : \eta_k) = \sum_{j \leq \mathrm{asInt}(k)} \mathrm{rank}(x : g_1^{-1}(\mathrm{asStr}(j))). \tag{3.16}$$

As the number of terms in this sum is a function of k, all we need to show is that each term can be computed in a time bound polynomial in $|x|$. This polynomial must be independent of k. To this end, we assume j is fixed and employ binary search. First, we use a series of strings $z_1, z_2, z_3, \ldots$, each string one bit longer than the one before it, in order to find a reasonably small value, z_{up}, for which we have

$$\mathrm{asInt}(x) \leq \mathrm{asInt}(g^{-1}(\mathrm{asStr}(j), z_{\mathrm{up}})). \tag{3.17}$$

Next, we use a binary search below z_{up} to find the largest z for which (3.17) holds. We have thus found the rank of x in $g_1^{-1}(\mathrm{asInt}(j))$ as $\mathrm{asInt}(z)$. If p is the polynomial bounding the running time of computing g^{-1}, then, for every x, the above procedure finishes in a time in $\mathcal{O}(|x|^2 \cdot p(|x|))$. For the complete sum (3.16), we get a time bound of $\mathcal{O}(\mathrm{asInt}(k) \cdot |x|^2 \cdot p(|x|))$, which is as required.

A fixed-parameter p-rankable parameterization of which all slices are sparse could be called *fixed-parameter p-printable*. Following our observations at the end of Section 3.3.3, the slices would then be sets of small generalized Kolmogorov complexity [8]. In that way, the fixed-parameter p-rankable parameterizations provide yet another way of combining algorithmic complexity and computational complexity. The slices of such a parameterization are a model for non-random sets, and at the same time, they are easy in the sense of Definition 3.3.25.

Being fixed-parameter p-rankable is a property of a parameterization in isolation. This means that such rankable parameterizations can be considered also in relation to sets that are themselves in no way p-rankable. Indeed, a slice of a fixed-parameter p-rankable parameterization may contain both members and nonmembers of the set at hand. In this context, it is worthwhile to note that it is not even expected that every set in **P** is p-rankable. If they were, then all slices of all parameterizations in $\mathcal{L}_{\mathbf{FPT}}$ would be p-rankable. Moreover, the polynomial hierarchy would then collapse to **P** [79]. This collapse is caused by the fact that the class of rankable sets is not closed under polytime-isomorphisms. If we broaden our scope to include such isomorphisms, we are looking at *scalable* sets instead of at rankable sets [69]. The difference between fixed-parameter scalable and fixed-parameter p-rankable sets can conveniently be pointed out in Example 3.4.36. For the scalable variant, we would not need to require that the polytime-isomorphism is increasing in its second component. Regarding the existence of non-scalable sets in **P**, we know little more than a result of Allender [7] on sparse sets and one-way functions: A sparse non-scalable set in **P** exists if and only if there is an honest, polynomial-to-one, polytime-function without a polytime-inverse on $\{0\}^+$.

With respect to a parameterization η, it may be hard to find a two-part encoding of a string x, let alone one that uses $\mathrm{pc}_\eta(x)$ bits. Note, however, that two-part encodings that are suboptimal in that they do not realize $\mathrm{pc}_\eta(x)$ may be worthwhile too. The second part of a two-part encoding of x is no longer than $|x|$ and the overhead imposed by the first part may be small. By recording a parameter value in a two-part data structure, we can keep track of what is known about x. The data structure has the advantage that it can be updated whenever a more compressing parameter value is found.

One way to obtain encodings of strings in our data structure is via transformations *inside* slices. Operations on data may exist that do not impact the parameter value k, and have a known impact on the rank in a slice. One can imagine a setting involving graphs where the act of adding an isolated vertex does not influence the sophistication. It may be possible to perform such operations on the two-part data structure directly, without first reconstructing the original graph, or, in general, string. Essentially, we record the steps by which a string was constructed, as a series of transformations that enumerate the elements of a slice [see also 80, Theorem 4.2]. In this way, the data structure is transparent to certain complexity-preserving operations. We could say that the data structure is thus *partially homomorphic*. As, by Theorem 3.4.33, the data structure may be compressing, we could even say that parameterizations allow for *partially homomorphic compression*.

Other compression methods that enable operations on the compressed representations exist. One nice example is the data store by Agarwal, Khandelwal, and Stoica [4] called *Succinct*. This compression method is designed specifically to allow certain pattern searches to be performed directly on the compressed data. Whether or not it is suitable in the context of a certain computational workload is left for the relevant specialist to decide.

Unique to our parameterized approach is that we can link our data structure to the computational complexity of a set. In general, the data structure we use determines what operations are feasible. If we require fewer operations to be feasible, a more compressing encoding of our data may be available. For a set A, the parameterizations in $\mathscr{F}_{\mathbf{FPT}}(A)$ are representative of the computational redundancies in deciding membership in A. The operations to which our two-part data structure is transparent are determined by these redundancies. This way, our approach provides data structures tailored to specific computational workloads.

3.5 as Computational Redundancy

We can picture a parameterization as an infinite stack of slices. So far, we have worked under the hypothesis that complexity increases as we ascend through this stack. As such, parameter values represent levels of complexity. However, when each slice represents a level of constant complexity, then, relative to their length, the complexity of instances decreases inside each slice. As the length of instances in a given slice goes up, their relative complexity goes down.

3.5.1. Example. With **FPT**, we consider decidability in polynomial time easy, and use parameter values to express computational complexity beyond polynomial time. Thinking of **FPT** as computational complexity up to computability in polynomial time, we can see the importance of the first elements of slices. Let A be a set that is in **FPT** with a parameterization η. This means that, for some function f and polynomial p, membership in A can be decided in $f(k) \cdot p(|x|)$ time steps, whenever we have $x \in \eta_k$. Now, for those x and k that satisfy $|x| \geq f(k)$, we have

$$f(k) \cdot p(|x|) \leq |x| \cdot p(|x|).$$

In other words, for sufficiently long strings x in a fixed slice η_k, membership in A can be decided in polynomial time, irrespective of k.

The example above shows that the especially interesting elements of a slice η_k are those of length at most $f(k)$. Typically, however, the function f is rather fast-growing. Therefore, the observation in the example may not be very effective in locating where the computational complexity of a set originates. There are simply too many strings x that satisfy $|x| \leq f(k)$. In many cases, the computational complexity of a set can be pinpointed to far shorter initial segments of the slices of a parameterization.

3.5.2. Example (continued from Example 3.2.13). Let A be a p-cylinder and $g\colon 2^+ \to 2^+ \times 2^+$ a corresponding isomorphism. Recall that the parameterization based on g was defined as

$$\eta = (\{x \mid \text{asInt}(g_1(x)) \leq \text{asInt}(k)\})_{k \in 2^+}.$$

In the parameterized decision procedure for A outlined in Example 3.2.13, only the first component of the image of g was taken into consideration. Thus, if g maps x to (y, z), then the length of z is largely irrelevant for the computational complexity of deciding membership of x in A. We might as well replace z with the constant string $\mathtt{0}$ of length one. Indeed, the string $x' = g^{-1}(y, \mathtt{0})$ is a member of the same slices as x. Additionally, we can bound the length of x', because g^{-1} is polytime-computable. In particular, the length of x' is polynomial in the length of $y = g_1(x)$. Thus for every k for which we have $x \in \eta_k$, we can bound $|x'|$ by a polynomial of $|k|$, because we must have $\text{asInt}(y) \leq \text{asInt}(k)$. Equivalently, we can bound $|x'|$ by a polylogarithmic function of $\text{asInt}(k)$.

In the example above, we sent a string x to another string x' in the same slice as x, in a way that preserves membership. Put differently, we defined a reduction from the set A to itself that was well-behaved with respect to the parameterization. The reduction is akin to an *autoreduction* in the spirit of Trakhtenbrot [142]. Where the definition of an autoreduction excludes reducing to the input string, our reduction satisfies a condition that is in some sense stronger. Namely, our reduction meets a bound, as a function of the parameter value, on the length of the strings to which an input is reduced. In this light, our reduction can be compared to a more restrictive type of autoreduction, the *self-reduction* [106]. In the textbook treatment by Balcázar, Díaz, and Gabarró [16, Section 4.5], self-reducibility is defined as autoreducibility where all strings to which an input is reduced are strictly shorter than the input. However, many of the results around self-reducibility extend to more general orders than the "shorter than"-order. Indeed, the definition of self-reducibility can be generalized to encompass such more general orders [92, 117, 30]. Unfortunately, even this generalized definition does not adequately capture the behavior of our reduction in Example 3.5.2 [33]. Reductions of this type form a class of reductions of their own.

One of the reasons this class of reductions is of interest is because of its relationship to *preprocessing*. The goal of preprocessing is to eliminate all computational redundancy from an input and identify its computationally hard part. In the case of Example 3.5.2, we could throw away the second component in the image of the isomorphism g. Crucially, doing so allowed us to reduce the input in polynomial time to an equivalent string of a length that was bounded by a function of the parameter value. Returning once more to Figure 3.5, we see that essentially only the useful information was relevant for deciding membership. The algorithmic noise could be considered a form of computational redundancy. Generally, preprocessing may not be so powerful and the image of the reduction may be far longer than the parameter value. In other words, the length of the result of preprocessing a string x is not guaranteed to be at most that of x. However, as long as the length of the result can be bounded by a function of a parameter value for x, the preprocessing has some guarantee on its effectiveness.

Synopsis

Preprocessing procedures such as the one introduced in Example 3.5.2 are formally known as *kernelizations*. Just like with other forms of reducibility, the definition of a kernelization allows for some variation. Varying the constraints we place on a kernelization, we can obtain anything from a many–one kernelization to a Turing kernelization. The spectrum of possible definitions is the topic of Section 3.5.1. As far as the existence of a kernelization for a set is concerned, there is no difference between the various kinds of kernelization. We shall see that a set A has a kernelization of any kind with respect to some given parameterization precisely when A is fixed-parameter tractable. However, a less constrained kind of kernelization may be able to reduce its inputs to shorter strings than a more constrained kind of kernelization could. In that sense, the various kinds of kernelization may differ in effectiveness.

The difference between the length of an input string and that of the string to which it is reduced is a measure of the computational redundancy in the input. Thus, smaller parameter values indicate more computational redundancy. A kernelization is a way to relate a parameterization with the computational redundancy present in the inputs to a decision problem. Differences in the effectiveness of kernelizations can therefore be interpreted as differences in the amount of computational redundancy that is recognized. Typically, a kernelization is considered to be reasonably effective if the length of the reduced strings is polynomial in the numeric parameter value. In Section 3.5.2, we go into detail about these *polynomial* kernelizations. We consider the possibility of a relationship between the size of a kernelization and the running time associated with fixed-parameter tractability. This relationship turns out to be very weak, thus we find that computational tractability and computational redundancy are distinct measures of complexity.

Moreover, we find in Section 3.5.2 that the existence of a polynomial kernelization depends on the kind of kernelization we target. For this finding, we introduce a technique for constructing sets that are kernelizable in one way but not another. This technique revolves around composing sets as the disjoint union of a hard part and a computationally redundant part that reduces to the hard part. The same technique is used in Section 3.5.3 to build a fine-grained hierarchy of polynomial kernelizability. We identify what additional constraints in the definition of a kernelization lead to a strictly smaller class of sets having polynomial kernelizations. Thus, the levels in our hierarchy correspond to different kinds of polynomial kernelization. Most of the results concerning the hierarchy are the outcome of a joint work with Ralph Bottesch and Leen Torenvliet [153].

At one level of our hierarchy of kernelizability, we can reinterpret the computational redundancy identified by a kernelization as related to "advice". With the corresponding definition of a kernelization, more computational redundancy is linked to less advice being needed by a decision procedure. Specifically, having a polynomial kernelization is conceptually the same as having a decision procedure that takes an advice string of polynomial length. In Section 3.5.4, we compare this notion of polynomial advice with the leading notion of polynomial advice in computational complexity theory. While the traditional notion is shown to have some strong ties with polynomial kernelizability, it is nevertheless different from our new notion of advice. Neither notion of polynomial advice is stronger than the other.

The level of our hierarchy of kernelizability that represents a notion of advice appears again in Section 3.5.5. In that section, it is shown that an established technique for lower bounding the minimum size of a kernelization can be generalized. The lower bounding technique was originally used to lower bound the size of the most restrictive kind of kernelizations for certain decision problems. However, for less constrained kinds of kernelization, the technique works just as well. We shall see that it generalizes to the kind of polynomial kernelization that we related earlier to a notion of polynomial advice.

3.5.1 Types of Kernelization

In the Flum and Grohe framework, a *kernelization* can be defined as follows. Let A be a set and κ a parameterization in the sense of Flum and Grohe as we have seen in Section 1.3. A kernelization for A with respect to κ is a polytime-computable function $\phi\colon 2^+ \to 2^+$ that satisfies two properties. The first is that for all strings x, we have $x \in A \iff \phi(x) \in A$. The other property required is that there is some computable function $h\colon 2^+ \to \mathbb{N}$ such that, for all strings x, we have $|\phi(x)| \leq h(\kappa(x))$. In other words, a kernelization is a polytime many–one reduction with a length bound on the image in terms of the parameter value of the input.

As we shall soon see, kernelizations are closely related to fixed-parameter tractability. However, there is no reason we should restrict ourselves to many–one reducibility. This was first observed by Lokshtanov [101], who introduced the more general *Turing kernelization*, where the type of reducibility at play is Turing reducibility. In our framework, this very general form of kernelization can be defined as follows.

3.5.3. Definition. Let ϕ be a parameterized procedure that can query an oracle about membership of strings in a set A. The procedure ϕ is a *Turing kernelization* for A with respect to a decidable parameterization η if

- there is a polynomial p such that ϕ terminates on any input string x within $p(|x|)$ steps, regardless of the parameter value provided as the second argument to ϕ,
- ϕ converges to A on η, and
- there is a computable function $h\colon 2^+ \to \mathbb{N}$ such that any query made by ϕ on an input (x, k) has a length of at most $h(k)$.

While the definition may seem daunting, the three items really codify the same properties that were required of traditional kernelizations. That is, a Turing kernelization runs in polynomial time, decides membership, and obeys a length bound on the queries it makes.

We remark that every Turing kernelization also has a computable upper bound on the minimum length of a parameter value with which a queried string is in η. In other words, if, on input (x, k), a Turing kernelization queries a string q, then $\mu_\eta(q)$ can be bounded from above by a function of k. This is so because we have $|q| \leq h(k)$ and we can compute the value of $\mu_\eta(q)$ for all possible queries. That μ_η

is a computable function is a result of the requirement that the parameterization η is decidable. In fact, the decidability requirement in the definition can be relaxed to the criterion that there exists a computable parameter estimator κ for η. An upper bound on the minimum length of a parameter value for the queries is then found by replacing $\mu_\eta(q)$ by $|\kappa(q)|$ in the expression above.

Besides the length bound, more restrictions on the queries made by a Turing kernelization can be put in place. We observe that in the execution of a Turing kernelization, which string is queried next may depend on the answers of the oracle to queries made previously. Because of this, a Turing kernelization is said to be *adaptive*. We may restrict a Turing kernelization so that it must determine all the queries it will make at once, before knowing the feedback of any of them. With this addition of this restriction, we have defined a *truth-table kernelization*. This type of kernelization is relevant in the context of parallelized computation. As observed by Weller [151], the independence of the queries makes it possible to compute their answers in parallel. This turns truth-table kernelization into a design pattern for parallel algorithms.

Note that, by the bound on its running time, a Turing kernelization can be made to terminate regardless of the oracle it is provided with. In this regard, Turing kernelizations differ from Turing reductions in *computability* theory [131, 115], which may never stop querying their oracle. Therefore, a Turing kernelization is really a *bounded Turing kernelization*, or, equivalently, a *weak truth-table kernelization* [115] [see also 46, Section 2.4.3]. Even then, the names do not fully describe all properties, as a weak truth-table reduction need not terminate after it has received the answers to its queries.

Examples of Turing kernelizations can be found in the works of Jansen [87] and Thomassé, Trotignon, and Vušković [141], and in the textbook by Cygan et al. [37, Section 9.4]. While the kernelizations in the first two works are adaptive, the kernelization discussed in the textbook is non-adaptive and thus a truth-table kernelization. Further examples of both Turing and truth-table kernelizations can be found in the textbook by Fomin et al. [58].

More restrictive still are Turing kernelizations that are only allowed to make a constant number of queries. Yet, even Turing kernelizations that only make at most one query can do something that traditional many–one kernelizations cannot do. They can invert the answer they receive from the oracle in their membership decision. A further restriction we shall consider is related to this ability and was introduced by Jockusch [88] in computability theory and, later, by Selman [135] in complexity theory.

3.5.4. Definition. Let ϕ be a parameterized decision procedure that can query an oracle. Additionally, denote the set ϕ decides when the oracle answers according to a set A by $\phi_A^{-1}(\mathtt{1})$. The procedure ϕ is *positive* if, for all sets A and B that satisfy $A \subseteq B$, we have $\phi_A^{-1}(\mathtt{1}) \subseteq \phi_B^{-1}(\mathtt{1})$.

The reason for calling kernelizations that behave as in the above definition "positive" can most easily be seen by looking at positive truth-table kernelizations. As it turns out, the dependence on the oracle of a positive truth-table kernelization can be expressed as a Boolean formula that does not use negation. The disjunctive kernelizations and conjunctive kernelizations of Kratsch [97] are examples of positive truth-table kernelizations.

Finally, a positive Turing kernelization that makes at most one query could indeed be called a *many–one kernelization.* The oracle access of a many–one kernelization is quite restrictive. However, such kernelizations are still quite powerful from a computational complexity point of view.

3.5.5. THEOREM. *The following statements about a decidable set A and parameterization η are equivalent.*

1. *There is a many–one kernelization for A with respect to η.*
2. *There is a Turing kernelization for A with respect to η.*
3. *A is in* **FPT** *with η.*

Proof:
$1 \implies 2$. As a many–one kernelization is a restricted form of a Turing kernelization, this implication is immediate.

$2 \implies 3$. Let ϕ be the Turing kernelization for A with respect to η and let ψ be a decision procedure for A. Furthermore, let p be the polynomial bounding the running time of ϕ and let h be the function bounding the length of the queries made by ϕ. We can modify ϕ and turn it into a witness for the fact that A is in **FPT** with η. Suppose we want to decide membership of a string x in A and are supplied with a parameter value k such that we have $x \in \eta_k$. We run ϕ on (x, k) and whenever ϕ would pose a query to its oracle, instead we make it use ψ to figure out itself what the oracle would answer. Two things we know about the queries ϕ makes are that there are at most $p(|x|)$ of them and that each is bounded in length by $h(k)$. Now, if we denote the running time of ψ by t, we thus find that the modification to ϕ prolongs its running time by no more than an additional $p(|x|) \cdot t(h(k))$ steps. Note that h is computable by Definition 3.5.3 and t is computable because ψ terminates on all inputs. Thus, the modified ϕ indeed witnesses that A is in **FPT** with η.

$3 \implies 1$. The core idea for this implication was present already in Example 3.5.1. Let f be the computable function and p the polynomial such that, for all x and k with $x \in \eta_k$, membership of x in A can be decided in $f(k) \cdot p(|x|)$ steps. As shown in Example 3.5.1, membership in A of sufficiently long instances can be decided in polynomial time without using an oracle. This leaves us to define a parameterized decision procedure for instances x and parameter values k that satisfy $|x| < f(k)$. Such membership questions can, however, simply be delegated to the oracle while still satisfying the definition of a many–one kernelization. □

The previous result urges us to look at even more restricted forms of kernelization. A closer look at Example 3.5.1 tells us that kernelizations show us where the computational complexity in a decision problem resides. If a set A has a kernelization with respect to a parameterization η, then all computational complexity of A resides in initial segments of the slices of η. Of course, when we attempt to identify the computationally hard part of the set, we try to keep this hard part as small as possible. The bound h in Definition 3.5.3 can be used to formalize what "small" should mean in this case. Because we are primarily interested in the limiting behavior of h, we shall think of h not as a function of a parameter value k, but as a function of $\text{asInt}(k)$. In many cases, this is quite natural. For example, with the parameterization defined in Example 2.2.3, the values of $\text{asInt}(k)$ relate to the lengths of instances. We say that a Turing kernelization is a *polynomial Turing kernelization* if the associated bound h can be a polynomial of $\text{asInt}(k)$. More generally, we refer to the bound as a function of $\text{asInt}(k)$ as the *size* of the kernelization.

3.5.6. Example (continued from Example 3.5.2). The construction of the previous example lies at the heart of a polynomial many–one kernelization for p-cylinders. For the p-cylinder A with isomorphism g and parameterization η, this kernelization would proceed as follows on an input (x, k). First, the procedure would compute $g_1(x)$ and verify that $\text{asInt}(g_1(x)) \leq \text{asInt}(k)$ holds. If it does not, we do not have $x \in \eta_k$ and the procedure returns `?`. Because g is computable in polynomial time, this can be done in a time that is bounded polynomially in $|x|$. When we do have $x \in \eta_k$, the next step of the procedure is to construct $x' = g^{-1}(g_1(x), \mathtt{0})$. As we had seen before, the length $|x'|$ can be bounded by a polylogarithmic function of $\text{asInt}(k)$. Finally, the procedure queries the oracle for membership of x' in A and returns the answer of the oracle. By returning the answer of the oracle, the kernelization ensures it is a positive kernelization. As it queries at most one string of a length that can be bounded by a polylogarithmic function, it is a polylogarithmic many–one kernelization. Of course, it is also a polynomial many–one kernelization.

Most treatments of kernelization, including the standard textbook of Fomin et al. [58], require an additional property of the size of a kernelization. With respect to a parameterization η, this additional property is as follows for a kernelization ϕ of size h: For any query q that ϕ poses to its oracle, there must be a parameter value k' for which we have $q \in \eta_{k'}$ and $\text{asInt}(k') \leq h(\text{asInt}(k))$. This is a technical restriction that codifies that the queries may not have an overly high complexity. It is not always imposed [e.g. 57], and sometimes relaxed to obtain a notion of *compression* [24]. With the additional requirement in place, the kernelization in the previous example is no longer a polylogarithmic kernelization: The parameter values for the query are the same as the parameter values for the input instance, hence $\text{asInt}(k')$ need not be polylogarithmic in $\text{asInt}(k)$. However, as each query is a member of the same slices as the input instance, our kernelization is still a polynomial kernelization. With respect to polynomial kernelizations, the additional requirement is often not much of an issue. In a typical parameterization η, each instance x has a parameter value k that satisfies $x \in \eta_k$ and $\operatorname{asInt}(k) \leq |x|$. This subsumes the additional requirement.

Ideally, the order on parameterizations as defined in Definition 3.2.20 should preserve the size of kernelizations. Suppose we are working with a set A and we have two parameterizations, η and ζ, both members of $\mathscr{L}_{\mathbf{FPT}}$, that satisfy $\eta \preccurlyeq \zeta$. In Theorem 3.2.27, we have seen that $\mathscr{F}_{\mathbf{FPT}}(A)$ is upward closed, meaning that if A is in **FPT** with η, it is also in **FPT** with ζ. Unfortunately, it is not the case that if A has a Turing kernelization of size h with respect to η, it also has one of size h with respect to ζ. This is because the size of a kernelization is not well-defined with respect to classes of parameterizations that are equivalent in the order on parameterizations. However, inside a class of equivalent parameterizations, suitable parameterizations can be found.

3.5.7. Theorem. *Let η and ζ be parameterizations such that η is below ζ in the order on parameterizations. If a set A has a Turing kernelization of size h with respect to η, then there is a parameterization ζ' such that*

- *ζ' is equivalent to ζ in the order on parameterizations, and*
- *A has a Turing kernelization of size h with respect to ζ'.*

Proof:
Let f be a computable function bounding $\operatorname{gap}_{\eta,\zeta}$ from above. If f does not grow faster than the identity function, a Turing kernelization of size h with respect to η would also be one of size h with respect to ζ. In that case, the theorem is proven by taking $\zeta' = \zeta$. Assume, therefore, that f does grow faster than the identity function. We shall construct ζ' so that it is equivalent to ζ and the gap from η can be bounded by a function that does *not* grow faster than the identity function. For this, the idea is to make the parameter values of ζ longer. Denote, for a number m and string k, the string consisting of the first m bits of k by $\operatorname{prefix}(m, k)$, and consider the parameterization

$$\zeta' = (\{x \mid \exists m\colon f(m) \leq |k'| \text{ and } x \in \zeta_{\operatorname{prefix}(m,k')}\})_{k' \in 2^+}.$$

If a string x is in slice k of ζ, then it is in a slice k' of ζ' where k is a prefix of k' and we have $f(|k|) \leq |k'|$. The converse is true as well and, for all x, we have $\mu_{\zeta'}(x) = f(\mu_\zeta(x))$. Thus, the definition of ζ' ensures that $\operatorname{gap}_{\zeta',\zeta}$ is bounded from above by f and that $\operatorname{gap}_{\zeta,\zeta'}$ is bounded from above by the inverse of f. Hence, ζ' is equivalent to ζ in the order on parameterizations. Moreover, the definition is so that $\operatorname{gap}_{\eta,\zeta'}$ is bounded from above by the identity function. By our earlier remarks, it follows that A has a Turing kernelization of size h with respect to ζ'. $\square$

Unfortunately, a similar argument can be used to shown that the size of a Turing kernelization does not mean much on an equivalence class of parameterizations. If a set A has a Turing kernelization of any computable size with respect to a parameterization η, then there is a parameterization η' such that

- η' is equivalent to η in the order on parameterizations, and
- A has a *polynomial* Turing kernelization with respect to η'.

The choice of a specific parameterization is therefore important in the study of kernelizations. As there is no formal definition of what constitutes a "natural" parameterization, this will always be more of an art than a science.

3.5.8. EXAMPLE. A polynomial many–one kernelization that has become part of parameterized comlexity folklore is that of the vertex cover problem. In Example 3.3.24, the vertex cover problem was defined as

$$\textsc{VertexCover} = \{(G, l) \mid \text{graph } G \text{ has a vertex cover of size at most } l\},$$

A polynomial many–one kernelization for VERTEXCOVER with respect to the parameterization that bounds the size of the solution,

$$\eta = (\{(G, l) \mid l \leq \mathrm{asInt}(k)\})_{k \in 2^+},$$

may proceed as follows given an instance (G, l) and a parameter value k.

1: If $l > \mathrm{asInt}(k)$, return ?. We do not have $(G, l) \in \eta_k$, so we do not need to decide wether (G, l) is a member of VERTEXCOVER or not.

2: Remove from G all isolated vertices. There is no reason to include any of these vertices in a vertex cover of G.

3: Set j to the number of vertices of G that are connected to more than l others, and remove from G all these vertices. Observe that each of the removed vertices must be included in a vertex cover of G of size at most l, since we cannot include all their neighbors.

4: If more than $(l-j)(l+1)$ vertices remain in G, return 0. The original graph had a vertex cover of size at most l if and only if the reduced graph G has a vertex cover of size at most $l - j$. Each of the at most $(l - j)$ vertices in a vertex cover is connected to at most l others. Therefore, at this point, G should not have more than $(l-j)+(l-j)*l = (l-j)(l+1) \leq (\mathrm{asInt}(k)+1)^2$ vertices.

5: Query the oracle for $(G, l - j)$ and return the result.

Note that each of the above steps can be performed within a number of steps that is polynomial in the size of the input (G, l). Also, note that this procedure converges to VERTEXCOVER on η. Lastly, the procedure makes at most one query of which the size can be bounded by a polynomial of $\text{asInt}(k)$ and it does not invert the answer of the oracle. Therefore, the procedure is indeed a polynomial many–one kernelization for VERTEXCOVER.

The queries made by a polynomial Turing kernelization are generally considered to be reasonably short [57]. Indeed, finding polynomial Turing kernelizations for various sets has gained significant interest [73, 37, 58]. As we shall soon see, polynomial many–one kernelizations are strictly less powerful than polynomial Turing kernelizations. For polynomial kernelizations, a theorem like Theorem 3.5.5 does not exist.

3.5.2 Polynomial Turing Kernelizations

Given Theorem 3.5.5, we may wonder whether the length of queries made by a kernelization relates to the parameterized running time of a decision procedure. At least in one direction, such a relationship can be found. Let A be a set that is decidable in exponential time and for which there is a polynomial Turing kernelization with respect to some parameterization η. We consider the parameterized running time of the parameterized decision procedure for A that is constructed in the proof of Theorem 3.5.5. Deciding queries in exponential time, we find that there must exist a polynomial q such that the parameter dependence in this running time is of the form $2^{q(\text{asInt}(k))}$. Of course, the factor in the running time that is dependent on the length of the instance can be bounded by a polynomial, say p. To summarize, because A is in **EXP** and has a polynomial Turing kernelization with respect to η, it is decidable in time $2^{q(\text{asInt}(k))} \cdot p(|x|)$ with respect to η.

That the converse of this observation fails to hold was proven by Bodlaender et al. [26] for the restricted case of polynomial many–one kernelizations. Using a different proof technique, we can extend this result to polynomial Turing kernelizations in their full generality.

3.5.9. THEOREM. *Let h be a time-constructible function in $2^{o(n)}$. There exists a set A and parameterization η satisfying*

- *A is in* **FPT** *with η, witnessed by a parameterized procedure, taking input of the form (x, k), that has a running time in $\mathcal{O}(2^{\text{asInt}(k)} + |x|)$, yet*

- *A admits no Turing kernelization of size h.*

In particular, there is a set that is decidable in time $\mathcal{O}(2^{\text{asInt}(k)} + |x|)$ with respect to a parameterization η, but admits no polynomial Turing kernelization.

Proof:
We shall derive the existence of a set A and parameterization η with the desired properties from the time hierarchy theorem [77, 82]. If the set would have a Turing kernelization of size h, using it in a decision procedure would lead to a speedup ruled out by the time hierarchy theorem.

Any set for which there is a kernelization of a size that is bounded from above by a constant is in $\mathbf{P}$. Hence, we may assume that h is an unbounded function. Consider the time bound

$$t(n) = 2^{\frac{1}{5}h^{-1}(n)},$$

where we define $h^{-1}(n)$ as $\min\{m \mid h(m) \geq n\}$. The constant $\frac{1}{5}$ in this expression is fairly arbitrary, but chosen to simplify the last part of this proof. We claim that the function t is time-constructible. If h grows slower than the identity function, computing $h^{-1}(n)$ and, in turn, $t(n)$ is possible in $(h^{-1}(n))^2$ steps. Since $t(n)$ is bigger than $(h^{-1}(n))^2$ for almost all n, it follows that t is time-constructible. If h grows at least as fast as the identity function, computing $h^{-1}(n)$ and, in turn, $t(n)$ is possible in n^2 steps. Equivalent to the fact that $h(n)$ is in $2^{\mathrm{o}(n)}$, is the fact that h^{-1} is superlogarithmic and thus that t grows faster than any polynomial. Again, it follows that t is time-constructible.

Now, let A be a set that is decidable in time t, but not in time t' for any function t' that satisfies $t'(n)\log t'(n) \in \mathrm{o}(t(n))$. By the time hierarchy theorem, such a set exists. Our set A is in **FPT** with the parameterization

$$\eta = \left(\{x \mid \tfrac{1}{5}h^{-1}(|x|) \leq \mathrm{asInt}(k)\}\right)_{k\in 2^+}.$$

As membership of any string x in a set A can be decided in time $t(|x|)$, it can be decided with respect to η in time $\mathcal{O}(2^{\mathrm{asInt}(k)} + |x|)$. Here, the term $|x|$ serves only to allow for sufficient time to read the entire input. It remains to show that A lacks a Turing kernelization of size h with respect to η.

A Turing kernelization for A is only given a polynomial amount of running time, yet t grows faster than any polynomial. Hence a Turing kernelization for A cannot work without querying its oracle. To see why a kernelization for A with respect to η cannot be of size h, fix a kernelization and let p be the polynomial bounding its running time. Suppose toward a contradiction that the kernelization is of size h. We can combine the kernelization with a decision procedure for A running in time t to answer oracle queries. This yields a parameterized decision procedure converging to A on η with a running time of $p(|x|) \cdot t(h(\mathrm{asInt}(k)))$. By using a parameter value k such that $\mathrm{asInt}(k)$ is at least $\frac{1}{5}h^{-1}(|x|)$, we can turn this parameterized decision procedure into a regular decision procedure. The procedure we end up with is one deciding whether a given string x is in A, roughly running in time

$$t'(n) = p(n) \cdot t(h(\tfrac{1}{5}h^{-1}(n))).$$

The proof can now be completed by showing that $t'(n)\log t'(n)$ is in $\mathrm{o}(t(n))$. By our choice of A, there is no decision procedure for A running in time t'

when $t'(n)\log t'(n)$ is in $\mathrm{o}(t(n))$. Thus, our assumption that A has a Turing kernelization of size h would be violated.

Our strategy is to prove that $t'(n)$ is in $\mathrm{o}(\sqrt{t(n)})$. Because $\log t'(n)$ would then also be in $\mathrm{o}(\sqrt{t(n)})$, it would follow that the product of the two would indeed be in $\mathrm{o}(t(n))$. For proving that $t'(n)$ is in $\mathrm{o}(\sqrt{t(n)})$, we use a similar strategy: We verify that both factors in the definition of $t'(n)$ are in $\mathrm{o}(\sqrt[4]{t(n)})$.

Earlier, we observed that t grows faster than any polynomial. Therefore, the polynomial factor, $p(n)$, in the definition of $t'(n)$ is in $\mathrm{o}(\sqrt[4]{t(n)})$. All that remains is to show that $t(h(\frac{1}{5}h^{-1}(n)))$ is in $\mathrm{o}(\sqrt[4]{t(n)}) = \mathrm{o}(2^{\frac{1}{20}h^{-1}(n)})$. By our choice of the constant $\frac{1}{5}$ in the definition of t, this is straightforward. We have

$$\begin{aligned} t(h(\tfrac{1}{5}h^{-1}(n))) &= 2^{\frac{1}{5}h^{-1}(h(\frac{1}{5}h^{-1}(n)))} \\ &= 2^{\frac{1}{25}h^{-1}(n)}, \end{aligned}$$

which is in $\mathrm{o}(2^{\frac{1}{20}h^{-1}(n)})$ because h^{-1} is an unbounded function. □

While the set of which the above proof establishes the existence is outside **P**, it is not too unwieldy. If the kernelization size, h, grows linearly or faster, the resulting set is even decidable in linear exponential time. That is, it is a member of the class

$$\mathbf{E} = \bigcup_{c\geq 1} \mathbf{TIME}(2^{c\cdot n}).$$

This is more or less by necessity, since sets that are too far removed from **E** cannot show the desired behavior. In fact, the relationship between kernelization size and computational complexity we found earlier can be reversed for sets that are bi-immune for **E**. Let A be a set that is bi-immune for **E**, let η be a parameterization, and let p and q be polynomials. Furthermore, suppose there is a parameterized procedure that converges to A on η and, on input (x,k), runs in time $p(|x|)\cdot 2^{q(\mathrm{asInt}(k))}$. We claim that in this case, A must have a polynomial many–one kernelization with respect to η. To see why this is the case, observe that for almost all x and k with $x \in \eta_k$, the expression $p(|x|)\cdot 2^{q(\mathrm{asInt}(k))}$ must be superexponential in $|x|$. If this were not the case, then A would not be bi-immune for **E**. A consequence thereof is that $q(\mathrm{asInt}(k))$ must outgrow $|x|$. Because of this, a procedure that simply queries the string it is given as input counts as a valid polynomial many–one kernelization for A with respect to η.

By Theorem 3.5.9, we cannot infer the existence of a polynomial Turing kernel by a glance at a parameterized running time. The theorem settles the unconditional existence of a certain set without a polynomial Turing kernelization. However, it provides no means to rule out polynomial Turing kernelizations for any particular set that we may be interested in. Methods for lower-bounding the size of kernelizations exist, but focus mostly on many–one kernelizations. A very productive way of coming up with superpolynomial lower bounds on the size of many–one kernelizations was developed by Bodlaender et al. [26]. An overview of the state-of-the-art in these methods is provided by Kratsch [97], and as an in-depth textbook by Fomin et al. [58]. About Turing kernelizations, far less is known. The non-existence of polynomial-sized Turing kernelizations has been linked, conditionally, to hardness for a certain class of sets under a specific reduction [83]. We shall not go into details about this line of research and focus on a series of unconditional separations instead. First, we shall show that the adaptive powers of polynomial Turing kernelizations truly distinguish them from polynomial truth-table kernelizations.

3.5.10. THEOREM. *There is a set A and a parameterization η such that A has a polynomial Turing kernelization, but no polynomial truth-table kernelization with respect to η.*

Proof:
We shall construct the set A as the disjoint union of two sets, and detail the definition of those two sets, W and X, separately. Specifically, A is defined via

$$A = \{\mathtt{0}w \mid w \in W\} \cup \{\mathtt{1}x \mid x \in X\}.$$

We shall define X and our parameterization η so that A has a polynomial Turing kernelization with respect to η. The set W will be used to make sure that A has no polynomial truth-table kernelization with respect to η.

Ensuring the existence of a polynomial Turing kernelization. Our polynomial Turing kernelization must be adaptive, for we do not want it to be a polynomial truth-table kernelization. In order to achieve this, we define a function $s\colon \mathtt{2}^+ \to \mathtt{2}^+$ by

$$s(q) = \begin{cases} \mathtt{0}q & \text{if } q \notin W, \\ \mathtt{1}q & \text{if } q \in W. \end{cases}$$

Using this function, we build the set X from the set W as

$$X = \{x \mid \log|x| \in \mathbb{N} \text{ and } (\underbrace{s \circ s \circ \cdots \circ s}_{(\log|x|)^2 \text{ times}})(\mathtt{0}^{\log|x|}) \in W\}.$$

Thus, ultimately, membership of a string x in X depends on membership of a string of length $\log|x| + (\log|x|)^2$ in W. This construction allows us to define a parameterization with respect to which the set A has a polynomial Turing kernelization, regardless of W. We define

$$\eta = (\{\mathtt{0}w \mid |w| \leq \text{asInt}(k)\} \cup \{\mathtt{1}x \mid \log|x| \leq \text{asInt}(k)\})_{k \in \mathtt{2}^+}.$$

With respect to this parameterization, a polynomial Turing kernelization may query its input when it is of the form $\mathtt{0}w$. For inputs of the form $\mathtt{1}x$, a polynomial

Turing kernelization can compute the appropriate series of strings to query using our function s. Note that it is important that the application of s in the definition of X is repeated $(\log|x|)^2$ times and not just $\log|x|$ times. If we would have repeated the application of s only $\log|x|$ times, the number of strings that would potentially be queried is only $2^{\log|x|+1} = 2 \cdot |x|$. This number is polynomial in $|x|$, hence a truth-table kernelization could simply query all these strings. By repeating the application $(\log|x|)^2$ we can define W so that a polynomial kernelization for A with respect to η is necessarily adaptive.

Preventing the existence of a polynomial truth-table kernelization. We construct the set W by diagonalizing against polynomial truth-table kernelizations. To ease our task, we adopt a particularly convenient model of computation in the presence of an oracle. Let $\phi_1, \phi_2, \phi_3, \ldots$ be an effective enumeration of parameterized procedures that invoke their oracle at most once, but may present it with a batch of queries. The oracle is expected to provide an answer to all of the queries it is presented. We may assume that every truth-table kernelization occurs in this list infinitely often. To diagonalize against all polynomial truth-table kernelizations, we run each of these procedures for a limited time, allowing only queries of limited length. When running some procedure ϕ_i, the running-time and query-size restriction we employ will be a polynomial of degree i.

The set W is constructed in stages. At stage $i \in \mathbb{N}$, we set n to a value that satisfies a few constraints, namely

1. we have $n^i \leq 2^n$ and $i < n$, and
2. no decision about membership in W is made about any string of length at least n.

Having thus set n, we define $x = \mathtt{0}^{2^n}$, which is so that we have $n = \log|x|$. Next, we run ϕ_i on input instance $\mathtt{1}x$ and parameter value $\mathrm{asStr}(n)$ for $|x|^i$ steps. Note that these inputs satisfy $\mathtt{1}x \in \eta_{\mathrm{asStr}(n)}$, so if ϕ_i is a kernelization with respect to η, then it will not output $\mathtt{?}$. In case ϕ_i invokes the oracle, let S be the set of strings it queries. If S includes a string of length greater than n^i, then ϕ_i cannot be a truth-table kernelization of size n^i. Therefore, if S includes such a long string, ϕ_i needs no further attention and we directly move on to the next stage, aborting the current stage. This way, we also make sure that $\mathtt{1}x$ is not a member of S, because, by constraint 1, its length, $2^n + 1$, is greater than n^i. By the bound imposed on the running time of ϕ_i, we find that S contains at most $|x|^i = 2^{n \cdot i}$ strings. Because of constraint 1, this number is strictly less than 2^{n^2}, and there must be a string $y \in \mathbb{2}^{n^2}$ such that the string $\mathtt{0}y\mathtt{0}^n$ is not in S. We can enforce membership of x in X to depend on membership of $\mathtt{0}^n$ in W. Because ϕ_i does not query $\mathtt{0}y\mathtt{0}^n$, it cannot know whether $y\mathtt{0}^n$ is a member of W, which gives us the freedom we need to diagonalize. Let $b_1, b_2, \ldots, b_{n^2}$ be the bits of y, that is, we have $y = b_{n^2} \cdots b_2 b_1$ and answer the queries made by ϕ_i as follows. All queries of which the answer is determined by previous stages of our construction of W are answered accordingly. This includes queries of the form $\mathtt{1}x$, which are treated in accordance with the definition of X. Queries of the form $\mathtt{0}b_j \cdots b_2 b_1 \mathtt{0}^n$, with $j < n^2$, are answered with b_{j+1}, ensuring that we have

$$y\mathtt{0}^n = (\underbrace{s \circ s \circ \cdots \circ s}_{(\log|x|)^2 \text{ times}})(\mathtt{0}^{\log|x|}).$$

All other queries are answered with $\mathtt{0}$. After thus answering the queries in S, we resume ϕ_i and keep running it for the remainder of its allotted $|x|^i$ steps. Finally, we place $y\mathtt{0}^n$ in W if and only if ϕ_i terminated and rejected its input $\mathtt{1}x$. This ensures that $\mathtt{1}x$ is a member of A if and only if ϕ_i rejects it.

With W constructed this way, we have made sure that there is no polynomial truth-table kernelization for A with respect to η. Suppose that there were such a truth-table kernelization of polynomial size q, running in time p. There would then be an index i such that ϕ_i codifies this kernelization and, for all $n > i$ we have $p(n+1) < n^i$ and $q(n) < n^i$. At stage i of the above construction of W, we made sure that the behavior of ϕ_i is in violation with membership in A for some string $\mathtt{1}x$. This shows that ϕ_i could not have been a polynomial truth-table kernelization for A with respect to η. Note that we simulated ϕ_i for only $|x|^i$ steps, instead of for $|\mathtt{1}x|^i$ steps, hence we needed to choose i so that we have $p(n+1) < n^i$. $\square$

Even with a bound on the length of queries in place, a truth-table kernelization cannot query all strings that are potentially queried by a given Turing kernelization. This observation lies at the heart of the above proof, and can be summarized as follows.

3.5.11. SLOGAN. *With kernelizations, the queries that* could *have been made matter too.*

Diagonalization is not a common technique in parameterized complexity theory. This is because it is not possible to diagonalize against parameterized running times, as there is no way to dominate the parameter dependence. The proof of Theorem 3.5.10 shows that it is possible to utilize diagonalization *inside* **FPT**. The running time of a kernelization is polynomial in the length of the input instance, and has no dependency on a parameter value. This observation can be used to further distinguish between different types of polynomial kernelizations.

3.5.3 A Hierarchy of Polynomial Kernelizations

The previous two theorems, Theorem 3.5.9 and Theorem 3.5.10, are the beginning of a hierarchy below **FPT**. Depending on the ways a kernelization is allowed to access its oracle, a polynomial kernelization may or may not exist. For convenience, let us define some parameterized complexity classes indexed by various forms of oracle access.

3.5.12. Definition. A set A is in $\mathbf{PKER}_{\text{Turing}}$ with parameterization η if there is a polynomial Turing kernelization for A with respect to η. When the oracle access is required to be of a restricted kind, this restriction is noted in place of the “Turing”-subscript.

With this notational aid, we can summarize a consequence of Theorem 3.5.9 as stating that the inclusion $\mathbf{PKER}_{\text{Turing}} \subset \mathbf{FPT}$ is proper. In turn, Theorem 3.5.10 can be summarized by the proper inclusion $\mathbf{PKER}_{\text{truth-table}} \subset \mathbf{PKER}_{\text{Turing}}$. These two inclusions form the top of a hierarchy between $\mathbf{PKER}_{\text{many–one}}$ and $\mathbf{FPT}$.

The approach taken in the proof of Theorem 3.5.10 can be used to obtain other separation results as well. To distinguish two types of kernelization, we construct a set that has a kernelization of one of the types, but does not have a kernelization of the other type. These sets are built as the disjoint union of two parts, where one part realizes the positive characteristic and the other the negative characteristic. In the proof of Theorem 3.5.10, we built the hard part, the part preventing the existence of a polynomial truth-table kernelization, by diagonalization. The diagonalization explicitly targeted polynomial truth-table kernelizations. However, it is also possible to reuse a single set as the hard part in multiple separations. This single set must withstand a form of autoreducibility and be sparse. We shall require it to be particularly sparse and have at most logarithmic density. By this, we mean that if we let $d(n)$ denote the number of elements in the set that are of length at most n, then $d(n)$ is in $\mathcal{O}(\log n)$.

3.5.13. Lemma. *There is a decidable set W with at most logarithmic density, for which no approximation that runs in linear exponential time and is allowed to adaptively query instances of W other than its input has an infinite domain.*

Proof:
A set W that is as required can be said to be bi-immune against linear exponential time Turing autoreducibility. As is typical for the creation of bi-immune sets, we shall use a finite injury priority construction [see 46, Section 2.11] for W.

Let $\phi_1, \phi_2, \phi_3, ...$ be an effective enumeration of procedures that can make use of an oracle. Contrary to what we did in the proof of Theorem 3.5.10, we allow the procedures to invoke their oracle any number of times. With each invocation, the membership of a single string is queried. We assign a higher priority to procedures with a lower index and diagonalize against procedures opportunistically. As a result, when we diagonalize against a procedure ϕ_i we may have to redo our diagonalization against procedures ϕ_j with $i < j$.

We shall phrase our construction as a procedure that determines membership of strings x in W, one string at a time. For this purpose, we think of W as a string-indexed array, or characteristic sequence, to which we can assign values. Initially, we know nothing about W and we set it to $(\mathsf{?}, \mathsf{?}, \mathsf{?}, \dots)$. Additionally, we keep track of the indices of procedures we have diagonalized against in a set I_{done} that is initially empty, $\emptyset$. As a final bookkeeping device, we maintain, for each string x, the maximum index of a procedure that we may want to diagonalize against at x. We codify this as another string-indexed array, I_{max}. By initializing I_{max} uniformly in x as $I_{\mathsf{max}}[x] = \log\log|x|$, we shall be able to realize the sparsity required of W. Throughout our construction of W, the set I_{done} will be finite and both I_{max} and W will only differ from their initial value at finitely many places.

For each successive string x, we go through each index i that satisfies $i < I_{\mathsf{max}}[x]$ and $i \notin I_{\mathsf{done}}$ in increasing order, and try to diagonalize against ϕ_i. We use a set Q to keep track of the strings that ϕ_i queries and of which membership in W is not yet determined.

```
1: Set $Q$ to $\emptyset$ and run $\phi_i$ on input $x$ for $2^{|x|^2}$ steps until any of the following happens:
    1.1: In case $\phi_i$ returns ? or runs out of time, we cannot diagonalize against $\phi_i$ at $x$, and continue with the next value of $i$ that satisfies $i \leq I_{\mathsf{max}}[x]$ and $i \notin I_{\mathsf{done}}$.
    1.2: In case $\phi_i$ queries a string $q$, we continue running $\phi_i$ after using the following to answer the query:
        1.2.1: If $W[q] \neq \mathsf{?}$, answer the query accordingly.
        1.2.2: Else, add $q$ to $Q$ and answer with $\mathsf{0}$.
    1.3: In case $\phi_i$ terminates and returns $b \in \mathbb{2}$, we can diagonalize against $\phi_i$ at $x$:
        1.3.1: For each $q \in Q$, set $W[q]$ to $\mathsf{0}$ and set $I_{\mathsf{max}}[q]$ to $i$. This ensures that $W[q]$ may only be changed later on in order to diagonalize against a procedure with a higher priority than $\phi_i$.
        1.3.2: Set $W[x]$ to $\mathsf{1} - b$, so that, with the current state of $W$, the procedure $\phi_i$ cannot be an approximation for $W$.
        1.3.3: Add $i$ to $I_{\mathsf{done}}$.
        1.3.4: Remove from $I_{\mathsf{done}}$ all values $j$ for which we have $i < j$. It is possible that $W[x]$ was previously set to $\mathsf{0}$ in order to diagonalize against a procedure of lower priority than $\phi_i$. These diagonalizations may now be broken.
```

If there are no more values of i to try and we have not decided on membership of x in W, we set $W[x]$ to $\mathsf{0}$ and continue with the next string x.

By running all procedures for $2^{|x|^2}$ steps on input x, the allotted time will outgrow any linear exponential time bound. Suppose that ϕ_i is an approximation for W that runs in linear exponential time and potentially queries instances of W other than its input. We claim that the domain of ϕ_i is necessarily finite. Assume, toward a contradiction, that the domain of ϕ_i were infinite. There must then exist a string x in the domain of ϕ_i with the following properties.

- The time taken by ϕ_i on input x is less than $2^{|x|^2}$.
- When the above procedure considers x, the value of $I_{\mathsf{max}}[x]$ is greater than i.
- From the point where the above procedure considers x onward, no value $j < i$ will ever be added to I_{done}.

The construction of W diagonalizes against ϕ_i at x and this diagonalization is never broken after that. Thus we may conclude that the domain of ϕ_i cannot be infinite.

It remains to show that W has at most logarithmic density. Suppose that at some point in the above construction of W only procedures with an index of at most c have been considered. We claim that at that point at most 2^c strings have been added to W. Observe that the only place where a string is added to W is in step 1.3.2 of the construction of W. Thus, at every provisional diagonalization against a procedure ϕ_i, at most one string is added to W. Additionally, note that ϕ_1 is diagonalized against at most once, ϕ_2 is diagonalized against at most twice, ϕ_3 at most four times, and so on. Thus, each ϕ_i contributes at most 2^{i-1} strings to W. From this, our claim follows, and because of the way we initialized I_{max}, there will be at most $2^{\log\log n} = \log n$ strings of length at most n in W. Therefore, the set W meets all requirements of the lemma. $\square$

Not only can a set W as given by Lemma 3.5.13 be reused for a range of separation results, we can do so with a single, point-cofinite, parameterization. This parameterization,

$$\eta = (\{\mathtt{0}w \mid |w| \leq \mathrm{asInt}(k)\} \cup \{\mathtt{1}x \mid \log|x| \leq \mathrm{asInt}(k)\})_{k\in 2^+}, \tag{3.18}$$

was already encountered in the proof of Theorem 3.5.10.

A first result we obtain is that the positive nature of many–one kernelizations is a genuine restriction.

3.5.14. Theorem. *With respect to the parameterization η defined by* (3.18)*, there is a set that has a polynomial kernelization that makes at most one query, but no polynomial positive truth-table kernelization.*

Proof:
Let W be a set as given by Lemma 3.5.13 and consider the set

$$A = \{\mathtt{0}w \mid w \in W\} \cup \{\mathtt{1}x \mid \mathrm{asStr}(|x|) \notin W\}.$$

Observe that the length of $\mathrm{asStr}(|x|)$ is logarithmic in $|x|$. Therefore, there is a polynomial kernelization for A with respect to the parameterization η defined by (3.18) that makes at most a single query: On the hard part, the strings of the form $\mathtt{0}w$, this kernelization queries its input. On the redundant part, the strings of the form $\mathtt{1}x$, this kernelization queries $\mathrm{asStr}(|x|)$ and returns the opposite of the answer of the oracle.

Assume, toward a contradiction, that A has a polynomial positive truth-table kernelization, ϕ, with respect to η. Using ϕ, we shall construct an approximation for W with an infinite domain. This approximation is allowed to query instances of W other than its input and runs in linear exponential time. By our choice of W in line with Lemma 3.5.13, such an approximation for W cannot exist. From this contradiction, the theorem follows.

Observe that, on any input of the form $\mathtt{1}x$, the kernelization ϕ does not query its oracle for a string $\mathtt{0}w$ for which we have $w = \mathrm{asStr}(|x|)$. This is because if its eventual decision would depend on the answer to such a query, ϕ would either be incorrect or not positive. Our approximation for W proceeds as follows on input w.

```
1: Set $x$ to $\mathtt{0}^{\mathrm{asInt}(w)}$, so that we have $\mathtt{1}x \in A \iff w \notin W$.
2: Set $k$ to $\mathrm{asStr}(\log|x|)$, so that we have $\mathtt{1}x \in \eta_k$.
3: Run $\phi(\mathtt{1}x, k)$ and in case it queries a string $q$, respond as follows:
    3.1: If $q$ is of the form $\mathtt{1}y$ with $|y| = |x|$, return ?.
    3.2: Else, if $q$ is of the form $\mathtt{1}y$, answer the query in accordance with
         *non*-membership of $\mathrm{asStr}(|y|)$ in $W$. By the previous if-clause, we
         can be sure that $\mathrm{asStr}(|y|)$ differs from $w$. Therefore, this step does
         not require us to query the oracle for the input string $w$.
    3.3: Else, $q$ is of the form $\mathtt{0}y$ and we answer the query in accordance with
         membership of $y$ in $W$. By our earlier observation, we once again
         have $y \neq w$.
4: Return the opposite of the value returned by $\phi(\mathtt{1}x, k)$.
```

By construction, this procedure is an approximation for W. The time required by this approximation can be bounded by a polynomial in $|\mathtt{1}x|$. Consequently, it can also be bounded by a linear exponential function of $|w|$, as required. Moreover, the approximation does not query the oracle for its input, so all that remains to be shown is that the domain of the approximation is infinite.

Because ϕ is a polynomial kernelization, the length of any query q in the above approximation for W can be bounded by a polynomial of $\mathrm{asInt}(k)$. Specifically, the length of the queries made by ϕ can be bounded by a polynomial of $\log|x|$. As a result, the length of a query made by ϕ on input $(\mathtt{1}x, k)$ can be greater than or equal to the length of x for only finitely many strings x. This means that the if-clause of step 3.1 in the above approximation algorithm for W is only satisfied for finitely many inputs w. Consequently, the approximation outputs ? for only finitely many inputs and the domain of the approximation must be infinite. This contradicts the nature of W, from which we conclude that A has no polynomial positive truth-table kernelization with respect to η. $\square$

Recall that a many–one kernelization is a positive kernelization that makes at most one query. Therefore, the above theorem implies an inclusion at the bottom of our hierarchy between $\mathbf{PKER}_{\text{many–one}}$ and $\mathbf{FPT}$.

3.5.15. Corollary. *We have* $\mathbf{PKER}_{\text{many–one}} = \mathbf{PKER}^{\text{positive}}_{1\text{ query}} \subset \mathbf{PKER}_{1\text{ query}}$.

Next to separations based on whether or not a polynomial kernelization is positive, we find separations based on the number of queries that are permitted.

3.5.16. Theorem. *With respect to the parameterization η defined by* (3.18)*, for every constant c, there is a set that has a polynomial positive kernelization that makes at most $c+1$ queries, but no polynomial kernelization that makes at most c queries.*

Proof:
Let W be a set as given by Lemma 3.5.13 and consider the sets defined uniformly in $c \in \mathbb{N}$ by

$$A_c = \{\mathtt{0}w \mid w \in W\} \cup \left\{ \mathtt{1}x \;\middle|\; \bigvee_{i \le c} \Big(\operatorname{asStr}(c \cdot (|x| - 1) + i) \in W \Big) \right\}.$$

As in Theorem 3.5.10 and Theorem 3.5.14, by construction these sets have a specific kernelization with respect to the parameterization η defined by (3.18). In this case, for every value of c, the set A_c has a polynomial positive kernelization that makes at most c queries. We shall show that A_{c+1} does not have a polynomial kernelization with respect to η that makes at most c queries. As in the proof of Theorem 3.5.14, the proof is by contradiction. Like before, the existence of such a kernelization implies the existence of a linear exponential time approximation for W with an infinite domain.

Suppose, toward a contradiction, that A_{c+1} has a polynomial kernelization ϕ that makes at most c queries. Our approximation for W proceeds as follows on input w.

```
1: Let $n$ and $i$ be the unique values such that $i$ is less than $c+1$ and we have
   $\operatorname{asInt}(w) = (c+1) \cdot (n-1) + i$, and set $x$ to $\mathtt{0}^n$. This way, if $w$ is a member
   of $W$, then $\mathtt{1}x$ is a member of $A_{c+1}$.
2: Set $k$ to $\operatorname{asStr}(\log |x|)$, so that we have $\mathtt{1}x \in \eta_k$.
3: Run $\phi(\mathtt{1}x, k)$ and in case it queries a string $q$, respond as follows:
    3.1: If $q$ is of the form $\mathtt{1}y$ with $|y| = |x|$, or if $q$ is of the form $\mathtt{0}w$, return ?.
    3.2: Else, if $q$ is of the form $\mathtt{1}y$, use $c + 1$ queries to $W$ to determine
         membership of $\mathtt{1}y$ in $A_{c+1}$ and answer the query accordingly. By the
         previous if-clause, this does not require us to query the oracle for the
         input string $w$.
    3.3: Else, $q$ is of the form $\mathtt{0}y$ and we answer the query in accordance with
         membership of $y$ in $W$. Again, because of the if-clause in step 3.1,
         we have $y \neq w$ and we do not need to query the oracle for the input
         string $w$.
4: If $\phi(\mathtt{1}x, k)$ returned 0, we infer that $w$ is not a member of $W$ and return 0.
5: Else, return ?.
```

Like in the proof of Theorem 3.5.14, this defines an approximation for W that runs in linear exponential time and only queries strings different from its input. All that is left is to show that its domain is infinite. Because ϕ is a polynomial kernelization, the length of any query q can be bounded by a polynomial of $\text{asInt}(k)$. As a result, the length of a query made by ϕ on input $(\mathtt{1}x, k)$ is less than the length of x for almost all strings x constructed by the approximation. Furthermore, observe that the same value is assigned to x for $c + 1$ different values of w, while ϕ can query at most c of those values of w. Combined, these observations tell us that the if-clause of step 3.1 is only satisfied for finitely many values of x. Lastly, if only finitely many values of x that we consider would be such that $\mathtt{1}x$ is not in A_{c+1}, then W would have exponential density. Hence, $\phi(\mathtt{1}x, k)$ must return 0 for infinitely many values of x, showing that the domain of the approximation outlined above is infinite. □

The above theorem says that, for every c and with respect to the parameterization of (3.18), there is a set in $\mathbf{PKER}^{\text{positive}}_{c\,+\,1\text{ queries}}$ that is not in $\mathbf{PKER}_{c\text{ queries}}$. From this, we obtain two strands of our hierarchy.

3.5.17. Corollary. *For every constant c, we have the two proper inclusions* $\mathbf{PKER}^{\text{positive}}_{c\text{ queries}} \subset \mathbf{PKER}^{\text{positive}}_{c\,+\,1\text{ queries}}$ *and* $\mathbf{PKER}_{c\text{ queries}} \subset \mathbf{PKER}_{c\,+\,1\text{ queries}}$.

The proofs of our separation results so far, Theorem 3.5.10, Theorem 3.5.14, and Theorem 3.5.16, all employed the same proof technique. Yet, each revolved around a different distinguishing feature of Turing kernelizations. The proof of the first of these theorems made use of the fact that Turing kernelizations can be *adaptive*, while truth-table kernelizations cannot. The proof of the second theorem demonstrated that requiring a kernelization to be *positive* is a real restriction of its capabilities. In the last of these proofs, the same was shown for the *number of queries* that we allow a kernelization to make. While we distinguished kernelizations that can make a constant number of queries, the result extends to kernelizations where this number grows unbounded.

3.5.18. Theorem. *With respect to the parameterization η defined by* (3.18)*, there is a set that has a polynomial positive truth-table kernelization, but no polynomial kernelization that makes a number of queries that can be bounded by a constant.*

Proof:
Let W be a set as given by Lemma 3.5.13 and consider the set

$$A = \{\mathtt{0}w \mid w \in W\} \cup \left\{ \mathtt{1}x \;\middle|\; \bigvee_{i \leq |x|} \left(\operatorname{asStr}\left(\frac{|x| \cdot (|x| - 1)}{2} + i \right) \in W \right) \right\}.$$

Apart from the number of terms in the disjunction, this definition is the same as that of the sets A_c in the proof of Theorem 3.5.16. Indeed, the remainder of the proof is mostly the same as well, and we shall only mention where the current proof differs from that of Theorem 3.5.16.

A minor, technical detail is that because of the change in the number of terms in the disjunction, we need a slight change in the approximation for W. In order to obtain a suitable string x for an input w, the beginning of the approximation should be changed as follows.

```
1: Let $n$ and $i$ be the unique values such that $i$ is less than $n$ and we have
   $\operatorname{asInt}(w) = \frac{n \cdot (n-1)}{2} + i$, and set $x$ to $\mathtt{0}^n$. This way, if $w$ is a member of $W$,
   then $\mathtt{1}x$ is a member of $A$.
```

To see why the modified approximation too has an infinite domain, two observations are needed. First, we note that the number of terms in the disjunction defining the redundant part of A outgrows any constant. Therefore, any polynomial kernelization making no more than a constant number of queries can be run to completion for infinitely many inputs $\mathtt{1}x$. Next, suppose that only finitely many of the strings $\mathtt{1}x$ that our approximation for W may consider would not be a member of A. In that case, for some constant c and all n, out of the first $\frac{n \cdot (n-1)}{2}$ strings, at least $n - c$ must be a member of W. As the length of the $\frac{n \cdot (n-1)}{2}$th string is roughly $2 \log n$, this would mean that W has exponential density. Yet, W has at most logarithmic density, thus the polynomial kernelization we run must reject infinitely many of our inputs $\mathtt{1}x$. However, this then means that our approximation for W has an infinite domain. We conclude that A cannot have a polynomial kernelization with respect to η that makes a number of queries that can be bounded by a constant. □

This theorem places polynomial truth-table kernelizations above those making at most a constant number of queries in our hierarchy. Assembling our separation results, we can depict our hierarchy as in Figure 3.7. This depiction includes one additional level that we have not discussed yet, namely that of psize kernelizations. These kernelizations, we shall go into next.

$$\begin{tikzcd}
 & \mathbf{FPT} \\
 & \mathbf{PKER}_{\text{Turing}} \arrow[u, phantom, sloped, "\subset"] \\
 & \mathbf{PKER}_{\text{truth-table}} \arrow[u, phantom, sloped, "\subset"] \\
\mathbf{PKER}^{\text{positive}}_{\text{truth-table}} \arrow[ur, phantom, sloped, "\subset"] & \mathbf{PKER}_{\text{psize}} \arrow[u, phantom, sloped, "\subset"] \\
\mathbf{PKER}^{\text{positive}}_{\text{psize}} \arrow[u, phantom, sloped, "\subset"] \arrow[ur, phantom, sloped, "\subset"] & \vdots \arrow[u, phantom, sloped, "\subset"] \\
\vdots \arrow[u, phantom, sloped, "\subset"] & \mathbf{PKER}_{3\text{ queries}} \arrow[u, phantom, sloped, "\subset"] \\
\mathbf{PKER}^{\text{positive}}_{3\text{ queries}} \arrow[u, phantom, sloped, "\subset"] \arrow[ur, phantom, sloped, "\subset"] & \mathbf{PKER}_{2\text{ queries}} \arrow[u, phantom, sloped, "\subset"] \\
\mathbf{PKER}^{\text{positive}}_{2\text{ queries}} \arrow[u, phantom, sloped, "\subset"] \arrow[ur, phantom, sloped, "\subset"] & \mathbf{PKER}_{1\text{ query}} \arrow[u, phantom, sloped, "\subset"] \\
\mathbf{PKER}^{\text{positive}}_{1\text{ query}} = \mathbf{PKER}_{\text{many–one}} \arrow[u, phantom, sloped, "\subset"] \arrow[ur, phantom, sloped, "\subset"] &
\end{tikzcd}$$

Figure 3.7: A hierarchy of polynomial kernelizations. Inclusions from left to right follow from Theorem 3.5.14. The vertical inclusions on the bottom part follow from Corollary 3.5.17. Those at the top follow from Theorem 3.5.22, Theorem 3.5.18, 3.5.10, and Theorem 3.5.9.

3.5.4 Polynomial Advice

Let us return for a moment to the interpretation of kernelization as preprocessing. Imagine we are interested in a set A and are somehow presented with an instance x of which we might later on need to know whether or not it is a member of A. We may find that storage space is a scarce resource, so we seek for alternatives to recording the entire instance. One way to save on storage space is to decide on membership of x in A and only record the membership decision. This brings the storage cost down from however many bits x is comprised of to just one bit. However, deciding membership in A may be a computationally costly affair. As we may never actually need to know whether the instance is in A or not, this computational cost is unacceptable.

This is where a many–one kernelization comes in. If A has a many–one kernelization with respect to some parameterization η, and η has a polytime-computable parameter estimator, we are in good shape. Running the many–one kernelization is a computationally manageable task and gets us an instance that is equivalent to x with regard to membership in A. The storage space required to store this equivalent instance can be bounded by a function of the parameter value associated with x by the parameter estimator. Especially convenient is the situation where the many–one kernelization is a polynomial many–one kernelization. In that case, the required storage space can be bounded by a polynomial of the numeric value of the parameter. If the length of x is large in comparison to the numeric parameter value, we thus save storage space. Conceptually, we use the kernelization to get rid of the computationally redundant part of x.

The hierarchy of polynomial kernelizations, Figure 3.7, shows that there are sets for which no polynomial many–one kernelization exists. However, these sets may still have, for instance, a polynomial kernelization that makes at most one query. Such a kernelization could equally well be used in our scenario. All we have to do is record the query made by the kernelization and whether or not its membership decision is to be inverted. Doing so again comes with a guarantee on the required storage space that is polynomial in the numeric value of the parameter.

This strategy does not extend to the level of polynomial truth-table kernelizations. It is true that each of the queries made by a polynomial truth-table kernelization is of a polynomially bounded length. However, the number of such queries is only bounded to be polynomial in the length of the input instance. Therefore, a polynomial truth-table kernelization does not guarantee useful preprocessing in terms of storage space requirements. Worse still, even if the number of queries could be bounded by a polynomial in the numeric parameter value, no such polynomial guarantee is available. This is because the size of the truth-table, listing the membership decision for all possible replies of the oracle, is exponential in the number of queries. When the number of queries would be logarithmic in the numeric parameter value, a polynomial truth-table kernelization would be useful for preprocessing. However, we can do better than that.

3.5.19. Definition. A polynomial truth-table kernelization ϕ is a *psize kernelization* if there is a polynomial p such that, on any input (x, k),

- the number of queries made by ϕ is at most $p(\text{asInt}(k))$, and
- the output of ϕ can be expressed as the output of a Boolean circuit of size at most $p(\text{asInt}(k))$ that takes as input the answers of the oracle to the queries.

Moreover, the circuits involved must be uniformly computable from the input instances in polynomial time.

In our scenario where we might, at some point, want to know membership of x in A, a psize kernelization offers a balance of computational cost and storage cost. On the one hand, the computational cost of running the kernelization algorithm is polynomial in the length of the instance. Assuming the thesis by Cobham and Edmonds, this is feasible. The storage cost, on the other hand, is polynomial in the numeric parameter value. Instead of recording x itself, we record which strings are queried by the psize kernelization on input x, along with the circuit that ties them together. This results in recording a polynomial number of strings of polynomial length, together with a circuit of polynomial size. All these polynomials are polynomials of the numeric parameter value, which may be far smaller than the length of x.

The name "psize kernelization" was coined by Witteveen, Bottesch, and Torenvliet [153], but some interesting properties of these kernelizations were already studied by Weller [151, Chapter 5]. In particular, Weller uncovered how the non-adaptive nature of psize kernelization influences the number of queries that are made. We highlight the main result, which presents a trade-off between being non-adaptive and making few queries.

3.5.20. Theorem (Weller [151, Theorem 5.1]). *The following statements about an* **NP***-complete set A, a parameterization η, and a polynomial p are equivalent.*

- *A has a psize kernelization with respect to η that, on any input (x, k), makes at most $p(\text{asInt}(k))$ queries.*
- *A has a polynomial Turing kernelization with respect to η that, on any input (x, k), makes at most $\log p(\text{asInt}(k))$ queries.*

Not every polynomial truth-table kernelization is a psize kernelization. Yet, all polynomial kernelizations that make a number of queries that can be bounded by a constant are psize kernelizations. Indeed, the psize kernelizations fit in between these levels of our hierarchy of polynomial kernelizations.

3.5.21. Theorem. *With respect to the parameterization η defined by* (3.18)*, there is a set that has a positive truth-table kernelization, but no psize kernelization.*

Proof:
The proof of Theorem 3.5.18 actually shows something stronger than Theorem 3.5.18. Let $f\colon \mathbb{N} \to \mathbb{N}$ be a function such that, for all but finitely many values of n, we have $f(n) < n$. The proof of Theorem 3.5.18 can be applied to polynomial kernelizations that, given an input instance $\mathtt{1}x$, make at most $f(|x|)$ queries.

With respect to the parameterization η defined by (3.18), the number of strings a psize kernelization can query on an input of the form $\mathtt{1}x$ is polynomial in $\log|x|$. Any polylogarithmic function f is so that, for all but finitely many values of n, we have $f(n) < n$. Therefore, the proof of Theorem 3.5.18 also proves the current theorem. □

This separates the polynomial truth-table kernelizations from the psize kernelizations. Another adaptation of the proof of Theorem 3.5.18 can be used to separate the psize kernelizations from those making at most a constant number of queries.

3.5.22. Theorem. *With respect to the parameterization η defined by* (3.18)*, there is a set that has a positive psize kernelization, but no polynomial kernelization that makes a number of queries that can be bounded by a constant.*

Proof:
So far, the membership of any string w in W played a role in the definition of the computationally redundant part of our sets. This is, however, not a necessity and the current theorem is more easily proven without such a tidy definition. Instead, let W be a set as given by Lemma 3.5.13 and consider the set

$$A = \{\mathtt{0}w \mid w \in W\} \cup \left\{ \mathtt{1}x \,\middle|\, \bigvee_{i \le \log|x|} \left(\operatorname{asStr}\left(\frac{|x| \cdot (|x|-1)}{2} + i \right) \in W \right) \right\}.$$

This definition is similar to the one used in the proof of Theorem 3.5.18, but the number of terms in the disjunction grows more slowly. As a result, A is a subset of the set used in the proof of Theorem 3.5.18.

Note that for all x and k that satisfy $x \in \eta_k$, where η is as in (3.18), we have $\log|x| \le \operatorname{asInt}(k)$. Thus, a number of queries that is linear in $\operatorname{asInt}(k)$ suffices for a polynomial kernelization for A with respect to η. The answers to the queries are combined disjunctively. Therefore, the dependence of the kernelization on the answers to the queries can be expressed as a circuit of a size that is polynomially bounded in $\operatorname{asInt}(k)$. Indeed, A has a psize kernelization with respect to η.

We show that A does not have a polynomial kernelization that makes a number of queries that can be bounded by a constant in the same way as before. This time, the first step in the approximation for W in the proof of Theorem 3.5.16 requires a little more tweaking. It should be replaced by the following three steps.

```
1: Set $n$ and $i$ to the unique values such that $i$ is less than $n$ and we have
   $\text{asInt}(w) = \frac{n\cdot(n-1)}{2} + i$.
2: If $i$ is at least $\log n$, then there is no string $1x$ such that the definition of $A$
   depends on $w$ and we can only return ?.
3: Set $x$ to $0^n$. This way, if $w$ is a member of $W$, then $1x$ is a member of $A$.
```

As A is a subset of the set used in the proof of Theorem 3.5.18, the same reasoning as used there rules out certain kernelizations for A with respect to η. Specifically, it follows that A does not have a polynomial kernelization that makes a number of queries that can be bounded by a constant. □

Combined, the last two theorems isolate the psize level in our hierarchy, as depicted in Figure 3.7.

So far, our main proof technique has been to create a set with a clear-cut distinction between its hard part and its computationally redundant part. We defined the redundant part of the set so that it could be reduced to the hard part in some specific way. With psize kernelizations, the amount of information about the hard part available to the kernelization could be said to be bounded polynomially. This information can be thought of as *advice* required for deciding the redundant part. In the study of computability with advice, the amount of advice is typically bounded by a function of the length of instances. Most prominently, this is the case with the complexity class $\mathbf{P_{/poly}}$ [13]. With psize kernelizations, however, the bound on the amount of advice is polynomial in the numeric parameter value. This in itself suggests that the two notions of polynomial advice, the complexity classes $\mathbf{P_{/poly}}$ and $\mathbf{PKER}_{\text{psize}}$, need not have much in common. Of course, by Corollary 3.1.4, we already know that the two are different, since $\mathbf{P_{/poly}}$ contains undecidable sets, whereas all sets in **FPT** are decidable. Yet, with respect to the parameterization defined by (3.18), we can characterize the *decidable* sets in $\mathbf{P_{/poly}}$ by the kernelizations they have.

3.5.23. Theorem. *Let η be the parameterization defined by* (3.18)*. The following statements about a set X are equivalent.*

1. *X is decidable and in $\mathbf{P_{/poly}}$.*
2. *There is a decidable set W such that the set*

$$\{0w \mid w \in W\} \cup \{1x \mid x \in X\}$$

has a linear truth-table kernelization with respect to η.

Proof:
$1 \implies 2$. Let ϕ be a polytime decision procedure taking advice of polynomially bounded length that decides membership in X. Note that, for any length n, we

can find an advice string for length n by verifying that it makes ϕ correct on all strings of length n. Consider the set of pairs of numbers

$$W = \{\langle n, i\rangle \mid \text{the } i\text{th bit of the advice string for length } n \text{ is } \texttt{1}\},$$

which is decidable because we can find the advice strings effectively. With this set W, a kernelization can retrieve the advice string required by ϕ for an instance of the form $\texttt{1}x$. For this, it makes a number of queries that can be bounded by a polynomial in $|x|$. It remains to show that the lengths of these queries can be bounded by a linear function in $\log|\texttt{1}x|$. Suppose that the number of queries made on an input of the form $\texttt{1}x$, where x is of length n, is at most n^c. This means that the longest query that is made has length $|\langle n, n^c\rangle|$. Indeed, with the standard encoding of numbers as strings and our pairing function of Definition 2.1.2, this length can be bounded linearly in $\log n$. With that, the bound on the lengths of the queries is proven.

$2 \implies 1$. Suppose we have a linear truth-table kernelization for our disjoint union with respect to η. For this kernelization, there is a constant c such that all queries made on an input of the form $\texttt{1}x$ are of a length bounded by $c \cdot \log|x|$. This means that there are no more than $2^{c\cdot\log|x|} = |x|^c$ strings the kernelization could possibly query. Hence, we can replace the oracle by a string of $|x|^c$ bits where the ith bit is $\texttt{1}$ precisely when we have $\text{asStr}(i) \in W$. As this string is only dependent on the length of x, and not on x itself, it can serve as polynomial advice, showing that X is in $\mathbf{P_{/poly}}$. Additionally, since W is decidable, so is X. □

We remark that the second part of the above proof also shows that we could have replaced truth-table by Turing in the statement of the theorem. A linear truth-table kernelization may query all strings that a linear Turing kernelization could potentially query.

In the context of the above theorem, it is noteworthy that the set constructed in the proof of Theorem 3.5.18 has a linear truth-table kernelization. Thus, the strengthening of that theorem, Theorem 3.5.21, can be strengthened further. It can be strengthened to read that there is a set that has a positive *linear* truth-table kernelization, but no psize kernelization. This, in light of Theorem 3.5.23, points at a difference between the notions of polynomial advice represented by $\mathbf{P_{/poly}}$ and $\mathbf{PKER}_{\text{psize}}$: There exists a decidable set X in $\mathbf{P_{/poly}}$ such that no set W exists such that

$$\{\texttt{0}w \mid w \in W\} \cup \{\texttt{1}x \mid x \in X\}$$

has a psize kernelization with respect to the parameterization η defined by (3.18). However, this does not mean that the notion of advice associated with $\mathbf{PKER}_{\text{psize}}$ is in any way more stringent than that associated with $\mathbf{P_{/poly}}$.

3.5.24. Theorem. *Let η be the parameterization defined by* (3.18)*. There exist decidable sets W and X such that*

- *X is not in $\mathbf{P}_{/\mathbf{poly}}$, and*
- *$\{\mathtt{0}w \mid w \in W\} \cup \{\mathtt{1}x \mid x \in X\}$ has a polynomial many–one kernelization with respect to η.*

Proof:
We denote the string consisting of the first i bits of a string x by prefix(i, x), and define X in terms of W as

$$X = \{x \mid \log|x| \in \mathbb{N} \text{ and } \operatorname{prefix}((\log|x|)^2, x) \in W\}.$$

This definition ensures that $\{\mathtt{0}w \mid w \in W\} \cup \{\mathtt{1}x \mid x \in X\}$ has a polynomial many–one kernelization with respect to η.

Next, we construct W in such a way that X is not in $\mathbf{P}_{/\mathbf{poly}}$. The only instances of W that influence X are those of which the length can be written as a square. Therefore, we may assume that W only contains strings of which the length is a square. Suppose toward a contradiction that X is in $\mathbf{P}_{/\mathbf{poly}}$ and let ϕ be a decision procedure for X that takes an advice string as a second argument. Using ϕ, we define a procedure for deciding membership in W that takes an instance and an advice string as inputs. Given a suitable advice string a for a given instance w, this procedure decides whether w is a member of W.

```
1: If $|w|$ is not a square, return 0.
2: Let $m$ be $2^{\sqrt{|w|}} - |w|$ and set $x$ to $w\mathtt{0}^m$. We have $\operatorname{prefix}((\log|x|)^2, x) = w$,
   and hence $w$ is a member of $W$ precisely when $x$ is a member of $X$.
3: Return $\phi(x, a)$.
```

The running time required by this procedure is polynomial in $|x|$, hence it is in $2^{\mathcal{O}(\sqrt{|w|})}$. By the same token, the length of a suitable advice string for an instance w is also in $2^{\mathcal{O}(\sqrt{|w|})}$. Note that the length of the string x constructed in the procedure is only dependent on the length of w. As a result, any two instances w_1 and w_2 that have the same length share the same advice string. Thus, for any number n, all 2^n strings of length n share the same advice string with a length in $2^{\mathcal{O}(\sqrt{n})}$. This allows us to diagonalize against procedures such as the one we just constructed. Doing so, we contradict the existence of ϕ and thus the assumption that X is in $\mathbf{P}_{/\mathbf{poly}}$.

We construct W in stages, at each stage resolving membership of all instances of a given length. For this, we introduce the characteristic sequence, χ, of W for a given length, n,

$$\chi(n) = (b_w)_{w \in 2^n}, \text{ where we have}$$
$$b_w = \begin{cases} \mathtt{1} & \text{if } w \in W, \\ \mathtt{0} & \text{otherwise.} \end{cases}$$

Our task is thus to determine $\chi(n)$, for all values of n. Let $\phi_1, \phi_2, \phi_3, \ldots$ be an effective enumeration of procedures that take two arguments. We may assume that each such procedure occurs in this list infinitely often. For a procedure ϕ_i, we consider the time-constrained characteristic sequence for a given length n and advice string a, defined by

$$\chi_i(n,a) = (b_w)_{w \in 2^n}, \text{ where we have}$$
$$b_w = \begin{cases} 1 & \text{if } \phi_i(w,a) \text{ returns } 1 \text{ within } 2^{\log|w|\sqrt{|w|}} \text{ steps,} \\ 0 & \text{otherwise.} \end{cases}$$

The bound on the running time, $2^{\log|w|\sqrt{|w|}}$, was chosen so that it grows faster than any function in $2^{\mathcal{O}(\sqrt{|w|})}$. Using these characteristic sequences, we build W by doing the following for each length n in $\mathbb{N}$.

1: If n is not a square, set $\chi(n)$ to

$$(\underbrace{0, 0, \ldots, 0}_{2^n \text{ times}}).$$

As we have noted earlier, we may assume that W only contains strings of which the length is a square.

2: Else, we consider procedure $\phi_{\sqrt{n}}$ for diagonalization:

2.1: We gather the decisions of $\phi_{\sqrt{n}}$ on strings of length n for all advice strings of a limited length and set

$$H = \{\chi_{\sqrt{n}}(n,a) \mid a \in 2^+ \text{ and } |a| < 2^{\log n \sqrt{n}}\}.$$

The bound on the length of the advice string is so that it grows faster than any function in $2^{\mathcal{O}(\sqrt{n})}$.

2.2: Observe that the collection H contains fewer than $2^{2^{\log n\sqrt{n}}}$ characteristic sequences of 2^n items. However, there are 2^{2^n} such sequences possible. Therefore, we can set $\chi(n)$ to a characteristic sequence of 2^n items that is not in H.

This construction of W rules out the existence of a procedure for deciding membership in W that takes advice of length $2^{\mathcal{O}(\sqrt{n})}$ and runs in time $2^{\mathcal{O}(\sqrt{n})}$. □

Note that a polynomial many–one kernelization is also a psize kernelization. Because of this, Theorem 3.5.23 and Theorem 3.5.24 show that the notions of polynomial advice represented by $\mathbf{P}_{/\mathbf{poly}}$ and $\mathbf{PKER}_{\text{psize}}$ are incomparable. Neither notion is more stringent than the other.

3.5.5 Lower Bounds

An immediate consequence of our hierarchy, Figure 3.7, is that not all fixed-parameter tractable problems have polynomial kernelizations. However, with respect to a given parameterization, the (non-)existence of a polynomial kernelization for any particular set may not be easy to establish. The most fruitful program for deriving superpolynomial lower bounds on the size of many–one kernelizations was started by Bodlaender et al. [26]. Building on this work, Dell and Van Melkebeek [39] were able to rule out certain improvements of existing polynomial kernelizations. They do so by establishing, under complexity-theoretic assumptions, a minimum amount of communication that is needed with a generalized oracle. In particular, this means that there cannot be a Turing kernelization of which the sum of the lengths of the queries it makes stays below this minimum. The lower bounds of Dell and Van Melkebeek go beyond psize kernelizations in that the oracle access may be adaptive.

The technique by Bodlaender et al. [26] for obtaining lower bounds on the size of kernelizations does not generalize to Turing kernelizations. However, an extension to psize kernelizations is feasible. We shall adapt a later iteration of the technique, as laid out by Bodlaender, Jansen, and Kratsch [24, Section 3]. For a more complete background of the technique, we refer to the survey by Kratsch [97], and the textbook by Fomin et al. [58].

Central to the lower bounds engine are two similar-looking classifications of instance aggregation. The first of these does not involve a parameterization.

3.5.25. Definition. A *weak* AND-*distillation* (*weak* OR-*distillation*) of a set A into a set B is a procedure that

- receives as input any finite sequence of strings $x_1, x_2, \dots, x_t$,
- uses time polynomial in $\sum_{i \le t} |x_i|$, and
- outputs a string y such that
  - we have $y \in B$ if and only if for all (any) i we have $x_i \in A$, and
  - $|y|$ is bounded by a polynomial in $\max_{i \le t} |x_i|$.

Note how the length of the output of a distillation is bounded by a polynomial in the *maximum* length of its inputs and not by the sum of the input lengths. Because of this bound, having a weak OR-distillation can be seen as a generalization of being p-selective, Definition 3.1.22.

Originally, distillations where considered where the target set B was equal to A, hence the 'weak' in this more general definition. Similarly, a parameterized counterpart to distillation, *composition*, was originally defined with the same parameterized set as source and target [26]. This was later generalized [24], with the additional change that the source set was considered outside the context of any parameterization. In the framework of Downey and Fellows, this change is significant, because of the distinction between non-parameterized and parameterized sets. In our framework, parameterizations are distinct entities and the change is not as significant.

3.5.26. Definition. An AND-*cross-composition* (OR-*cross-composition*) with respect to a parameterization η of a set A into a set B is a procedure that

- receives as input any finite sequence of strings $x_1, x_2, \dots, x_t$,
- uses time polynomial in $\sum_{i \le t} |x_i|$, and
- outputs a string y and parameter value k satisfying $y \in \eta_k$, such that
  - we have $y \in B$ if and only if for all (any) i we have $x_i \in A$, and
  - $\mathrm{asInt}(k)$ is bounded by a polynomial in $\max_{i \le t} |x_i| + \log t$.

Contrary to the definition of cross-composition by Bodlaender, Jansen, and Kratsch [24], we make no mention of a "polynomial equivalence relation". We have left this aspect out to keep our presentation focused. Nevertheless, those familiar with cross-composition will find no trouble reintroducing the equivalence relation in the upcoming theorem, Theorem 3.5.27.

With cross-composition, a bound is placed on the *parameter value* of the output of the procedure, rather than on the *length* of the output instance. Conceptually, a bound of this kind makes sense as parameter values serve as a measure of the computational hardness of instances. Thus, a set has a cross-composition when instances can be combined efficiently, without an increase in computational hardness.

It was shown by Bodlaender et al. [26] and Bodlaender, Jansen, and Kratsch [24] that polynomial many–one kernelizations tie the two ways of aggregating instances together. We find that the same is true of psize kernelizations. The relevance of the following theorem lies in its use in ruling out the existence of a psize kernelization for certain sets and parameterizations.

3.5.27. Theorem. *Let A be a set that has an* AND-*cross-composition (*OR-*cross-composition) into a set B with respect to a parameterization η. If B has a psize kernelization with respect to η, then there is a set C into which A has a weak* AND-*distillation (weak* OR-*distillation).*

Proof:
We shall track the proof of Bodlaender, Jansen, and Kratsch [24, Theorem 3.4], which in turn follows the approach of Bodlaender et al. [26, Lemma 2]. Assuming B has a psize kernelization, we shall define a weak distillation for A. The target of this weak distillation will be the set

$$\textsc{Circuit}(B) = \{\langle \phi, (q_i)_{i \le r} \rangle \mid r \in \mathbb{N} \text{ and } \phi \text{ is a circuit with } r \text{ inputs, accepting } (q_1 \in B, q_2 \in B, \dots, q_r \in B)\}.$$

That is, Circuit(B) is the set of pairs of a circuit ϕ and a finite sequence of strings $(q_1, q_2, \dots, q_r)$, such that ϕ outputs 1 when fed $(q_1 \in B, q_2 \in B, \dots, q_r \in B)$.

We may assume that the input sequence, $(x_1, x_2, \dots, x_t)$, of a distillation does not contain any duplicates. Indeed, duplicates only make the task of a distillation easier. If we define $s = \max_{i \leq t} |x_i|$, we may therefore also assume that we have $t \leq 2^s$, and hence that we have $\log t \leq s$. Our weak distillation of A into Circuit(B) proceeds as follows on input $(x_1, x_2, \dots, x_t)$.

1: Run the cross-composition on $(x_1, x_2, \dots, x_t)$ to obtain a string y and parameter value k satisfying $y \in \eta_k$. The string y is so that it is a member of B precisely when all (or any, depending on the type of cross-composition) input instances are in A. Furthermore, by our assumption on $\log t$, we find that $\mathrm{asInt}(k)$ can be bounded by a polynomial in s.

2: Run the psize kernelization for B on y and k to obtain a circuit ϕ and a finite sequence of query strings $q_1, q_2, \dots, q_r$. By definition of a psize kernelization, the size of the circuit can be bounded by a polynomial in $\mathrm{asInt}(k)$, and therefore also by a polynomial in s. Likewise, both the number of query strings and the length of these strings can be bounded by a polynomial in s.

3: Return $\langle \phi, (q_i)_{i \leq r} \rangle$. By our previous observations, the length of this pair can be bounded by a polynomial in s. We remark that if we did take into account a polynomial equivalence relation in the style of Bodlaender, Jansen, and Kratsch [24], we would at this point have multiple circuits and inputs. These circuits would then need to be combined in a way determined by the type of cross-composition.

The output of this procedure meets the requirements of a distillation. Additionally, the time required for each step can be bounded by a polynomial in $\sum_{i \leq t} |x_i|$, which also bounds the length of the output at each step. Therefore, the above procedure indeed defines a weak distillation of A into Circuit(B). □

In light of the work of Bodlaender, Jansen, and Kratsch [24], we note two generalizations of the above theorem that can be made. First, as mentioned before, the definition of a cross-composition can be weakened somewhat. It need not be possible to aggregate just any finite set of strings. Instead, it is sufficient if we can quickly partition strings $x_1, x_2, \dots, x_t$ into a number of subsets that is polynomial in the length of the longest string. Each of these subsets is then aggregated separately. This generalization is formalized by means of "polynomial equivalence relations" by Bodlaender, Jansen, and Kratsch [24]. The second generalization of the above theorem that can be made, is in the requirement of the existence of a psize kernelization. In the proof, it is not necessary that the queries made by the psize kernelization are queries about membership in B. Put differently, the psize kernelization may as well have been a reduction from B to a different set. Such a relaxed version of kernelization is called *compression* by Bodlaender, Jansen, and Kratsch [24].

When the framework behind Theorem 3.5.27 was first published [26], it was immediately linked to the fact that **NP**-hard sets are unlikely to have weak distillations. It was quickly shown by Fortnow and Santhanam [60] that, unless we have $\mathbf{NP} \subseteq \mathbf{coNP}_{/\mathbf{poly}}$, no **NP**-hard set admits a weak OR-distillation. The inclusion of **NP** in $\mathbf{coNP}_{/\mathbf{poly}}$ is deemed unlikely, as it implies a collapse of the polynomial hierarchy to its third level [155]. The history of the result by Fortnow and Santhanam can be traced back to results on p-selective sets by Selman [136] and Ko [92]. We remark that in a similar fashion, Hemaspaandra et al. [81] had shown that sets with *nondeterministic* selector functions are in $(\mathbf{NP} \cap \mathbf{coNP})_{/\mathbf{poly}}$.

A result like that of Fortnow and Santhanam for weak AND-distillations was initially left as a conjecture [26]. This conjecture was proven by Drucker [47], who showed that **NP**-hard sets also do not admit weak AND-distillations unless we have $\mathbf{NP} \subseteq \mathbf{coNP}_{/\mathbf{poly}}$. These results translate into a technique for obtaining lower bounds on the size of kernelizations.

3.5.28. Corollary. *If an* **NP***-hard language has an* AND*-cross-composition or an* OR*-cross-composition into a set A with respect to a parameterization η, then A does not have a psize kernelization with respect to η unless we have* $\mathbf{NP} \subseteq \mathbf{coNP}_{/\mathbf{poly}}$.

Many **NP**-hard sets can be shown to have a cross-composition to themselves with respect to some natural parameterization [24, 58]. Accordingly, our hierarchy of polynomial kernelization, Figure 3.7, is not merely relevant to synthetic problems such as the ones used in our proofs. Of many natural problems, the place in the hierarchy is lower bounded too, adding to the appeal of the hierarchy.

3.5.29. Example. Recall the definition of Clique from Example 1.2.2,

$$\text{Clique} = \{(G, l) \mid \text{there is a set of at least } l \text{ vertices of the graph } G \text{ in which each pair of vertices is connected by an edge}\}.$$

In Example 2.2.10, we have seen that Clique is in **FPT** with the vertex cover parameterization. Here, the vertex cover parameterization was defined as

$$\eta = (\{(G, l) \mid G \text{ has a vertex cover of at most asInt}(k) \text{ vertices}\})_{k \in 2^+}.$$

Note that the second component of the instances, l, is ignored in the definition of η. It is only there to align η with Clique.

As shown by Fomin et al. [58, Section 17.3.2], there is an OR-cross composition with respect to the vertex cover parameterization of Clique into itself. Thus, by corollary 3.5.28, there is no psize kernelization for Clique with respect to the vertex cover parameterization. This can be contrasted with Example 3.5.8, which shows that VertexCover has a polynomial many–one kernelization.

Using the technique of Bodlaender et al., we are unable to rule out any *adaptive* kernelizations. This is because of the dependence of the technique on distillations, which do not allow for any adaptive behavior. Sometimes, however, it is possible to sidestep distillations and directly derive $\mathbf{NP} \subseteq \mathbf{coNP}_{/\mathbf{poly}}$ from the assumed existence of some form of kernelization. This approach is taken by Dell and Van Melkebeek in a variant of Corollary 3.5.28 [39, Lemma 1]. They too consider the situation where an **NP**-hard language has an OR-cross-composition into a set A with respect to a parameterization η. Assuming we do not have $\mathbf{NP} \subseteq \mathbf{coNP}_{/\mathbf{poly}}$, Dell and Van Melkebeek rule out a more general class of parameterizations than ruled out by Corollary 3.5.28. Specifically, A does not have a polynomial kernelization with respect to η of which the number of queries made can be bounded by a polynomial in the numeric parameter value. This includes adaptive kernelizations. The proof of Dell and Van Melkebeek only applies to OR-cross-composition and a similar result for AND-cross-composition remains open. It appears disjunctive instance aggregation repeatedly yields results more easily than conjunctive instance aggregation.

Chapter 4

Conclusion

The main results of this thesis were presented not only as formal theorems, but also as informal slogans. A listing of all the slogans is provided at the front of this thesis. However, there is more to this thesis than just those results.

We started this thesis with some questions and observations in Section 1.1 and Section 1.2, respectively. In the current chapter, we shall revisit the topics of those sections. First, in Section 4.1, we shall use our parameterized framework to finally provide some answer to the question “what is the size of a cube?” Following that, in Section 4.2, we assemble a historical timeline of parameterized complexity theory. In this timeline, we have included the milestones in the development of parameterized analysis of complexity, as far as we encountered them in this thesis. Lastly, in Section 4.3, we briefly look ahead and point out some possible future directions for the parameterized analysis of complexity. This includes further forms of complexity that may be analyzed using our framework, as well as some ideas for applications. Additionally, we have listed in this section a few major questions that have remained unanswered in this thesis.

4.1 The Size of a Cube

The dimensions of a physical cube-shaped object determine how large a pocket, box, or hangar needs to be in order to contain the object. For an abstract cube, we can look at how long a description of it is, in accordance to some description method. This length is also a dimension in the sense that it determines how much storage space we need to contain the abstract cube. In Section 3.4.5 and Section 3.4.6 we have used parameterizations as the basis of description methods. These methods start by specifying a property, as represented by a parameter value, of the object that is being described. Thus, the length of a specification can serve as a proxy of the metrics we have considered in Section 1.1.

4.1.1. Example. Suppose we use the number of vertices in a graph as the basis of our description method for graphs. Because, for any fixed number n, there are only finitely many graphs with n vertices, we use an encoding where all n-vertex graphs are equally likely. This is exactly what is achieved by the adjacency-matrix representation of graphs [35].

Alternatively, we could use the number of edges as the basis of our description method for graphs. In this case, all graphs that have some fixed number of edges cannot be equally likely since, for each n, there are infinitely many graphs with n edges. However, as demonstrated in Section 3.4.5, solutions to this problem exist. An example of an edge-centric representation of graphs is the adjacency-list representation, which basically presents a graph as a list of its edges [35].

The second description method is most useful when the graphs we are working with have relatively few edges. With the graph for the dodecahedron, Figure 1.1c, for instance, it makes sense to use an adjacency-list representation. Recording for each possible combination of vertices whether there is an edge connecting them is wasteful for this graph. By contrast, the graph for the tetrahedron, Figure 1.1a, has as many edges as are possible, and an adjacency-matrix representation may be preferable.

What metric most faithfully represents our notion of size of objects depends on the context in which we are dealing with the objects. Correspondingly, some ways of describing a graph are more accommodating to answering a given question about a graph than others. However, when the size of an object is measured by the length of a description in accordance with some description method, there exists a universal measure. Kolmogorov complexity is the length corresponding to the most general description method available. In this description method, any property of an object may be used in a description, as long as the property itself is specified as well. As a consequence, we can speak of *the* Kolmogorov complexity of a graph, without having to specify a description method. Note that this does assume that we restrict our attention to effective description methods. Contrary to the metrics we have considered in Section 1.1, the Kolmogorov complexity of a graph is not computable.

We remark that Kolmogorov complexity tells us something about the efficiency of certain description methods. In particular, it tells us something about description methods with which we can effectively get from an object to its description and back. For every such description method, there are objects of which that description method is the best way to describe the object. With this, we mean that the description method assigns to the object a code that is as short as possible. Specifically, the length of the code equals the Kolmogorov complexity of the object. These objects for which the description method is optimal are the objects that are described by incompressible strings. In that sense, no effective description method is more "blessed" than any other.

Besides the fact that Kolmogorov complexity is not computable, there are other objections against its use as a measure of the size of objects. The most obvious objection is that Kolmogorov complexity does not take computation time into account. If decoding the description of an object takes prohibitively long, the description is not very useful. We may want to require that decoding the description of an object should somehow run in polynomial time, relying on the thesis by Cobham and Edmond. Unfortunately, we cannot do so unless we are willing to specify a specific polynomial. This is a consequence of the fact that Kolmogorov complexity is a nonuniform notion in the sense that it treats each object separately. As a procedure, a description of an object takes no input and produces only a single output. We can evaluate a specific function in, say, the length of the output, but this will only give us a constant. Without fixing a specific polynomial, there is no way to classify a procedure that does not take any input as running in polynomial time.

Another source of objections against the use of Kolmogorov complexity is the fact that it is defined up to an additive constant. We have seen in Section 1.2.3 that we should include as much cultural context in our model of computation as possible. If we fail to do so, then the additive constant that is involved with Kolmogorov complexity may overshadow the actual complexity of the objects we work with. At the same time, the cultural context of human perception of complexity is continually evolving. As a result, our notion of complexity may change and the additive constant with respect to any fixed model of computation may keep increasing without bounds.

The best notion of size, then, depends on the context it is to be used in. In turn, an analysis of any form of complexity in terms of the size of objects should be explicit about how the size of objects is measured. Our framework for the parameterized analysis of complexity offers a mathematical model to do just that. Central to this framework is the notion of a *parameterization*, a family of sets indexed by parameter values. In Chapter 3, we have seen that the framework is able to deal with a multitude of complexities, both algorithmic and computational in nature. The framework offers five levels of abstraction for the analysis of complexity.

Instance At the instance level, a parameterization η links an instance x to a set of parameter values $\{k \mid x \in \eta_k\}$. When expressing complexity in terms of parameter values, we strike a balance between being too specific and not being specific enough. What we mean by this can be demonstrated by considering the running time of some procedure as a form of complexity. The running time of a procedure can be computed as a function of its input by running the procedure and clocking the time it takes. This is an overly specific way of measuring the running-time complexity of instances, as it does not generalize in any way. On the other hand, a worst-case running time expressed as a function of the length of inputs is blind for more input-specific behavior of complexity.

Slice The starting point of a parameterized analysis of complexity is grouping instances that are somehow comparable. By definition, a parameterization is a collection of sets of instances. This way, parameterizations enable the comparison of complexity between instances. Additionally, by not looking at instances in isolation, but collectively, this opens the door for the consideration of uniformity constraints.

Parameterization When even more uniformity is required, we turn to collections of slices. This is the level of abstraction at which parameterized complexity has been most successful thus far. It allows us to study how complexity scales as a function of parameter values.

Filter At the level of individual filters, we step away from specific parameterizations and thus from any specific context in which the size of objects is measured. This higher level of abstraction is appropriate for the study of what the parameterizations that relate to the complexity of a certain problem are like. At this level, all notions of length that are relevant to a certain problem are bundled and studied as a whole.

Filters The level of filters, plural, is where we find out how the form of complexity we are analyzing ties in with specific computational problems. Each parameterization, or, for that matter, each notion of length, represents a distribution of complexity. In turn, a filter of parameterizations represents all applicable distributions of complexity. The relation between such filters then tells us how specific these distributions are to the computational problems we are analyzing.

In this thesis, we have looked at various forms of complexity and analyzed them at various levels of abstraction. At the same time, these analyses provided insight into the details of the framework, and in particular into the class of fixed-parameter tractable sets. Section 3.1, where we studied computability as a notion of complexity, revealed to what sets a notion of fixed-parameter tractability may apply.

In particular, fixed-parameter tractability is concerned with decidable sets, or, if we stretch our definitions, with sets at the Δ_2^0 level of the arithmetical hierarchy. Next, in Section 3.2, we found that most sets that interest us have no optimal parameterizations in relation to fixed-parameter tractability. In Section 3.3, we obtained results on two levels of abstraction. On the level of filters, plural, we looked at sets with the same complexity profile from a fixed-parameter tractability point of view. We found that such sets might alternatively be characterized as sets of which the symmetric difference is in **P**. On the level of instances, we looked at the similarity of parameterized complexity with measures of algorithmic complexity. We found that the measure of complexity embodied by a parameterization is more similar to instance complexity than to Kolmogorov complexity. More so than these two measures of algorithmic complexity, parameterizations offer opportunities for the consideration of uniformity constraints. Another comparison with an existing framework was made in Section 3.4. Here, parameterizations were related to statistical model classes. As it turns out, parameter estimation can be interpreted both in the context of computational complexity, and in the context of statistics. These two different interpretation allow insights from both contexts to be shared. Lastly, in Section 3.5, we returned to a complexity-theoretic study of the complexity class of fixed-parameter tractability, **FPT**. This class is closed under several related notions of reducibility known collectively as kernelization. By looking at refined version of these reducibilities, polynomial kernelization, we uncovered a proper hierarchy inside **FPT**, shown in Figure 3.7.

What, then, is the size of a cube? If the size of a cube relates to how convenient it is to work with compared to other shapes, the size necessarily depends on what we try to do with the cube. However, even if we have a specific application in mind, there may be multiple ways to measure the cube.

Our goal may be to decide membership in a given set and convenience may be determined purely by the time needed to do so. In that case, the ways to measure the cube are given by the parameterizations with which our decision problem is in **FPT**. As we have seen in Section 3.2, there may thus be no definitive way to measure the cube. Perhaps the best we can do, mathematically, is to consider the filter of all these parameterizations. At least, this filter holds information on all the ways the complexity of the cube can relate to the complexity of other shapes.

Convenience could be determined by more than the time needed to decide membership in a given set. It could be that our use case desires that a certain polynomial kernelization exists for our decision problem. In that case, following Section 3.5, the ways to measure the cube are given by a subset of the parameterizations with which our decision problem is in **FPT**.

Whatever parameterization we end up with to measure the cube, we know from Section 3.4 that we are not measuring only computational complexity. Every measure given by a parameterization is also a measure of useful information.

4.2 Historical Encounters

In computer science, there is a long, but not extremely visible, tradition of bringing together different notions of complexity. Our framework for the analysis of complexity is a continuation of that tradition and aims for an explicit unification of complexity theories. We feel that using parameters as the basis of a model of complexity provides us with sufficient generality to do so. The parameterized approach to complexity theory is not new and has a history even before the first parameterized complexity classes were defined by Downey and Fellows [41]. In this thesis, we have come across many incarnations of parameterized complexity theory before it was “parameterized complexity theory”.

1944, *Post [123].* In the context of decidability, Post noted that even sets that are not decidable may have infinite subsets that are. This is a form of parameterized analysis of complexity at the level of slices. While a set as a whole may be complex, in this case meaning undecidable, there may be subsets, slices, that are not complex.

1965, *Putnam [124] and Gold [66].* Staying with undecidability as a notion of complexity, we mention the work of Putnam and Gold. They noted that some undecidable sets can be approximated by decision procedures that are allowed to change their mind. The more often a decision procedure changes its mind on an instance, the more complex we say the instance is. In that sense, this form of parameterized analysis of complexity happens on the level of instances. However, given the focus on decision procedures that are allowed to change their mind, the analysis can also be placed at the level of parameterizations.

1968, *Jockusch [88].* Next to looking at how complex, in this case undecidable, instances are in isolation, it is possible to compare the undecidability of instances. The set of instances that can be reduced to a given instance x form a slice of instances that are no more complex than x. This way, it is possible to reveal redundancy, even in undecidable sets. The work of Jockusch in this area can be thought of as a very early study of kernelization.

1974, *Flajolet and Steyaert [55].* Parameterized reasoning in computational complexity theory, as opposed to computability theory, can be traced back at least to the work of Flajolet and Steyaert. They asked the questions that Post asked in 1944 again, but this time in the context of computational complexity. To our knowledge, this marks the first time that the polynomial-time decidable subsets of computationally complex sets were studied.

1975, *Lynch [102].* The next step after considering polynomial-time decidable subsets of computationally complex sets was to look at all such subsets at once. By doing so, Lynch pushed the parameterized analysis of computational complexity from the level of slices to the level of parameterizations.

1979, *Selman [136] and Meyer and Paterson [106].* The strengths of various notions of reducibility in the presence of a polynomial running-time bound were studied by Selman and by Meyer and Paterson. Results in this setting build on the work of Jockusch from 1968. By the addition of the polynomial running-time bound, similarities with the study of kernelization are reinforced.

1979, *Garey and Johnson [63].* The first to investigate running times of decision procedures as a function of not only the length of instances, but also of natural parameters were Garey and Johnson. Their work lacks proper definitions and, in general, a formal framework. Nevertheless, the beginnings of a parameterized theory of computational complexity at the level of parameterizations are recognizable.

1983, *Hartmanis [74].* One of the earliest attempts at bringing together algorithmic complexity and computational complexity comes from Hartmanis. The study of the effect of algorithmic complexity on computational complexity takes place at the level of parameterizations. Nevertheless, algorithmic complexity remains a nonuniform notion and therefore this analysis of complexity can also be placed at the level of instances.

1985, *Orponen, Russo, and Schöning [118].* Looking at the polynomial-time decidable subsets of a computationally complex set A may provide new insights into the complexity of A. Some sets can be approximated reasonably well by their polynomial-time decidable subsets. For others, such approximations necessarily proceed in small steps, adding only finitely many instances at a time. The research in this area by Orponen, Russo, and Schöning has taken the parameterized analysis of complexity to the level of individual filters.

1986, *Orponen [116].* The collections of polynomial-time decidable subsets of computationally complex sets showcase some algebraic structure with regard to set-theoretic operations. It was noted by Orponen that there are only very few possibilities for what the structure that emerges from a set looks like. The investigations of what structures are possible can be thought of as a first analysis at the level of filters, plural, and the relations between them.

1992, *Downey and Fellows [41].* The first formal framework for the parameterized analysis of computational complexity, including definitions of complexity classes, was given by Downey and Fellows. This framework marks the start of modern parameterized complexity theory. We observe that the ideas by Flajolet and Steyaert predate this framework some eighteen years.

Subsequent developments regarding the framework were discussed in Section 2.2.

4.3 Future Encounters

There is still much unexplored territory in the parameterized analysis of complexity. Directions in which research can be taken range from purely theoretical to highly applied. We shall list a few directions in which the work in this thesis can be continued.

Other Forms of Complexity

In this thesis, the primary computational resource we have looked at is running time. In light of the sequential computation thesis, this is a good resource to look at because it is fairly independent of the model of computation. Similarly, we could have looked at the space usage of computations and indeed, many of our techniques can be applied to space usage with minimal change [152]. In practice, however, we sometimes experience forms of complexity that are more strongly dependent on a model of computation.

One such form of complexity came up in a private communication with Tom van der Zanden about his work on computing treewidth on the GPU [157]. When programming for the GPU, the best results are obtained if the same code can be executed on multiple parts of the data in parallel. We experience instances on which computation can be parallelized efficiently as not very complex. This suggests trying to bound the extent to which computation on an instance can be parallelized as a function of parameter values. As stated, this is not at all precise, yet it may serve as inspiration for future work on something akin to fixed-parameter parallel computation. We note that Weller [151] already performed a parameterized analysis of parallel computation, using kernelization.

Another form of complexity that is encountered in practice is energy consumption. Perhaps surprisingly, not all operations on data require energy to be performed. As it turns out, no energy expenditure is necessary if the operations are logically reversible [100, Section 8.2]. This suggests that we try to bound the number of irreversible steps taken in some computation as a function of parameter values. Both for this notion of complexity and the previous one, the most interesting part of a future study may be finding relevant parameterizations.

Irreversible computation can be simulated in a reversible way. A way to do so that incurs only a linear overhead in either running time or space usage was presented by Buhrman, Tromp, and Vitányi [31]. Because of these two extremes, a linear overhead has become the benchmark in the reversible simulation of irreversible computation. Thus, a simulation with a linear overhead in terms of running time is time-efficient, and a simulation with a linear overhead in space usage is space-efficient. Currently, no reversible simulation of irreversible computation that is efficient both in terms of running time and in terms of space usage is known. Of interest, then, is what the precise trade-off between overhead in running time and overhead in space usage is. Our parameterized framework may provide a decent means of analyzing such trade-offs. Parameter values may be interpreted as pairs $\langle t, s\rangle$, where t serves as a bound on the running time of a procedure while the space usage is bounded in s. In turn, a parameterization records, for each instance x, the pairs $\langle t, s\rangle$ for which the procedure completes its computation on input x. In other words, for each instance, the parameterization represents the frontier corresponding to the trade-off between the two resource bounds.

Again, the analysis of frontiers of multiple resource bounds need not be limited to running time and space usage. For example, in a probabilistic model of computation, we could look at the trade-off between running time and the use of randomness. This provides another interface between computational complexity and algorithmic complexity.

Design Patterns

The results in this thesis are of a theoretical nature. Their implications, however, may be practical and can sometimes be framed as templates for solving recurring problems. Such templates are known as *design patterns*. Some design patterns were mentioned in Section 3.5, where the spread of a computational workload over a computing system was discussed. A more straightforward application of kernelization, not involving system architectures made up of multiple units capable of computing, is available too. This application concerns the use of kernelizations in the design of algorithms. A kernelization is in essence a recursive algorithm. Queries to the oracle can be treated as recursive invocations. This is true not only of many–one kernelization, where the recursion can be limited to tail-recursion, but of Turing kernelization in general. Of course, some provision has to be put in place for the case where on some input x the kernelization queries x itself, perhaps indirectly. This may happen when x meets the size bound of the kernelization.

A common technique to amortize the cost of multiple invocations of a recursive algorithm is *memoization*, which is a form of dynamic programming [see 35, Chapter 16]. With memoization, the output of an algorithm is stored with the input in a lookup table so that, for any input, the algorithm needs to be run at most once. The size of a lookup table can get rather large and care must be taken that its benefits outweigh this additional storage requirement. With kernelizations, we can get a grip on the size of a lookup table. To do so, we no longer blindly store all input and output values that we encounter in a lookup table. Instead, we only store those values when the input meets the size bound of the kernelization. This way, the size of the table is kept in check without sacrificing the polynomial running time of the kernelization.

We remark that dynamic programming is also used in parameterized complexity theory for the design of parameterized decision procedures [see 113, Chapter 9]. For this case, the technique is used to realize a bound on the running time that is as required to show that a set is fixed-parameter tractable.

In Section 3.4.6, we mentioned another way in which our parameterized analysis leads to a design pattern. Here, the observation was that a two-part encoding of data may be compressing while some operations on the data can be performed without decompressing first. Moreover, these two-part data structures may be useful for benchmarking purposes. In benchmarking, we want to sample random instances from realistic distributions. The relationship between parameterizations and realistic distributions is discussed in Section 3.4.5 and in particular in Example 3.4.32. As a result, with our two-part data structures, random sampling from realistic distributions simply means initializing data structures uniformly at random.

Open Problems

Several open problems have been identified in this thesis. We repeat them here.

- Regarding the use of parameterizations for lossy compression, an oddity is observed in Section 3.4.6 on page 180. The performance of lossy compression formats is often measured as a compression ratio. This implies that, on average, a constant percentage of bits is saved by compression, regardless of the length of the input. By contrast, parameterizations typically promise a compression performance that increases with the length of the input.
- We have shown in Theorem 3.3.20 that algorithmic complexity is a source of computational complexity. Whether or not the converse is true as well we do not know. The best result available in this direction is Theorem 3.3.29, which goes back to Hartmanis [74, Theorem 14].
- Sets in **P** are fixed-parameter tractable with any parameterization. In fact, as mentioned in Section 3.3.2, a set A is in **P** precisely when $\mathcal{F}_{\mathbf{FPT}}(A)$ equals $\mathcal{L}_{\mathbf{FPT}}$. Also in that section, we noted that **P** is a subgroup of the commutative group of decidable sets with the symmetric-difference operator, $\triangle$. In Conjecture 3.3.12, we postulate that $\mathcal{F}_{\mathbf{FPT}}$ preserves this group structure and is a group homomorphism. Regarding the nonuniform case, we know by Theorem 3.3.11 that $\mathcal{F}_{\mathbf{FPT}_{\mathbf{nu}}}$ is *not* a group homomorphism.
- Many sets do not have optimal parameterizations as far as **FPT** is concerned. This was the result of Theorem 3.2.41. Recall that the difference between **FPT** and ***X*P** is that in **FPT** the degree of the polynomial in the running time bound is held fixed. At the same time, no specific value for the degree is prescribed and in the proof of Theorem 3.2.41, this degree is increased. Possibly, the theorem would be false if we require the degree to stay the same. Stated formally, there may be a constant c and a set A without a maximal $\mathcal{O}(n^c)$-segment such that $\mathcal{F}_{\boldsymbol{X}\mathbf{TIME}(n^c)}(A)$ is principal. We shall leave this as an open problem, but note that $\mathcal{F}_{\boldsymbol{X}\mathbf{TIME}(n^c)}$ is indeed a filter. This can be shown completely analogously to Theorem 3.2.27.

Notation

The following notation is invented in this thesis. In these definitions, n s a natural number, x a binary string, k a parameter value (also a string), $\eta = (\eta_k)_{k \in 2^+}$ a parameterization, A a set, and $\boldsymbol{C}$ a parameterized complexity class.

$$\begin{aligned}
\mathcal{F}_{\boldsymbol{C}}(A) &= \{\eta \mid (A, \eta) \in \boldsymbol{C}\} \\
\mathcal{L}_{\boldsymbol{C}} &= \{\eta \mid \exists A\colon (A, \eta) \in \boldsymbol{C}\} \\
\mu_\eta(x) &= \min\{|k| \mid x \in \eta_k\} \\
\mathrm{M}_\eta(n) &= \max\{\mu_\eta(x) \mid x \in 2^n\} \\
\mathrm{N}_\eta(n, k) &= \text{the number of elements in } 2^n \cap \eta_k
\end{aligned}$$

Bibliography

[1] Samson Abramsky and Achim Jung. “Domain Theory”. In: *Handbook of Logic in Computer Science*. Ed. by Samson Abramsky, Dov M. Gabbay, and Tom S.E. Maibaum. Vol. 3. Clarendon Press, 1994, pp. 1–168.

[2] Pieter Adriaans. *Facticity as the Amount of Self-Descriptive Information in a Data Set*. 2012. arXiv: `1203.2245 [cs.IT]`.

[3] Pieter Adriaans and Johan van Benthem. *Handbook of the Philosophy of Science*. Vol. 8: *Philosophy of Information*. Ed. by Dov M. Gabbay, Paul Thagard, and John Woods. Elsevier, 2008.

[4] Rachit Agarwal, Anurag Khandelwal, and Ion Stoica. “Succinct: Enabling Queries on Compressed Data”. In: *12th USENIX Symposium on Networked Systems Design and Implementation*. 2015, pp. 337–350.

[5] Manindra Agrawal and Osamu Watanabe. “One-Way Functions and the Isomorphism Conjecture”. In: *Electronic Colloquium on Computational Complexity*. 19. 2009.

[6] Eric W. Allender. “Isomorphisms and 1-L Reductions”. In: *Journal of Computer and System Sciences* 36.3 (1988), pp. 336–350.

[7] Eric W. Allender. “The Complexity of Sparse Sets in **P**”. In: *Structure in Complexity Theory*. 1986, pp. 1–11.

[8] Eric W. Allender and Roy S. Rubinstein. “**P**-Printable Sets”. In: *SIAM Journal on Computing* 17.6 (1988), pp. 1193–1202.

[9] Klaus Ambos-Spies, Klaus Weihrauch, and Xizhong Zheng. “Weakly Computable Real Numbers”. In: *Journal of Complexity* 16.4 (2000), pp. 676–690.

[10] Luís Antunes and Lance Fortnow. “Sophistication Revisited”. In: *Theory of Computing Systems* 45.1 (2009), pp. 150–161.

[11] Luís Antunes et al. “Sophistication vs Logical Depth”. In: *Theory of Computing Systems* 60.2 (2017), pp. 280–298.

[12] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. “Complexity of Finding Embeddings in a k-Tree”. In: *SIAM Journal on Algebraic Discrete Methods* 8.2 (1987), pp. 277–284.

[13] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[14] Marat Arslanov. “Degree Structures in Local Degree Theory”. In: *Complexity, Logic, and Recursion Theory*. Ed. by Andrea Sorbi. CRC Press, 1997, pp. 49–74.

[15] José Luis Balcázar and Ronald V. Book. “Sets With Small Generalized Kolmogorov Complexity”. In: *Acta Informatica* 23.6 (1986), pp. 679–688.

[16] José Luis Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity I*. 2nd ed. Springer, 1995.

[17] José Luis Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity II*. Springer, 1990.

[18] José Luis Balcázar and Uwe Schöning. “Bi-Immune Sets for Complexity Classes”. In: *Theory of Computing Systems* 18.1 (1985), pp. 1–10.

[19] Leonard Berman and Juris Hartmanis. “On Isomorphisms and Density of **NP** and Other Complete Sets”. In: *SIAM Journal on Computing* 6.2 (1977), pp. 305–322.

[20] Nick Berry. *PIN Number Analysis*. Sept. 2012. URL: `http://www.datagenetics.com/blog/september32012/`.

[21] Peter Bloem, Steven de Rooij, and Pieter Adriaans. “Two Problems for Sophistication”. In: *Algorithmic Learning Theory*. 2015, pp. 379–394.

[22] Peter Bloem et al. “A Safe Approximation for Kolmogorov Complexity”. In: *Algorithmic Learning Theory*. 2014, pp. 336–350.

[23] Hans L. Bodlaender. “A Partial k-Arboretum of Graphs With Bounded Treewidth”. In: *Theoretical computer science* 209.1-2 (1998), pp. 1–45.

[24] Hans L. Bodlaender, Bart M.P. Jansen, and Stefan Kratsch. “Kernelization Lower Bounds by Cross-Composition”. In: *SIAM Journal on Discrete Mathematics* 28.1 (2014), pp. 277–305.

[25] Hans L. Bodlaender and Arie M.C.A. Koster. “Combinatorial Optimization on Graphs of Bounded Treewidth”. In: *The Computer Journal* 51.3 (2008), pp. 255–269.

[26] Hans L. Bodlaender et al. “On Problems Without Polynomial Kernels”. In: *Journal of Computer and System Sciences* 75.8 (2009), pp. 423–434.

[27] Ronald V. Book, D-Z Du, and David A. Russo. “On Polynomial and Generalized Complexity Cores”. In: *Third Annual Conference on Structure in Complexity Theory.* 1988, pp. 236–250.

[28] Dhruba Borthakur. *The Hadoop Distributed File System: Architecture and Design.* 2007. URL: `https://svn.apache.org/repos/asf/hadoop/common/tags/release-0.14.0/docs/hdfs_design.pdf`.

[29] Harry Buhrman and Pekka Orponen. “Random Strings Make Hard Instances”. In: *Journal of Computer and System Sciences* 2.53 (1996), pp. 261–266.

[30] Harry Buhrman and Leen Torenvliet. “**P**-Selective Self-Reducible Sets: A New Characterization of **P**”. In: *Journal of Computer and System Sciences* 53.2 (1996), pp. 210–217.

[31] Harry Buhrman, John Tromp, and Paul M.B. Vitányi. “Time and Space Bounds for Reversible Simulation”. In: *Journal of Physics A: Mathematical and General* 34.35 (2001), pp. 6821–6830.

[32] Abbas Cheddad et al. “Digital Image Steganography: Survey and Analysis of Current Methods”. In: *Signal processing* 90.3 (2010), pp. 727–752.

[33] Yijia Chen, Jörg Flum, and Moritz Müller. “Lower Bounds for Kernelizations and Other Preprocessing Procedures”. In: *Theory of Computing Systems* 48.4 (2011), pp. 803–839.

[34] Stephen A. Cook. “The Complexity of Theorem-Proving Procedures”. In: *Third Annual ACM Symposium on Theory of Computing.* 1971, pp. 151–158.

[35] Thomas H. Cormen et al. *Introduction to Algorithms.* 3rd ed. MIT Press, 2009.

[36] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* 2nd ed. John Wiley & Sons, 2006.

[37] Marek Cygan et al. *Parameterized algorithms.* Springer, 2015.

[38] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 2002.

[39] Holger Dell and Dieter van Melkebeek. “Satisfiability Allows No Nontrivial Sparsification Unless the Polynomial-Time Hierarchy Collapses”. In: *Journal of the ACM* 61.4 (2014), 23:1–23:27.

[40] Reinhard Diestel. *Graph Theory.* 5th ed. Springer, 2017.

[41] Rodney G. Downey and Michael R. Fellows. “Fixed-Parameter Tractability and Completeness”. In: *Congressus Numerantium* 87 (1992), pp. 161–187.

[42] Rodney G. Downey and Michael R. Fellows. “Fixed-Parameter Tractability and Completeness I: Basic Results”. In: *SIAM Journal on Computing* 24.4 (1995), pp. 873–921.

[43] Rodney G. Downey and Michael R. Fellows. “Fixed-Parameter Tractability and Completeness III: Some Structural Aspects of the **W** Hierarchy”. In: *Complexity Theory.* Ed. by Klaus Ambos-Spies, Steven Homer, and Uwe Schöning. Cambridge University Press, 1993, pp. 191–225.

[44] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer Science & Business Media, 1999.

[45] Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. “Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability”. In: *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future.* Vol. 49. 1999, pp. 49–99.

[46] Rodney G. Downey and Denis R. Hirschfeldt. *Algorithmic Randomness and Complexity.* Springer Science & Business Media, 2010.

[47] Andrew Drucker. “New Limits to Classical and Quantum Instance Compression”. In: *SIAM Journal on Computing* 44.5 (2015), pp. 1443–1479.

[48] Peter Elias. “Universal Codeword Sets and Representations of the Integers”. In: *IEEE Transactions on Information Theory* 21.2 (1975), pp. 194–203.

[49] Peter van Emde Boas. “Machine Models and Simulations”. In: *Handbook of Theoretical Computer Science. Algorithms and Complexity.* Ed. by Jan van Leeuwen. 1990. Chap. 1, pp. 3–66.

[50] Richard L. Epstein, Richard Haas, and Richard L. Kramer. “Hierarchies of Sets and Degrees Below $\emptyset'$”. In: *Logic Year 1979–80.* Ed. by Manuel Lerman, James H. Schmerl, and Robert I. Soare. Springer, 1981, pp. 32–48.

[51] Paul Erdős and Alfréd Rényi. “On Random Graphs I”. In: *Publicationes Mathematicae Debrecen* 6 (1959), pp. 290–297.

[52] Yurii L. Ershov. “A Hierarchy of Sets, I”. In: *Algebra and Logic* 7.1 (1968), pp. 25–43.

[53] Yurii L. Ershov. “On a Hierarchy of Sets, II”. In: *Algebra and Logic* 7.4 (1968), pp. 212–232.

[54] Michael R. Fellows, Bart M.P. Jansen, and Frances Rosamond. “Towards Fully Multivariate Algorithmics: Parameter Ecology and the Deconstruction of Computational Complexity”. In: *European Journal of Combinatorics* 34.3 (2013), pp. 541–566.

[55] Philippe Flajolet and Jean-Marc Steyaert. “On Sets Having Only Hard Subsets”. In: *International Colloquium on Automata, Languages, and Programming.* Springer. 1974, pp. 446–457.

[56] Jörg Flum and Martin Grohe. "Describing Parameterized Complexity Classes". In: *Information and Computation* 187.2 (2003), pp. 291–319.

[57] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer Science & Business Media, 2006.

[58] Fedor V. Fomin et al. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

[59] Lance Fortnow and Martin Kummer. "On Resource-Bounded Instance Complexity". In: *Theoretical Computer Science* 161.1-2 (1996), pp. 123–140.

[60] Lance Fortnow and Rahul Santhanam. "Infeasibility of Instance Compression and Succinct PCPs for **NP**". In: *Journal of Computer and System Sciences* 77.1 (2011), pp. 91–106.

[61] Rich Franzen and the Eastman Kodak Company. *Kodak Lossless True Color Image Suite*. 1999. URL: `http://r0k.us/graphics/kodak/`.

[62] Péter Gács, John T. Tromp, and Paul M.B. Vitányi. "Algorithmic Statistics". In: *IEEE Transactions on Information Theory* 47.6 (2001), pp. 2443–2463.

[63] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of* **NP**-*Completeness*. W.H. Freeman, 1979.

[64] John G. Geske, Dung T. Huynh, and Joel I. Seiferas. "A Note on Almost-Everywhere-Complex Sets and Separating Deterministic-Time-Complexity Classes". In: *Information and Computation* 92.1 (1991), pp. 97–104.

[65] Edgar N. Gilbert. "Random Graphs". In: *The Annals of Mathematical Statistics* 30.4 (1959), pp. 1141–1144.

[66] Mark E. Gold. "Limiting Recursion". In: *The Journal of Symbolic Logic* 30.1 (1965), pp. 28–48.

[67] Andrew V. Goldberg and Michael Sipser. "Compression and Ranking". In: *SIAM Journal on Computing* 20.3 (1991), pp. 524–536.

[68] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[69] Judy Goldsmith and Steven Homer. "Scalability and the Isomorphism Problem." In: *Information Processing Letters* 57.3 (1996), pp. 137–143.

[70] Judy Goldsmith, Deborah Joseph, and Paul Young. "A Note on Bi-immunity and p-Closeness of p-Cheatable Sets in $\mathbf{P}_{/\mathbf{poly}}$". In: *Journal of Computer and System Sciences* 46.3 (1993), pp. 349–362.

[71] Solomon Golomb. "Run-Length Encodings". In: *IEEE Transactions on Information Theory* 12.3 (1966), pp. 399–401.

[72] Peter D. Grünwald. *The Minimum Description Length Principle.* MIT Press, 2007.

[73] Jiong Guo and Rolf Niedermeier. “Invitation to Data Reduction and Problem Kernelization”. In: *ACM SIGACT News* 38.1 (2007), pp. 31–45.

[74] Juris Hartmanis. “Generalized Kolmogorov Complexity and the Structure of Feasible Computations”. In: *24th Annual Symposium on Foundations of Computer Science*. 1983, pp. 439–445.

[75] Juris Hartmanis. “Relations Between Diagonalization, Proof Systems, and Complexity Gaps”. In: *Theoretical Computer Science* 8.2 (1979), pp. 239–253.

[76] Juris Hartmanis and Lane A. Hemachandra. “One-Way Functions and the Nonisomorphism of **NP**-Complete Sets”. In: *Theoretical Computer Science* 81.1 (1991), pp. 155–163.

[77] Juris Hartmanis and Richard E. Stearns. “On the Computational Complexity of Algorithms”. In: *Transactions of the American Mathematical Society* 117 (1965), pp. 285–306.

[78] Juris Hartmanis and Yaacov Yesha. “Computation Times of **NP** Sets of Different Densities”. In: *Theoretical Computer Science* 34.1-2 (1984), pp. 17–32.

[79] Lane A. Hemachandra and Steven Rudich. “On the Complexity of Ranking”. In: *Journal of Computer and System Sciences* 41.2 (1990), pp. 251–271.

[80] Lane A. Hemachandra et al. “On Sets Polynomially Enumerable by Iteration”. In: *Theoretical Computer Science* 80.2 (1991), pp. 203–225.

[81] Lane A. Hemaspaandra et al. “Nondeterministically Selective Sets”. In: *International Journal of Foundations of Computer Science* 06.04 (1995), pp. 403–416.

[82] Fred C. Hennie and Richard E. Stearns. “Two-Tape Simulation of Multitape Turing Machines”. In: *Journal of the ACM* 13.4 (1966), pp. 533–546.

[83] Danny Hermelin et al. “A Completeness Theory for Polynomial (Turing) Kernelization”. In: *Algorithmica* 71.3 (2015), pp. 702–730.

[84] Sachin Hosmani, H.G. Rama Bhat, and K. Chandrasekaran. “Dual Stage Text Steganography Using Unicode Homoglyphs”. In: *International Symposium on Security in Computing and Communication.* 2015, pp. 265–276.

[85] Troy Hunt. *Have I Been Pwned: Pwned Passwords.* Version 4. 2019. URL: `https://haveibeenpwned.com/Passwords`.

[86] Marcus Hutter. “The Fastest and Shortest Algorithm For All Well-Defined Problems”. In: *International Journal of Foundations of Computer Science* 13.3 (2002), pp. 431–443.

[87] Bart M.P. Jansen. “Turing Kernelization for Finding Long Paths and Cycles in Restricted Graph Classes”. In: *Journal of Computer and System Sciences* 85 (2017), pp. 18–37.

[88] Carl G. Jockusch. “Semirecursive Sets and Positive Reducibility”. In: *Transactions of the American Mathematical Society* 131.2 (1968), pp. 420–436.

[89] Paris C. Kanellakis and John C. Mitchell. “Polymorphic Unification and ML Typing”. In: *16th ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages*. 1989, pp. 105–115.

[90] Stephen Cole Kleene. *Mathematical Logic*. Reprint of the John Wiley & Sons 1967 edition. Dover Publications, 2002.

[91] Stephen Cole Kleene. “Recursive Predicates and Quantifiers”. In: *Transactions of the American Mathematical Society* 53.1 (1943), pp. 41–73.

[92] Ker-I Ko. “On Self-Reducibility and Weak **P**-Selectivity”. In: *Journal of Computer and System Sciences* 26.2 (1983), pp. 209–221.

[93] Ker-I Ko and Daniel Moore. “Completeness, Approximation and Density”. In: *SIAM Journal on Computing* 10.4 (1981), pp. 787–796.

[94] Ker-I Ko et al. “What is a Hard Instance of a Computational Problem?” In: *Structure in Complexity Theory*. 1986, pp. 197–217.

[95] Christian Komusiewicz and Rolf Niedermeier. “New Races in Parameterized Algorithmics”. In: *Mathematical Foundations of Computer Science*. Vol. 12. 2012, pp. 19–30.

[96] Moshe Koppel. “Structure”. In: *The Universal Turing Machine: A Half-Century Survey*. Ed. by Rolf Herken. Oxford University Press, 1988, pp. 435–452.

[97] Stefan Kratsch. “Recent Developments in Kernelization: A Survey”. In: *Bulletin of EATCS* 2.113 (2014), pp. 57–97.

[98] Martin Kummer. “Kolmogorov Complexity and Instance Complexity of Recursively Enumerable Sets”. In: *SIAM Journal on Computing* 25.6 (1996), pp. 1123–1143.

[99] Stuart A. Kurtz, Stephen R. Mahaney, and James S. Royer. “The Isomorphism Conjecture Fails Relative to a Random Oracle”. In: *Twenty-First Annual ACM Symposium on Theory of Computing*. 1989, pp. 157–166.

[100] Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. 3rd ed. Springer, 2008.

[101] Daniel Lokshtanov. “New Methods in Parameterized Algorithms and Complexity”. PhD thesis. University of Bergen, 2009.

[102] Nancy Lynch. “On Reducibility to Complex or Sparse Sets”. In: *Journal of the ACM* 22.3 (1975), pp. 341–345.

[103] Miroslaw Malek and Ahmed A. Moin. *A Hybrid Technique for Deterministic Algorithms with a Shortest Path Example.* Tech. rep. Department of Computer Science Technical Report TR94-15, The University of Texas at Austin, 1994.

[104] Elvira Mayordomo. “Almost Every Set in Exponential Time is **P**-Bi-Immune”. In: *Theoretical Computer Science* 136.2 (1994), pp. 487–506.

[105] M. Douglas McIlroy. “A Killer Adversary for Quicksort”. In: *Software: Practice and Experience* 29.4 (1999), pp. 341–344.

[106] Albert R. Meyer and Mike S. Paterson. “With What Frequency are Apparently Intractable Problems Difficult?” In: *MIT LCS Technical Report* 126 (1979).

[107] Alexey Milovanov. “Algorithmic Statistics, Prediction and Machine Learning”. In: *33rd Symposium on Theoretical Aspects of Computer Science.* Vol. 47. 2016, 54:1–54:13.

[108] Alexey Milovanov. “On Algorithmic Statistics for Space-Bounded Algorithms”. In: *International Computer Science Symposium in Russia.* 2017, pp. 232–244.

[109] Andrei A. Muchnik, Alexei L. Semenov, and Vladimir A. Uspensky. “Mathematical Metaphysics of Randomness”. In: *Theoretical Computer Science* 207.2 (1998), pp. 263–317.

[110] Carl Mummert. “Filter Quantifiers”. 2014. URL: `http://science.marshall.edu/mummertc/papers/quantifiers.pdf`.

[111] David R. Musser. “Introspective Sorting and Selection Algorithms”. In: *Software: Practice and Experience* 27.8 (1997), pp. 983–993.

[112] Mark E.J. Newman. “Power Laws, pareto Distributions and Zipf’s Law”. In: *Contemporary Physics* 46.5 (2005), pp. 323–351.

[113] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[114] Rolf Niedermeier. “Reflections on Multivariate Algorithmics and Problem Parameterization”. In: *27th Symposium on Theoretical Aspects of Computer Science.* 2010, pp. 17–32.

[115] Piergiorgio Odifreddi. *Classical Recursion Theory.* Elsevier, 1992.

[116] Pekka Orponen. “A Classification of Complexity Core Lattices”. In: *Theoretical Computer Science* 47 (1986), pp. 121–130.

[117] Pekka Orponen, David A. Russo, and Uwe Schöning. “Optimal Approximations and Polynomially Levelable Sets”. In: *SIAM Journal on Computing* 15.2 (1986), pp. 399–408.

[118] Pekka Orponen, David A. Russo, and Uwe Schöning. "Polynomial Levelability and Maximal Complexity Cores". In: *International Colloquium on Automata, Languages, and Programming*. 1985, pp. 435–444.

[119] Pekka Orponen et al. "Instance Complexity". In: *Journal of the ACM* 41.1 (1994), pp. 96–121.

[120] Ian Parberry. "Parallel Speedup of Sequential Machines: A Defense of the Parallel Computation Thesis". In: *ACM SIGACT News* 18.1 (1986), pp. 54–67.

[121] Tim Peters. *timsort documentation*. First publication: 2002. 2013. URL: `https://github.com/python/cpython/blob/v2.7.6/Objects/listsort.txt`.

[122] Emil L. Post. "Degrees of Recursive Unsolvability". In: *Bulletin of the American Mathematical Society*. Vol. 54. 7. 1948, pp. 641–642.

[123] Emil L. Post. "Recursively Enumerable Sets of Positive Integers and Their Decision Problems". In: *Bulletin of the American Mathematical Society* 50.5 (1944), pp. 284–316.

[124] Hilary Putnam. "Trial and Error Predicates and the Solution to a Problem of Mostowski". In: *The Journal of Symbolic Logic* 30.1 (1965), pp. 49–57.

[125] Henry G. Rice. "Classes of Recursively Enumerable Sets and Their Decision Problems". In: *Transactions of the American Mathematical Society* 74.2 (1953), pp. 358–366.

[126] Erik Riedel, Garth Gibson, and Christos Faloutsos. "Active Storage for Large-Scale Data Mining and Multimedia Applications". In: *24th Conference on Very Large Databases*. 1998, pp. 62–73.

[127] Jorma Rissanen. "A Universal Prior for Integers and Estimation by Minimum Description Length". In: *The Annals of Statistics* 11 (1983), pp. 416–431.

[128] Jorma Rissanen. "Modeling by Shortest Data Description". In: *Automatica* 14.5 (1978), pp. 465–471.

[129] Neil Robertson and Paul D. Seymour. "Graph minors II: Algorithmic Aspects of Tree-Width". In: *Journal of algorithms* 7.3 (1986), pp. 309–322.

[130] Neil Robertson and Paul D. Seymour. "Graph minors XIII: The Disjoint Paths Problem". In: *Journal of Combinatorial Theory, Series B* 63.1 (1995), pp. 65–110.

[131] Hartley Rogers. *Theory of Recursive Functions and Effective Computability*. Vol. 5. McGraw–Hill, 1967.

[132] Roy S. Rubinstein. *A Note on Sets With Small Generalized Kolmogorov Complexity*. Tech. rep. Technical Report TR86-4, Iowa State University, 1986.

[133] Khalid Sayood. *Introduction to Data Compression*. 5th ed. Morgan Kaufmann, 2017.

[134] Uwe Schöning. “Probabilistic Complexity Classes and Lowness”. In: *Journal of Computer and System Sciences* 39.1 (1989), pp. 84–100.

[135] Alan L. Selman. “Analogues of Semirecursive Sets and Effective Reducibilities to the Study of **NP** Complexity”. In: *Information and Control* 52.1 (1982), pp. 36–51.

[136] Alan L. Selman. “**P**-Selective Sets, Tally Languages, and the Behavior of Polynomial Time Reducibilities on **NP**”. In: *Mathematical Systems Theory* 13.1 (1979), pp. 55–65.

[137] Alexander Shen. “The Concept of (α, β)-Stochasticity in the Kolmogorov Sense, and its Properties”. In: *Soviet Mathematics – Doklady* 28.1 (1983), pp. 295–299.

[138] Joseph R. Shoenfield. “On Degrees of Unsolvability”. In: *Annals of Mathematics* (1959), pp. 644–653.

[139] Jon Sneyers and Pieter Wuille. “FLIF: Free Lossless Image Format Based on MANIAC Compression”. In: *IEEE International Conference on Image Processing*. 2016, pp. 66–70.

[140] Robert I. Soare. *Turing Computability: Theory and Applications*. Springer, 2016.

[141] Stéphan Thomassé, Nicolas Trotignon, and Kristina Vušković. “A Polynomial Turing-Kernel for Weighted Independent Set in Bull-Free Graphs”. In: *Algorithmica* 77.3 (2017), pp. 619–641.

[142] Boris A. Trakhtenbrot. “On Autoreducibility”. In: *Soviet Mathematics* 11 (1970), pp. 814–817.

[143] Anne Sjerp Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. 2nd ed. 43. Cambridge University Press, 2000.

[144] Alan M. Turing. “On Computable Numbers, With an Application to the Entscheidungsproblem”. In: *Proceedings of the London Mathematical Society* 2.1 (1937), pp. 230–265.

[145] Nikolai K. Vereshchagin and Paul M.B. Vitányi. “Rate Distortion and Denoising of Individual Data Using Kolmogorov Complexity”. In: *IEEE Transactions on Information Theory* 56.7 (2010), pp. 3438–3454.

[146] Nikolay K. Vereshchagin. “Algorithmic Minimal Sufficient Statistic Revisited”. In: *Computability in Europe*. 2009, pp. 478–487.

[147] Nikolay K. Vereshchagin and Alexander Shen. “Algorithmic Statistics: Forty Years Later”. In: *Computability and Complexity*. Springer, 2017, pp. 669–737.

[148] Nikolay K. Vereshchagin and Paul M.B. Vitányi. “Kolmogorov’s Structure Functions and Model Selection”. In: *IEEE Transactions on Information Theory* 50.12 (2004), pp. 3265–3290.

[149] Paul M.B. Vitányi. “Meaningful Information”. In: *IEEE Transactions on Information Theory* 52.10 (2006), pp. 4617–4626.

[150] Paul M.B. Vitányi and Ming Li. “Minimum Description Length Induction, Bayesianism, and Kolmogorov Complexity”. In: *IEEE Transactions on Information Theory* 46.2 (2000), pp. 446–464.

[151] Mathias Weller. “Aspects of Preprocessing Applied to Combinatorial Graph Problems”. PhD thesis. Technische Universität Berlin, 2013.

[152] Jouke Witteveen. “Structural Parameterized Complexity”. Master’s thesis. ILLC, Universiteit van Amsterdam, 2015.

[153] Jouke Witteveen, Ralph Bottesch, and Leen Torenvliet. “A Hierarchy of Polynomial Kernels”. In: *International Conference on Current Trends in Theory and Practice of Informatics*. 2019, pp. 504–518.

[154] Jouke Witteveen and Leen Torenvliet. “Fixed-Parameter Decidability: Extending Parameterized Complexity Analysis”. In: *Mathematical Logic Quarterly* 62.6 (2016), pp. 596–607.

[155] Chee K. Yap. “Some Consequences of Non-Uniform Conditions on Uniform Classes”. In: *Theoretical Computer Science* 26.3 (1983), pp. 287–300.

[156] Paul Young. “Some Structural Properties of Polynomial Reducibilities and Sets in **NP**”. In: *Fifteenth Annual ACM Symposium on Theory of Computing*. 1983, pp. 392–401.

[157] Tom C. van der Zanden and Hans L. Bodlaender. “Computing Treewidth on the GPU”. In: *12th International Symposium on Parameterized and Exact Computation*. Vol. 89. 2018, 29:1–29:13.

Index

Gearfetting

Oerset troch Janneke Spoelstra

Kompleksiteit kin in protte foarmen hawwe, dochs is der net ien wiskundige definysje fan kompleksiteit dêr't se allegearre oan foldogge. Yn dit proefskrift yntrodusearje wy in wiskundich ramt foar de analyze fan ferskillende foarmen fan kompleksiteit.

Us ramt is in fuortsetting fan de parametrisearre oanpak fan komputasjonele kompleksiteit sa't Downey en Fellows dat earder dien hawwe. Sa giet it yn dit proefskrift ek net allinnich om de analyze fan kompleksiteit, mar ek om de teory fan parametrisearre komputasjonele kompleksiteit. Der binne twa typen resultaten yn dit proefskrift: resultaten oangeande it tapassen fan in unifoarme analyze fan kompleksiteit mei gebrûk fan ús ramt, en resultaten oangeande de klasse fan goed te behappen samlingen foar in fêste parameter, **FPT**.

Twa konkrete domeinen dêr't wy de kompleksiteit yn analysearje mei gebrûk fan ús ramt binne statistyske ynferinsje en algoritmeûntwerp. Foar statistyske ynferinsje jout ús ramt rjochtlinen om it lykwicht te bewarjen tusken underfitting en overfitting fan statistyske modellen. Us ramt biedt yndikatoaren foar algoritme-ûntwerp, dy't brûkt wurde kinne by it besluten fan hoefolle berekkening oft der wannear en wêr dien wurde moat, foar minimaal totaal gebrûk fan helpboarnen.

Oangeande de aard fan **FPT** hawwe wy trije resultaten. Mei gebrûk fan in rangoarder fan parametrisaasjes ôflaat fan **FPT** litte wy sjen dat der faak gjin bêste parametrisaasje is ûnder dy dy't in samling yn **FPT** pleatse. Twad litte wy sjen dat der in strikte hiërargy is ûnder **FPT** dy't basearre is op polynomiale kernelisaasjes. As lêste fine wy bewiis foar in alternative karakterisearring fan **FPT** as de kosjintgroep $\Delta_1^0/\mathbf{P}$ gebrûk meitsjend fan it symmetrysk ferskil. Hjir is Δ_1^0 de klasse fan beslútbere samlingen, it earste trochsneednivo fan de rekkenkundige hiërargy.

Samenvatting

Complexiteit kent vele vormen. Iets kan bijvoorbeeld complex zijn omdat het niet eenvoudig te beschrijven is. In dat geval spreken we van *algoritmische complexiteit*, soms ook wel descriptieve complexiteit genoemd. Een andere vorm van complexiteit vinden we bij computationele vraagstukken waarvoor beantwoording veel rekentijd of geheugen vereist. Deze vorm van complexiteit is bekend als *computationele complexiteit*. Wanneer voor een computationeel vraagstuk geen antwoord te berekenen valt, ongeacht de beschikbare hoeveelheid rekentijd en geheugen, zeggen we dat het vraagstuk *onberekenbaar* is. Onberekenbaarheid kan gezien worden als een extreme vorm van computationele complexiteit. Hoewel algoritmische complexiteit, computationele complexiteit, en onberekenbaarheid allen vormen van complexiteit zijn, is er geen overkoepelende wiskundige definitie waaraan ze allen voldoen.

In dit proefschrift introduceren we een wiskundig kader waarbinnen complexiteit geanalyseerd kan worden. Dit kader is dermate algemeen dat de analyse van elk van de drie eerdergenoemde vormen van complexiteit erin mogelijk is. Hierdoor is het ook mogelijk om verschillende vormen van complexiteit met elkaar te vergelijken. Zodoende vinden we onder andere dat computationele complexiteit geïmpliceerd wordt door algoritmische complexiteit.

Ons kader behelst een voortzetting van de geparametriseerde benadering van computationele complexiteitstheorie door Downey en Fellows. Traditioneel wordt in complexiteitstheorie gekeken naar hoe de benodigde hoeveelheid rekentijd van een collectie verwante vraagstukken afhangt van de lengte van de specificaties van de vraagstukken. Deze collectie wordt een computationeel *probleem* genoemd, en de afzonderlijke vraagstukken *instanties* van het probleem. In geparametriseerde complexiteitstheorie worden naast de lengte van de specificatie van een instantie ook andere aspecten ervan in ogenschouw genomen. Hierdoor wordt de complexiteitsanalyse ingewikkelder, maar kan zij ook beter aansluiten bij hoe we complexiteit in de praktijk ervaren. Neem bijvoorbeeld het bepalen van de kortste route tussen twee punten op een kaart. De instanties van dit computationele probleem zijn de vraagstukken die ontstaan waarbij een specifieke kaart en twee specifieke punten gegeven zijn. Wanneer de kaart een erg groot gebied bestrijkt, kan de instantie tamelijk moeilijk zijn. Echter, zelfs wanneer de kaart erg groot is, is het bepalen van de kortste route tussen nabijgelegen punten doorgaans makkelijk. Voor het bepalen van kortste routes hebben we aldus twee factoren geïdentificeerd die van invloed zijn op de complexiteit. Enerzijds kunnen we stellen dat instanties van het probleem moeilijker worden naarmate de kaart groter wordt. Anderzijds blijft het probleem makkelijk zolang de punten waartussen de kortste route moet worden gevonden dichtbij elkaar liggen. In geparametriseerde complexiteitstheorie worden aspecten van een instantie, waaronder de grootte, de *parameters* van het probleem genoemd. Een functie die gegeven een probleeminstantie de bijbehorende parameterwaarde bepaald heet een *parametrisatie*.

Ook bij computationele problemen die aanzienlijk ingewikkelder zijn dan het vinden van kortste routes kunnen we dikwijls verschillende factoren aanwijzen die van invloed zijn op de complexiteit. Deze observatie ligt aan de basis van de klasse der *fixed-parameter tractable* problemen: computationele problemen waarvan de complexiteit te overzien is zolang we ons beperken tot instanties met een vaste parameterwaarde. Meer exact zijn de fixed-parameter tractable problemen die problemen waarbij, voor instanties met een vaste parameterwaarde, de toename in rekentijd als functie van de lengte van de instantie begrensd kan worden door een polynoom. De graad van dit polynoom mag niet afhangen van de parameterwaarde, maar de coëfficiënten van het polynoom mogen dat wel.

Het wiskundig kader voor de analyse van complexiteit dat in dit proefschrift geïntroduceerd wordt is opgebouwd rondom parametrisaties. Derhalve zijn er twee soorten resultaten in dit proefschrift: resultaten ten aanzien van een overkoepelende analyse van complexiteit, en resultaten ten aanzien van fixed-parameter tractability.

We analyseren complexiteit in verschillende domeinen, waaronder dat van inferentie en model selectie in de statistiek. In dit domein gaat het erom uit een verzameling modellen een model aan te wijzen dat het meest waarschijnlijk is gegeven enige waargenomen data. Enerzijds mag dit model niet te specifiek zijn, aangezien het dan meer structuur in de data suggereert dan geoorloofd is. Anderzijds verschaft een te algemeen model weinig inzicht in de structuur die daadwerkelijk in de data aanwezig is. De specificiteit van een model is een vorm van algoritmische complexiteit en ons wiskundige kader biedt richtlijnen voor de keuze van de juiste mate van specificiteit gegeven de waargenomen data.

Een vorm van complexiteit die dichter bij computationele complexiteit ligt treffen we aan in de algoritmiek. Een goed algoritme is in staat een computationeel probleem, zoals het vinden van kortste routes, efficiënt op te lossen. Hiertoe moet, gegeven een probleeminstantie, worden bepaald hoeveel rekenwerk er op welk moment en op welk systeem gedaan moet worden teneinde de totale rekenkosten te minimaliseren. Wanneer een instantie eenvoudig te herkennen redundantie bevat, doen we er goed aan de instantie zo vroeg mogelijk van deze redundantie te ontdoen. Zonder de redundante delen neemt de specificatie van de instantie minder geheugen in beslag en kan de instantie efficiënter gecommuniceerd worden naar andere rekeneenheden. Wanneer redundantie te verwachten is, is een gefaseerd algoritme dat de instantie eerst van eventuele redundantie ontdoet dus aan te bevelen. Voor instanties zonder eenvoudig te herkennen redundantie biedt een gefaseerde aanpak geen voordelen. Echter, wat er wel en niet als redundantie gezien mag worden hangt af van de details van het systeem waarop het algoritme draait. In meer abstracte zin kunnen we stellen dat het computationeel model van invloed is op de notie van complexiteit in het kader van het ontwerp van algoritmen.

De analyse van complexiteit in verband met gefaseerde algoritmen verschaft ons tevens inzicht in fixed-parameter tractability. Een standaardresultaat uit de geparametriseerde complexiteitstheorie luidt dat fixed-parameter tractability overeenkomt met de mogelijkheid een probleeminstantie op een bepaalde manier voor te bewerken. Deze voorbewerking komt erop neer dat de grootte van een instantie in polynomiale tijd zoveel kan worden teruggebracht, dat zij kan worden begrensd door een functie van de parameterwaarde behorende bij de instantie. In het licht van onze complexiteitsanalyse correspondeert dit met een erg algemene vorm van redundantie. We definiëren in dit proefschrift een hiërarchie van meer specifieke vormen van redundantie en leggen zodoende een hiërarchie van klassen van computationele problemen bloot binnen de fixed-parameter tractable problemen.

Of een probleem fixed-parameter tractable is hangt af van de parametrisatie die we in gedachten hebben. Echter, ieder probleem dat met *enige* parametrisatie fixed-parameter tractable is, is berekenbaar. We tonen aan dat, omgekeerd, ieder berekenbaar probleem fixed-parameter tractable gemaakt kan worden. Dit resultaat laat zich generaliseren naar een meer non-uniforme situatie, waarbij beslisbaarheid vervangen wordt het Δ^0_2-niveau van de arithmetische hiëarchie.

We werpen ook een meer parametrisatie-gerichte blik op fixed-parameter tractability, en bekijken de collectie parametrisaties waarmee een gegeven probleem fixed-parameter tractable is. In het bijzonder kijken we naar het verband tussen verschillende problemen waarvoor deze collectie hetzelfde blijft. We vinden aanwijzingen dat het verband tussen zulke problemen ook gekarakteriseerd kan worden op een manier die los staat van geparametriseerde overwegingen. Een vergelijkbare karakterisering gaat niet op in de non-uniforme situatie.

Onze intuïtie zegt ons dat naarmate een probleem met meer parametrisaties fixed-parameter tractable is, het minder moeilijk is. Aan de hand hiervan construeren we een ordening van parametrisaties. Makkelijkere problemen zijn fixed-parameter tractable met parametrisaties die eerder komen in deze ordening. Het kan voorkomen dat er onder de parametrisaties die een probleem fixed-parameter tractable maken een parametrisatie is die in deze ordening voor alle anderen geordend wordt. Echter, we tonen aan dat dit niet het geval is voor de meest interessante problemen: dergelijke optimale parametrisaties bestaan niet voor problemen die niet **P**-bi-immuun zijn.

Abstract

Complexity can have many forms. An object, for instance, can be complex because a lot of words are needed to describe it. Such an object is said to have high *algorithmic complexity.* This form of complexity is also referred to as “descriptive complexity”, but that term is used for entirely different concepts as well. Another example is a computational problem that is complex because it can only be solved via computations that use a lot of time or memory. Such a problem is said to have high *computational complexity.* Of comparable, but more extreme flavor would be a computational problem that is complex because no amount of time or memory is enough for solving it. Such a problem is said to be *uncomputable.* While all three of these examples are forms of complexity, there is no single mathematical definition of complexity that they all adhere to.

In this thesis, we introduce a mathematical framework for the analysis of complexity. The framework is versatile enough to deal with the three aforementioned forms of complexity and more. Because of this versatility, we are able to compare different forms of complexity to each other. Doing so, we find, for instance, that algorithmic complexity implies computational complexity.

Our framework is a continuation of the parameterized approach to computational complexity pioneered by Downey and Fellows. In the parameterized approach, computational complexity is not measured simply as a function of the length of the specification of a computational problem. Instead, other information about the computational problem is taken into account as well. This makes the analysis of complexity harder, but it can also paint a more accurate picture of computational complexity as encountered in practice. For example, the problem of finding the shortest route between two points on a map may be relatively difficult if the map is very large. However, even on a large map, the problem is easy if the two points are in close proximity to each other. Thus, we have two factors that influence the computational complexity of the general problem of finding shortest routes. On the one hand, instances of the problem potentially become more difficult as the size of the map increases. On the other, the problem remains easy as long as the points are close together. Aspects of a problem instance, including, but not limited to, the length of its specification, are called *parameters*. A mapping of problem instance to a parameter value is called a *parameterization*.

The phenomenon of multiple factors influencing the complexity can be observed in problems far more difficult than finding shortest routes. This leads to the central notion of the parameterized approach to computational complexity, that of *fixed-parameter tractability*. As is common in computational complexity theory, tractability is equated to computation in polynomial time. Thus, a problem is fixed-parameter tractable if it can be solved in polynomial time on those instances that have some fixed parameter value. In other words, the computational complexity beyond polynomial-time computability is restricted by the parameter.

Parameterizations also form the backbone of our framework for the analysis of various notions of complexity. As a result, this thesis is not only concerned with the analysis of complexity, but also with the theory of parameterized computational complexity. Specifically, we investigate the class of fixed-parameter tractable problems. Consequently, there are two types of results in this thesis: results on the application of a unified analysis of complexity using our framework, and results on fixed-parameter tractability.

One domain to which we apply our analysis of complexity is statistical inference. Specifically, we look at model selection. In model selection, our task is to select, from a set of candidate models, a statistical model that best describes some given data. When selecting a model, we must balance *underfitting* and *overfitting*. Whether or not a model is a good fit depends on the complexity of the model as well as on the complexity of the given data. A good model captures all the structure that is present in the data, but is not suggestive of any structure that cannot reasonably be said to be present. Because of this role of the complexity of models, we are able to use our framework as a guide in model selection. Our framework enables us to quantify not only the complexity of statistical models, but also to what degree they are underfitting or overfitting the given data.

The complexity of a statistical model can be taken as a variant of algorithmic complexity. A version of complexity more related to computational complexity is found in the domain of algorithm design. Specifically, we also apply our analysis of complexity to preprocessing. For preprocessing, we must determine how much computation to perform at what moment, in order to minimize the total usage of computational resources. In many cases, we even get to determine on what part of a system computation is to be performed. In short, we must decide how much computation to perform when and where. If there is easily recognizable redundancy in the specification of an instance of a computational problem, then we might as well remove it immediately. Without the redundant parts, the instance consumes less memory and can more efficiently be communicated to other parts of the system. Thus, with instances of a computational problem that have redundancy in their specification, computation can be pulled apart. Conversely, we find that with instances that have little or no redundancy in their specification, we do best to perform all computation at once. The computational complexity of such problem instances is high and preprocessing offers no benefits for these instances. However, we further show that what counts as computational redundancy depends on the computational system we are working with. More abstractly, we find that our model of computation and preprocessing is relevant to our notion of computational redundancy. In other words, the details of the model influence the notion of complexity.

In parameterized complexity theory, preprocessing is modeled by *kernelization* with respect to a parameterization. In its most basic form, a kernelization reduces an instance of a problem to an equivalent instance of which the size is restricted by the parameter. Different types of preprocessing correspond to different types of kernelization. With kernelizations, the parameter functions as a bound on the amount of information in the specification of a problem that is not redundant. The redundancy must be easily recognizable, as kernelizations are required to run in polynomial time. We put the different types of kernelization in a hierarchy and show that the power of kernelization increases strictly throughout the hierarchy. Conversely, with respect to a given parameterization, the lower we go in the hierarchy, the fewer problems have a kernelization with the corresponding power. It is a standard result in parameterized complexity theory that if a computational problem has any kernelization at all, then it is fixed-parameter tractable. Thus, our hierarchy of kernelizations is also a proper hierarchy of computational problems inside the class of fixed-parameter tractable problems.

Whether or not a computational problem is fixed-parameter tractable depends on the parameterization that is used. However, every problem that is fixed-parameter tractable with respect to *some* parameterization is decidable. In this thesis, we find that, moreover, every decidable problem can be made fixed-parameter tractable. Additionally, in a more nonuniform theory, the same is true when we replace decidability with the Δ_2^0 level of the arithmetical hierarchy.

The previous characterization of the fixed-parameter tractable problems is rather indifferent to parameterizations. Therefore, we consider the collection of parameterizations with respect to which a given set is fixed-parameter tractable. We conjecture that two sets for which this collection is the same have a symmetric difference that lies in $\mathbf{P}$. For a more nonuniform parameterized complexity theory this is not true, but we are able to provide some evidence for the conjecture in the uniform setting.

Intuitively, the more parameterizations make a problem fixed-parameter tractable, the easier the problem is. We take this intuition further and use it to define an order on parameterizations. Easier problems are fixed-parameter tractable with respect to parameterizations that are lower in this order. Potentially, there is a parameterization among those that make a given problem fixed-parameter tractable that is below all others. However, we show that most interesting sets, specifically those that are not $\mathbf{P}$-bi-immune, do not have such an optimal parameterization.