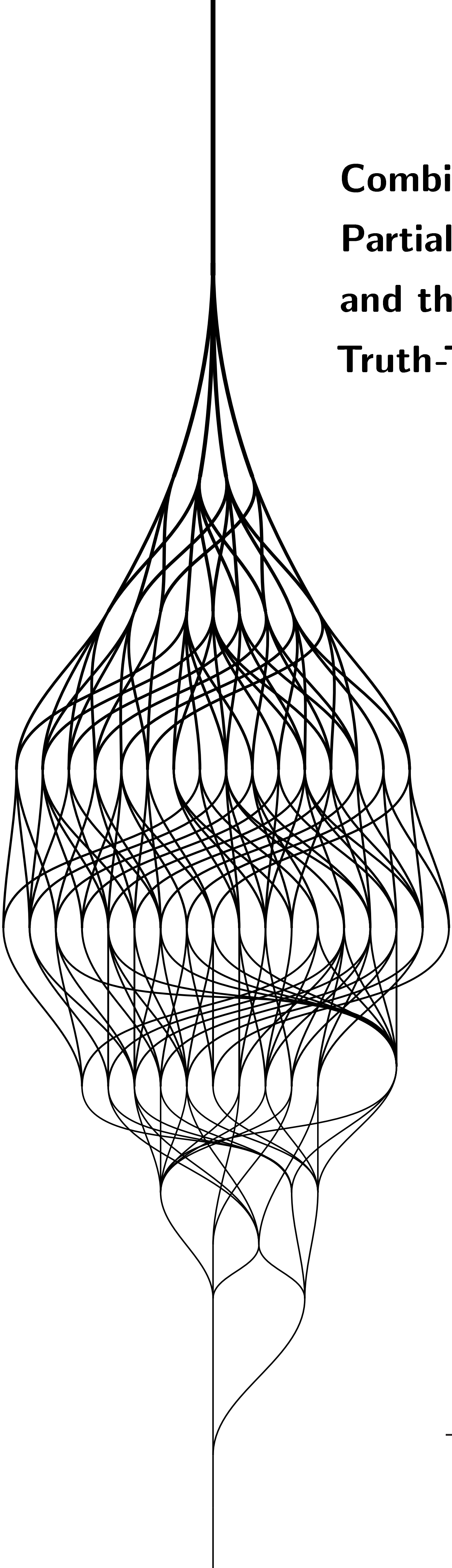
Combinatorial Representations of Partial Information Classes and their Truth-Table Closures



TILL TANTAU

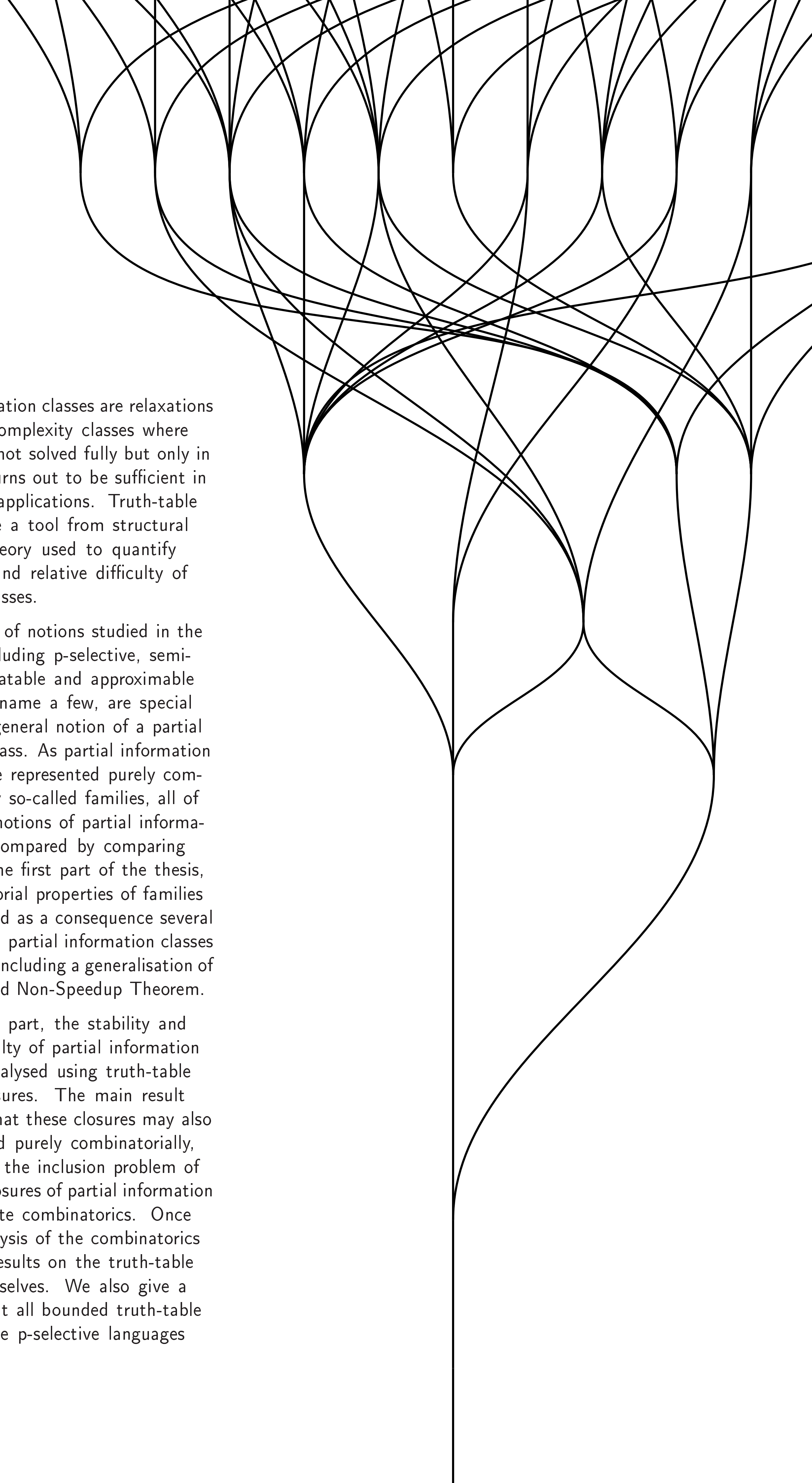
The figure on the cover page is adopted from Figure 3-8 on page 44 and depicts the partially ordered set of weak 3-units. Every antichain in this partially ordered set stands for one weakly normal 3-family.

The figure contains 5 098 786 antichains.

Partial information classes are relaxations of standard complexity classes where problems are not solved fully but only in part, which turns out to be sufficient in a number of applications. Truth-table reductions are a tool from structural complexity theory used to quantify the stability and relative difficulty of complexity classes.

A rich variety of notions studied in the literature, including p-selective, semi-recursive, cheatable and approximable languages to name a few, are special cases of the general notion of a partial information class. As partial information classes can be represented purely combinatorially by so-called families, all of the different notions of partial information can be compared by comparing families. In the first part of the thesis, the combinatorial properties of families are studied and as a consequence several new results on partial information classes are obtained, including a generalisation of the Generalised Non-Speedup Theorem.

In the second part, the stability and relative difficulty of partial information classes are analysed using truth-table reduction closures. The main result obtained is, that these closures may also be represented purely combinatorially, thus reducing the inclusion problem of truth-table closures of partial information classes to finite combinatorics. Once more, an analysis of the combinatorics yields novel results on the truth-table closures themselves. We also give a new proof that all bounded truth-table closures of the p-selective languages differ.



Diplomarbeit

Combinatorial Representations of Partial Information Classes and their Truth-Table Closures

vorgelegt von

Till Tantau

am Fachbereich Informatik
der Technischen Universität Berlin

betreut durch
Prof. Dr. Dirk Siefkes
Dr. Arfst Nickelsen

August 1999

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt zu haben.

Berlin, August 1999

Danksagungen

Zu aller erst möchte ich Dirk Siefkes und Arfst Nickelsen danken, die diese Arbeit betreut haben. Arfst hat mir sehr geholfen mit vielen nützlichen Hinweisen und Ratschlägen. Ein besonderer Dank auch an das Team der Gruppe TAL. Ich möchte auch meinen lieben Eltern danken, denn ohne ihre Unterstützung wäre diese Arbeit so nicht möglich gewesen. Und dann danke ich noch Tux, der ohne auch nur einmal zu murren die ganze Zeit geduldig neben mir gesessen hat.

Abstract

Partial information classes are relaxations of standard complexity classes where problems are not solved fully but only in part, which turns out to be sufficient in a number of applications. Truth-table reductions are a tool from structural complexity theory used to quantify the stability and relative difficulty of complexity classes.

A rich variety of notions studied in the literature, including p-selective, semirecursive, cheatable and approximable languages to name a few, are special cases of the general notion of a partial information class. As partial information classes can be represented purely combinatorially by so-called families, all of the different notions of partial information can be compared by comparing families. In the first part of the thesis, the combinatorial properties of families are studied and as a consequence several new results on partial information classes are obtained, including a generalisation of the Generalised Non-Speedup Theorem.

In the second part, the stability and relative difficulty of partial information classes are analysed using truth-table reduction closures. The main result obtained is, that these closures may also be represented purely combinatorially, thus reducing the inclusion problem of truth-table closures of partial information classes to finite combinatorics. Once more, an analysis of the combinatorics yields novel results on the truth-table closures themselves. We also give a new proof that all bounded truth-table closures of the p-selective languages differ.

Zusammenfassung

Teilinformationsklassen sind eine Verallgemeinerung klassischer Komplexitätsklassen, bei denen Probleme nicht vollständig, sondern nur zum Teil gelöst werden. Es stellt sich heraus, daß dies in einer Reihe von Anwendungen ausreichend ist. Truth-Table-Reduktionen sind ein strukturanalytisches Hilfsmittel, das häufig verwendet wird, um die Stabilität und Kompliziertheit von Komplexitätsklassen zu quantifizieren.

Viele in der Literatur untersuchte Spielarten der Teilinformation, so zum Beispiel semirekursive, p-selektive, cheatable oder auch approximierbare Sprachen, erweisen sich als Spezialfälle der allgemeinen Definition von Teilinformation. Da Teilinformationsklassen kombinatorisch repräsentiert werden können durch sogenannte Familien, können all diese Arten von Teilinformation verglichen werden, indem man die zugehörigen Familien vergleicht. Untersuchungen der Kombinatorik von Familien im ersten Teil der Arbeit ermöglichen es, neue Resultate bezüglich der zugehörigen Teilinformationsklassen zu erhalten. Ein solches Resultat ist eine Verallgemeinerung des Generalised Non-Speedup Theorems.

Im zweiten Teil werden die Stabilität und Kompliziertheit von Teilinformationsklassen mit Hilfe von Truth-Table-Reduktionsabschlüssen untersucht. Das Hauptresultat lautet, daß solche Abschlüsse ebenfalls rein kombinatorisch beschreibbar sind. Hierdurch wird das Inklusionsproblem für Truth-Table-Abschlüsse auf endliche Kombinatorik zurückgeführt. Auch im zweiten Teil ergibt eine Analyse dieser Kombinatorik neue Resultate bezüglich der Truth-Table-Abschlüsse selbst. Unter anderem ergibt sich ein neuartiger Beiweis dafür, daß alle beschränkten Reduktionsabschlüsse der p-selektiven Sprachen unterschiedlich sind.

Contents

Preface

Motivation and Aims vi
Overview and Main Results viii

I Combinatorial Representations of Partial Information Classes

Introduction to Partial Information **2**

1.1 Definition of Pools and Families 3
1.2 Definition of Cartesian and Compositionally Closed Function Classes . 6
1.3 Definition of Partial Information Classes 8
1.4 Computing Partial Information for Satisfiability 11
1.5 Computing Partial Information for Reachability and Circuit Value . . 15

Representation of Resource Bounded Partial Information **19**

2.1 Definition of Subset Closed Families 20
2.2 Definition of Normal Families 21
2.3 Definition of Recursively Presentable Function Classes 23
2.4 Proof of the Unique Normal Form Theorem 26

Structure of Normal Families **29**

3.1 Definition of Generating Systems 30
3.2 Definition of Bases 31
3.3 Representing Normal Families by Antichains 32
3.4 Upward Translation of Normal Families 34
3.5 Upward Translation of Special Families 37
3.6 Well-Foundedness of Inclusion on Cheatable Partial Information . . . 40
3.7 Figures and Tables 42

Structure of Stable Families **46**

4.1 Definition of Hard Tuples 47
4.2 Definition of Hard Remainder Pools 48
4.3 Computing Partial Information using Hard Tuples 50
4.4 Computing Partial Information for Branches 54

II Truth-Table Closures of Partial Information Classes

Introduction to Reductions **59**

5.1 Review of Many-One, Turing and Positive Reductions 60
5.2 Definition of Truth-Table Reductions 62
5.3 Representing Bounded Truth-Table Reductions by Evaluation Types . 64
5.4 Representing Bounded Turing Reductions by Evaluation Types 66

Representation of Truth-Table Closures **70**

6.1 Definition of Cones 71
6.2 Definition of Canonically Complete Languages 73
6.3 Proof of the Cone Theorem 73
6.4 Checking Basic Closure Properties using Cones 75

Structure of Selective, Bottom and Top Cones **78**

7.1 Definition of Walks and Change Numbers 79
7.2 Characterising the Cone Property for Selective Families 80
7.3 Separating the Closures of Selective Classes 82
7.4 Shrinking the Evaluation Type of Reductions to Selective Languages . 83
7.5 Separating the Closures of Bottom and Top Classes 84
7.6 Separating the Positive Closures of Bottom and Top Classes 86

Structure of Cones over Families **88**

8.1 Characterising the Cone Property for Upward Translated Families . . 88
8.2 Closure Property of Approximable Classes 89
8.3 Closure Property of Approximable Polynomial Time is Optimal 90

Conclusion

Representing Classes by Families 93
Representing Closures by Cones 94
Outlook 95

References

Bibliography **96**

List of Notations **102**

Index **105**

Preface

So eine Arbeit wird eigentlich nie fertig, man muß sie für fertig erklären,
wenn man nach Zeit und Umständen das mögliche getan hat.

— Johann Wolfgang von Goethe, *Italienische Reise*, 1797

Gentle reader, Goethe's words point out a problem you will undoubtedly have spotted immediately upon taking this thesis into your hands—it is much longer than I had originally intended. The world of partial information to which Arfst Nickelsen introduced me, whom I would like to thank once more for doing so, is a rich and diverse world. I sincerely hope that this text grew longer than originally anticipated, not so much because I failed to express myself succinctly, but rather because I tried to give the two main themes of this thesis—combinatorial representations of partial information classes and of their truth-table closures—a treatment as thorough as each of them merits. I hope to reward your reading of this thesis, by giving you some new insights into an exciting subject.

Motivation and Aims

In computer science, many problems can be solved using divide and conquer algorithms. A given problem instance is broken up into smaller problems, which can be decided independently and whose solutions effectively solve the given problem. One motivation for the study of *partial information* is that we often do not need to solve *all* subproblems or that a small fraction of the solutions may be allowed to be incorrect.

Partial information was first used as a theoretical tool in recursion theory. The earliest definition of a notion of partial information is the definition of frequency computations due to Rose (1960). If we represent subproblems by words for which membership with respect to some language needs to be decided, Rose considered the situation where for n given distinct words we are required to decide membership for these words, but are allowed to *make up to r mistakes*.

If the number of incorrect answers is relatively small, the answers may suffice to solve the problem at hand. As a matter of fact, Trakhtenbrot (1963) showed that for $r < n/2$ *deciding a language is no harder than computing partial information* in the recursive case. Phrased differently, once we have an algorithm that can compute this kind of *partial information* we can always turn it into an algorithm that *decides* the given language. Depending on the problem, this may be an improvement as it is presumably simpler to find an algorithm for computing 'just' partial information than to find a decision algorithm.

This thesis presents a general framework for the study of partial information, which was first suggested in Definition 5.5 of Beigel *et al.* (1995a) and put onto a firm theoretical foundation by Nickelsen (1997, 1999). The basic idea is to *represent partial information combinatorially* and to argue in the combinatorial domain rather than to argue about languages. With the general theory that has been established for this framework, many questions about partial information boil down to finite combinatorial problems. For example, once we construct a *combinatorial representation* of Rose's form of partial information, Trakhtenbrot's result becomes an easy consequence of the theory of partial information.

Combinatorial descriptions of notions of partial information are called *families*. Families are made up of *pools*, which are sets of *possible characteristic vectors* for a tuple of words. For example, a pool might contain all bitstrings differing at no more than one position from some fixed bitstring b. If this is a set of possible characteristic vectors for a tuple of words, then the characteristic vector of the words is b with an error margin of one bit. Hence Rose's frequency computations for $r = 1$ can be described by the family of all pools of this form.

Notions of partial information studied in the literature include approximable or non-superterse languages, cheatable languages, selective languages, easily sortable languages, as well as cardinality and frequency computations. All of these—and others—can be described by families and hence the general theory of partial information applies to them.

The first aim of this thesis is to further our understanding of the combinatorial properties of pools and families and their relationship to the corresponding notions of partial information.

Once we fix a notion of partial information, all languages for which such partial information can be computed form a *partial information class*. Apart from the problem how partial information classes relate with respect to inclusion and equality, we will also be interested in their *relative difficulty*.

A successful approach to the study of the relative difficulty of languages has been the study of reductions. Relative difficulty can be quantified by examining which reductions are possible between languages and which are not. For example, knowing that a language in some partial information class is reducible to languages in another class with, say, five queries, but never with four or less queries, nicely pinpoints their relative difficulty.

The second aim of this thesis is to demonstrate, that reduction closures of partial information classes can also be represented combinatorially.

Once representations are available, many reduction closure and reduction hierarchy problems concerning partial information classes reduce to finite combinatorics. For example, using combinatorial arguments, we can identify the partial information classes *closed* under bounded truth-table reductions. As another example, I give a novel combinatorial proof that the truth-table closures of the p-selective languages form a strict hierarchy.

Overview and Main Results

This thesis is structured into two parts. The first part treats *partial information classes*, while the second part treats *their truth-table closures*. Both parts are structured similarly: First, the treated class domains are introduced, partial information classes in the first part and their truth-table closures in the second; next, the class domains are *translated into combinatorial domains*. The remaining chapters in each part investigate the *structure of the combinatorial domains*. Due to the tight link between class domains and combinatorial domains, all combinatorial results directly reflect properties of partial information classes or of their truth-table closures.

Please note, that all notations used in the main text are explained in detail in the list of notations, starting on page 102.

Part I — Combinatorial Representations of Partial Information Classes

1 Introduction to Partial Information

This chapter introduces three basic concepts. *Pools and families* are the basic combinatorial domains used in this thesis. *Cartesian and compositionally closed function classes* are a novel notion used to model the complexity of computing pools. Putting these two concepts together yields the definition of *partial information classes*. As applications, we analyse the partial information complexity of the satisfiability, reachability and circuit value problems.

It is shown, how several notions of partial information studied in the literature can be described using families. An analysis of the partial information complexity of the satisfiability problem due to Beigel, Kummer and Stephan yields that *no useful partial information at all can be calculated for this problem in polynomial time, unless* $\mathbf{P} = \mathbf{NP}$. Similar but weaker novel results are obtained for the reachability and circuit value problems.

2 Representation of Resource Bounded Partial Information

This chapter establishes a link between the class domain and the family domain for resource bounded partial information. First, the simple but useful notion of *subset closed families* is introduced. A much stronger concept are Nickelsen's *normal families*, which suffice to produce all partial information classes. A review is given of *recursively presentable* function classes, which are used to model resource boundedness.

It is shown, that *normal families uniquely represent partial information classes, provided the underlying function class is recursively presentable.* This extends a result of Nickelsen (1997) who considered only polynomial time.

3 Structure of Normal Families

Borrowing from linear algebra, this chapter starts with definitions of *generating systems*, *eigenpools* and *bases*. Bases serve as succinct descriptions of normal families. Next, *units* are introduced which are sets of pools which generate the same family. This chapter also treats changing input tuple lengths, which gives rise to the definition of *upward translations of families* which have been studied extensively in the literature though not under that name and not in a unified way.

It is shown, that *bases of normal families are almost uniquely determined* and that *all bases of a normal family are representative systems of an antichain in the unit poset.* Put together with the upward translation of families, this result allows to make some progress on the conjecture of Nickelsen (1999) that *inclusion is well-founded on the set of all partial information classes.*

4 Structure of Stable Families

This chapter makes some progress on the question of *stability of families.* Stable families are introduced as the analogue of normal families for unbounded resources. *Hard tuples* are introduced as word tuples for which partial information is hard to come by. Nevertheless, they can be useful for computing partial information for other words in the form of *hard remainder pools.* At the end of the chapter, all stable 2-families are identified.

It is shown, that *for any language either no tuple is hard or hard remainder pools can be computed for all words.* As an application, a generalised version of the *Generalised Non-Speedup Theorem* is proved.

Part II — Truth-Table Closures of Partial Information Classes

5 Introduction to Reductions

After a review of the most important reductions, this chapter introduces a framework, called *Ψ-reductions*, for the study of bounded truth-table reductions. The *evaluation types* Ψ are sets of Boolean functions.

It is shown, that *most notions of query bounded reductions studied in the literature are Ψ-reductions for appropriate evaluation types Ψ.* It turns out that *even query bounded Turing reductions can be modelled by appropriate evaluation types.*

6 Representation of Truth-Table Closures

This chapter establishes a link between truth-table closures of partial information classes and a special property of pools, namely the *cone property.* A *cone* is a pool whose image under products of Boolean functions lies within a given family.

It is shown, that *the problem, whether a specific truth-table closure of some partial information class is contained in some other class, can be turned into* *a problem concerning the cone property of pools*. This result will be called the *Cone Theorem* in the following. As applications, some simple closure properties of partial information classes are established combinatorially, which have previously been proved directly in the literature.

7 Structure of Selective, Bottom and Top Cones

This chapter treats the question, for which families all selective pools have the cone property. The basic tool for the description of these families are *walks with bounded change numbers*, which are walks on the hypercube $\mathbb{B}^n$ where each position may be 'bit-flipped' only a fixed number of times. The chapter concludes with a translation of the established results to bottom and top partial information.

It is shown, that *the selective pools are all Φ^k-cones exactly over those families, which contain all walks whose change numbers are at most k. The lengths of maximal walks with different change numbers differ*. A direct consequence is that *all bounded truth-table closures of the p-selective languages differ*. This was first proved by Hemaspaandra *et al.* (1996) who used a direct diagonalisation argument.

8 Structure of Cones over Families

This chapter treats the question, which pools are cones over a given family. The main focus is on the investigation of the cone property of upward translated families. For bounded truth-table reductions, the cone property of upward translated families is fully characterised.

It is shown in the first section, which *partial information classes are closed under bounded truth-table reductions*. The second section shows that *approximable partial information is closed under bounded truth-table reductions*—and the third section shows that *this is result is optimal for polynomial time*. This latter result is a strengthened version of a result due to Beigel *et al.* (1994), who showed that approximable polynomial time is not closed under truth-table reductions.

Part I

Combinatorial Representations of Partial Information Classes

§ 6 Was beim Rechnen erfordert werde

Das *Rechnen* ist die Wissenschaft, Größen zu mehren oder zu mindern und aus bekannten unbekannte zu finden.

Hierzu ist Beurtheilungskraft und Gedächtnis gleich nothwendig.

Erstere muß bei jeder Rechnung das Verhältnis beurtheilen lehren, in welchem wir die verschiedenen Dinge, die zu berechnen sind, betrachten und gegeneinander halten sollen. Das *Gedächtnis* hingegen muß uns helfen, die Zahlen, nach den bekannten Rechnungsarten zu mindern oder zu mehren.

Beurtheilungskraft bleibt immer der Gegenstand eines denkenden Wesens, das durch keine Maschine erzeugt wird; wogegen man dem Gedächtnis auf mancherlei Art zu Hülfe kommen kann.

§ 7 Was eine Rechenmaschine leisten könne

Hieraus folgt, daß auch bei einer Rechen-Maschine die Anwendung des Verstandes, im Ordnen der verschiedenen Rechnungs-Sätze, unentbehrlich ist, und daß man sich daher unter einer dergleichen Maschine kein solches Werkzeug denken dürfe, welches für sich selbst rechnet und demjenigen, der sich solcher bedient, nichts als die Mühe, es in Bewegung zu setzen, koste; sondern sich eine solche Maschine vorstellen müsse, die auch in ihrer größten Vollkommenheit, blos ein Erleichterungs-Mittel beim Rechnen seyn könne.

— Johann Paul Bischoff, *Versuch einer Geschichte der Rechenmaschine*, Ansbach 1804

1

Introduction to Partial Information

A. C. Jones in his paper "A Note on the Theory of Boffles",
Proceedings of the National Society, 13,
first defined a Biffle to be a non-definite Boffle
and asked if every Biffle was reducible.
C. D. Brown in "On a paper by A. C. Jones", Biffle, 24,
answered in part this question by defining a Wuffle to be a reducible Biffle
and he was then able to show that all Wuffles were reducible.

— A. K. Austin, *The Mathematical Gazette*, 51:149–150, 1967

Consider a language and let's say five words. For the *decision problem*, we are asked to decide which of these words are elements of the language—possibly within limited time or space. For instance, if the decision problem can be solved in deterministic polynomial time, the language is an element of the complexity class **P**.

Many interesting languages fail to be decidable in polynomial time. Still, for a given language and the five words we might be asked to compute *only a set of possible values for their characteristic string*, such that the correct value is in it. Such a set will be called a *pool* and it represents some *partial information* about the language and the words. Intuitively, computing partial information appears to be simpler than deciding a language and indeed, as shall be shown, this weaker problem can be solved for more languages than the decision problem.

This chapter is divided into five sections. The first section presents the formal definitions of *pools* and *families* which are the basic combinatorial descriptions of partial information. The second section introduces *Cartesian and compositionally closed function classes* which are used to describe the complexity of computing pools. In the third section, we put together the introduced concepts and define *partial information classes*.

As an example application of these notions, the fourth section studies the partial information complexity of the satisfiability problem. Partial information turns out to be rather hard to compute for **NP**-complete problems. Beigel, Kummer and Stephan (1994) proved Theorem 1.23 below, which states that for them *no useful partial information at all* can be computed in polynomial time, unless they are decidable in deterministic polynomial time.

To conclude the chapter, the fifth section presents two novel theorems which treat the partial information complexity of two problems *inside* **P**.

Please note, that all notations used in the main text are explained in detail in the list of notations, starting on page 102.

1.1 Definition of Pools and Families

In this first section, Definition 1.1 defines a pool to be a subset of $\mathbb{B}^n$ and a family to be a covering of $\mathbb{B}^n$—these two basic definitions are due to Arfst Nickelsen and partly myself. The rest of the section presents numerous examples of families whose partial information classes have been studied in the literature. All of these families are studied in detail in Nickelsen's PhD thesis. At the end of the section, Table 1-3 on page 6 gives an overview of the families referred to in this thesis.

When computing partial information we will want to specify what kind of partial information—and hence which pools—we are interested in. For example, for any language and any n words the pool $\mathbb{B}^n$ of all bitstrings of length n necessarily contains their characteristic string—so the pool $\mathbb{B}^n$ represents rather *useless* partial information. A computed pool will be required to be an element of a set of *allowed* pools. Following Nickelsen (1997, 1999) such a set will be called a *family*.

In formal terms, the problem of calculating a given kind of partial information described by a family $\mathcal{F}$ is: Given a language L, is it possible to compute for any n given words $w_1, \dots, w_n$ a pool $P \in \mathcal{F}$ such that the characteristic string $\chi_L(w_1, \dots, w_n)$ is in P? If it is possible, we shall write $L \in \mathbf{REC}[\mathcal{F}]$; if the pool can even be computed in deterministic polynomial time, we shall write $L \in \mathbf{P}[\mathcal{F}]$.

Not every set of pools qualifies as a family. The trouble is, we must compute a pool such that the characteristic string of the given words is in it. Obviously, this is only possible if there *exists* such a pool in the set. If is does not, we cannot compute it. Hence, a set of n-pools is called a family only, if for every bitstring of length n there exists a pool in the set that contains this bitstring, i. e., if the set forms a *covering* of $\mathbb{B}^n$.

1.1 Definition (Pool, Family)
An *n-pool* is a subset of $\mathbb{B}^n$. An *n-family* is a covering of $\mathbb{B}^n$.

For a language L and words $w_1, \dots, w_n$ we will say *P is a pool for $w_1, \dots, w_n$ and L*, if $\chi_L(w_1, \dots, w_n) \in P$. In this case the pool represents *partial information* about the words with respect to the language.

1.2 Notation (Index of Families)
We write special n-families in capital letters with the number n written as an index, like in SEL_n. The number n will be called the family's *index*.

The index notation is different from the notation used in other publications which deal with families. There, the index is written in front of the family, like in 2-SEL, which can then easily be read as 'two-selective'. Unfortunately, most interesting families have *two* parameters, namely the index and another variable parameter. The notation $\text{SIZE}_5(2)$ clearly states that the index is five and some size parameter is two—while in the notation $(2,5)$-SIZE it is much harder to remember what is what.

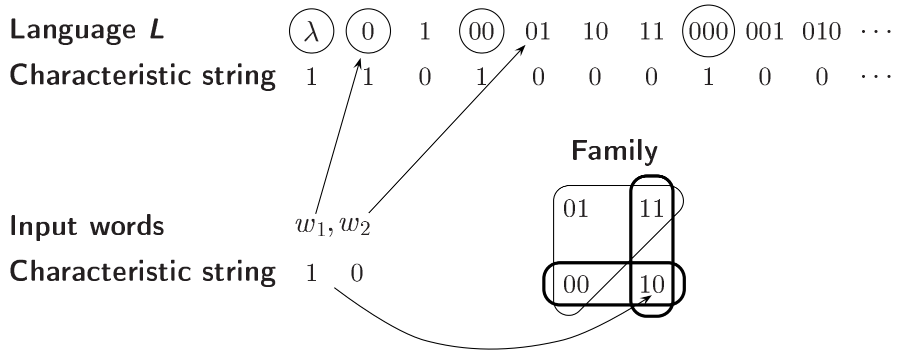


Figure 1-1
Computation of partial information using pools and families. In the top line, the tally language $L = \{\, 0^n \mid n \in \mathbb{N} \,\}$ is indicated by circles around its elements. For the two input words $w_1 = 0$ and $w_2 = 01$ their *characteristic string* is $\chi_L(w_1, w_2) = 10$. This string is an element of the two pools shown in bold. These two pools together with a third pool $\{00, 01, 11\}$ form a covering of $\mathbb{B}^2$ and hence a family.

The following Examples 1.3, 1.5 and 1.6 demonstrate three different ways of defining interesting families for a given index n. All of the families in these examples were introduced in Nickelsen (1997). In the first example, the *sizes* of pools are restricted in various ways. Next, we can define families in terms of *order theoretical* and *topological* properties of $\mathbb{B}^n$.

The set $\mathbb{B}^n$ is a Boolean algebra with the bitwise and, bitwise or and the bitwise negation as operations. Every Boolean algebra induces a partial ordering of its elements—for $\mathbb{B}^n$ this results in what will be called the *pointwise ordering*, defined by $b \leq_{\text{pw}} c$, iff $b\texttt{[}i\texttt{]} \leq c\texttt{[}i\texttt{]}$ for all positions i.

Topological properties are induced by the *Hamming distance* $\mathrm{d}(b, c)$ between two bitstrings b and c, defined as the number of bits where these bitstrings differ. The Hamming distance yields the Hamming spaces $(\mathbb{B}^n, \mathrm{d})$, which are metric spaces visualised as hypercubes in Figure 1-2 on the next page.

1.3 Example (Size families)

The most natural way to restrict pools in families is to restrict their size. For a size $s \geq 1$ we define $\text{SIZE}_n(s) := \{\, P \subseteq \mathbb{B}^n \mid |P| \leq s \,\}$ as the set of all n-pools which contain a maximum of s bitstrings.

Among the size families, some families are special. First of all, $\text{SIZE}_n(2^n - 1)$ is the largest family which does not contain the pool $\mathbb{B}^n$ which renders a family useless. This largest family will be denoted APPROX_n.

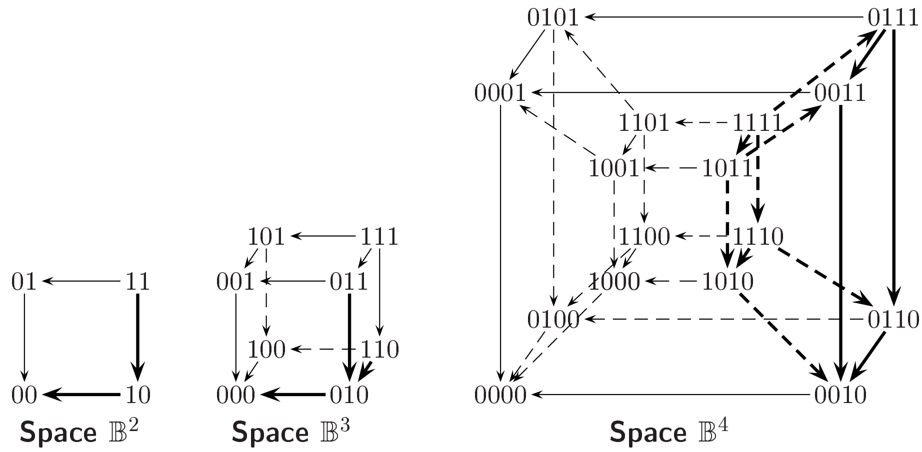


Figure 1-2
The Hamming spaces $\mathbb{B}^2$, $\mathbb{B}^3$ and $\mathbb{B}^4$ visualised as hypercubes. The length of the shortest path between two vertices is their Hamming distance. Arrows indicate the pointwise ordering. In the left algebra, the vertices connected by the bold arrows form a maximal chain. Together with the other maximal chain, these two pools make up the maximal pools of the selective family. Inside the middle algebra, the closed ball of radius 1 and diameter 2 around 010 is shown. To the right, a pool of diameter 3 is shown.

Some especially well-behaved families—despite their name—are the cheatable families, defined by $\text{CHEAT}_n := \text{SIZE}_n(n)$. The reason for the unsavoury name is explained in a small anecdote on page 39 after Lemma 3.20. □

Size families are important for the analysis of partial information as they often give a 'first impression' of the complexity of a problem. As we will often be interested in the smallest size family that contains a given family, the following definition introduced by Ronneburger (1998) is useful:

1.4 Definition (Maximum Pool Size)

For a family $\mathcal{F}$ let $\tau_{\mathcal{F}}$ denote the size of the largest pool in $\mathcal{F}$.

The next example introduces the selective families, whose partial information classes—which habe been most influential—were introduced by Alan Selman in 1979.

1.5 Example (Selective families)

Recall that a chain in a partial ordering is a linearly ordered subset. The *selective families* SEL_n are the sets of all chains in $(\mathbb{B}^n, \leq_{\text{pw}})$. The two maximal chains in $\mathbb{B}^2$ are $\{00, 01, 11\}$ and $\{00, 10, 11\}$. If one of these maximal chains is a pool for two words, one of the two bitstrings 10 and 01 is excluded as possible value of their characteristic string. Examples 1.12 and 1.16 will demonstrate how this information can be put to good use. □

1.6 Example (Topological families)

The *diameter* of a subset of a metric space is the supremum of the distances of the subset's elements. As $\mathbb{B}^n$ is a metric space when equipped with the

Hamming metric, we can talk about the *diameter of a pool*. We will be especially interested in the family consisting of all pools of maximal diameter 1. A pool of diameter 1 cannot contain more than two different bitstrings and these cannot differ at more than one position. As the next chapter will show that this family is the *minimal* non-trivial *weakly* normal family of index n, we call it WEAKMIN_n.

Alternatively, we can also consider pools which are closed balls in the metric space. The *closed ball* around a bitstring b with radius r is the set $\mathrm{B}_r(b) := \{ c \in \mathbb{B}^n \mid \mathrm{d}(b, c) \leq r \}$. The set of all pools contained in closed balls of radius r around bitstrings in $\mathbb{B}^n$ forms the *frequency family* $\text{FREQ}_n(r)$. □

Table 1-3
List of important families. Definitions of many more families can be found in Nickelsen (1999) where all of the below families are studied in detail. Recall that $\mathrm{B}_r(b)$ is the closed ball of radius r around the bitstring b. An eigenpool is obtained from a pool by removing all constant and all duplicate columns—see Definition 3.4 on page 30 for details.

Family	Property of its Pools
APPROX_n	Non-trivial, i. e., not $\mathbb{B}^n$.
BOTTOM_n	Eigenpool contained in $\mathrm{B}_1(0^m)$ for appropriate m.
$\text{BOTTOM}_n(s)$	Eigenpool contained in $\mathrm{B}_s(0^m)$ for appropriate m.
CHEAT_n	Size bounded by n.
COSMC_n	Does not contain $\mathrm{B}_1(0^n)$.
$\text{FREQ}_n(r)$	Contained in a closed ball of radius r.
MIN_n	Eigenpool is $\mathbb{B}$ or empty.
SEL_n	Chain in $\mathbb{B}^n$.
$\text{SIZE}_n(s)$	Size bounded by s.
SMC_n	Does not contain $\mathrm{B}_1(1^n)$.
TOP_n	Eigenpool contained in $\mathrm{B}_1(1^m)$ for appropriate m.
$\text{TOP}_n(s)$	Eigenpool contained in $\mathrm{B}_s(1^m)$ for appropriate m.
WEAKMIN_n	Diameter bounded by 1.

1.2 Definition of Cartesian and Compositionally Closed Function Classes

Which complexity classes can be relaxed to *partial* information classes? Asked differently, what complexity classes can be considered for the *complexity of computing pools*? In the literature, only deterministic polynomial time and recursiveness have been considered. It seems natural enough to apply the general concept of partial information to other classes as well like, say, logarithmic space. In his PhD thesis, Nickelsen pointed out that most results on partial information hold if we replace 'polynomial time' with other complexity classes.

In this section, *Cartesian and compositionally closed function classes*, c.c.c. in short, are proposed as a new general model of computation for partial information. Example 1.9 demonstrates that not only **FP**, but also many other well-known function classes are Cartesian and compositionally closed.

To compute partial information from a family $\mathcal{F}$, we would like to use functions of the form $f\colon (\Sigma^*)^n \to \mathcal{F}$. The idea is that *we can compute partial information from the family $\mathcal{F}$ with the function f for a language L*, if for all words $w_1, \dots, w_n$ we have $\chi_L(w_1, \dots, w_n) \in f(w_1, \dots, w_n)$.

The *complexity of computing pools* can be described by specifying a function class from which the function f may be drawn. Unfortunately, allowing the functions to be drawn from *arbitrary* function classes inhibits our proving anything useful about partial information. What we need are basic closure properties of the function classes. Definition 1.7 below lists the properties we will need. Fortunately, these turn out to be very basic indeed—you should have no problem proving that your pet complexity class satisfies them.

Common function classes, like the class of all functions computable in polynomial time or the class of all recursive functions, are typically given as a class of functions mapping words to words. In order to use these standard classes conveniently we will consider functions of the form $f\colon \Sigma^* \to \Sigma^*$ as elements of our function classes, instead of functions of the form $f\colon (\Sigma^*)^n \to \mathcal{F}$.

Naturally, we then have to be able to encode a tuple of words into a single word as well as to encode a pool into a word. For the encoding of word tuples, for each n we use a function $\langle \cdot, \dots, \cdot \rangle : (\Sigma^*)^n \to \Sigma^*$ implemented by writing its parameter words alongside each other, but doubling all bits in the words and inserting a 01 stop sequence between words. Tuples can be decomposed using the projection functions $\mathrm{p}_i\colon \Sigma^* \to \Sigma^*$, which map tuples to their i-th component and ill-formed words to the empty word. An n-pool can easily be coded as a word by writing its bitstrings alongside each other.

1.7 Definition (Cartesian and Compositionally Closed Function Class)
A function class $\boldsymbol{FC}$ of functions from Σ^* to Σ^* is *Cartesian and compositionally closed*, c.c.c. in short, if

1 for all $f, g \in \boldsymbol{FC}$ we have $f \times g \in \boldsymbol{FC}$, where $(f \times g)\big(\langle u, v\rangle\big) := \big\langle f(u), g(v)\big\rangle$ and $(f \times g)(w) := \lambda$, if w codes no pair of words.

2 for all $f, g \in \boldsymbol{FC}$ we have $f \circ g \in \boldsymbol{FC}$,

3 we have $\mathbf{FL} \subseteq \boldsymbol{FC}$.

1.8 Lemma (Closure under Tupling)
Let $\boldsymbol{FC}$ be c.c.c. Then for all $f, g \in \boldsymbol{FC}$ we have $\langle f, g\rangle \in \boldsymbol{FC}$, where $\langle f, g\rangle(w) := \langle f(w), g(w)\rangle$.

Proof. The function $d\colon \Sigma^* \to \Sigma^*$ with $d(w) := \langle w, w\rangle$ is in $\boldsymbol{FC}$ as it is clearly computable in logarithmic space. Hence if $f, g \in \boldsymbol{FC}$, by the closure properties $(f \times g) \circ d \in \boldsymbol{FC}$. But $(f \times g) \circ d = \langle f, g\rangle$. □

1.9 Example (Logarithmic and polynomial space are c.c.c.)
The class **FL** of functions computable in logarithmic space is c.c.c. The only difficulty lies in proving that the composition of two functions $f, g \in \mathbf{FL}$ is still computable in logarithmic space. We cannot simply compute $f(w)$ and then use g to compute $g(f(w))$, as the output word $f(w)$ may have polynomial length and hence a machine that computes $g \circ f$ this way would have to remember a *polynomially long temporary string*—which is clearly unacceptable for logarithmic space. The idea, see Proposition 8.2 of Papadimitriou (1994) for details, is not to remember this string at all. Instead, whenever the simulation of g needs some bit of the word $f(w)$, we simply recalculate this single bit. From a practical point of view this appears to be a very inefficient way to compute $g \circ f$, however, it is very efficient with respect to the space used.

Polynomial space is also c.c.c. provided that we limit it to functions whose output has length polynomial in its input. If we do not add this limitation, then the function class would not be closed under composition, as the function f which maps a word w to $2^{|w|}$ many 0's is then computable in polynomial space, while the function $f \circ f$ which maps w to $2^{2^{|w|}}$ many 0's is not. □

Considering only functions from Σ^* to Σ^* makes several arguments easier in the following, but it has the drawback that we must differentiate between a pool and its coding. The following notation is intended to remedy this problem.

1.10 Notation (Pools as Results of Function Application)
For a function $f\colon \Sigma^* \to \Sigma^*$ and words $w_1, \dots, w_n$ we write $f(w_1, \dots, w_n)$ for the pool coded by the bitstring $f\big(\langle w_1, \dots, w_n\rangle\big)$.

1.3 Definition of Partial Information Classes

Pools and families describe notions of partial information. Cartesian and compositionally closed function classes model the complexity of computing pools. Putting them together yields *partial information classes* which were first proposed by Beigel *et al.* (1995a) for the recursive case and transferred to polynomial time by Nickelsen (1997).

1.11 Definition (Partial Information Class)
Let $\boldsymbol{FC}$ be c.c.c. and let $\mathcal{F}$ be an n-family. The *partial information class $\boldsymbol{C}[\mathcal{F}]$ over $\mathcal{F}$ and $\boldsymbol{FC}$* consists of all languages L for which there exists a function $f \in \boldsymbol{FC}$ such that for all words $w_1, \dots, w_n \in \Sigma^*$ we have

$$\chi_L(w_1, \dots, w_n) \in f(w_1, \dots, w_n) \in \mathcal{F}.$$

One might wonder why we write $\boldsymbol{C}[\mathcal{F}]$ instead of perhaps $\boldsymbol{FC}[\mathcal{F}]$. The notation is intended to emphasise that although we need a function class to compute pools, a partial information class contains *languages*, not functions. However, one should keep in mind that the definition uses function classes 'internally'.

Sometimes, we will also consider the situation where we are only asked to compute pools for *distinct* words for some language L. Following Nickelsen, this case will be denoted $L \in \boldsymbol{C}_{\text{dist}}[\mathcal{F}]$. Intuitively, this appears to be marginally easier than computing pools for *any* words and indeed, there are families $\mathcal{F}$ for which there exist languages which are an element of $\boldsymbol{C}_{\text{dist}}[\mathcal{F}]$ but not of $\boldsymbol{C}[\mathcal{F}]$. A proof for this claim will have to wait till the next chapter which studies the separation of partial information classes in detail.

The next three examples present three special cases of partial information classes which are fundamentally different in many regards, as will be proved *peu à peu* in the following three chapters.

1.12 Example (Dedekind cuts are selective, but not necessarily recursive)
Consider a linear ordering $\leq$ of Σ^* which is decidable in polynomial time. A typical such ordering is the *lexicographical* or *dictionary ordering* $\leq_{\text{lex}}$ of Σ^*. A non-empty set $D \subsetneq \Sigma^*$ is called a *Dedekind cut* of a linear ordering $\leq$, if $u \leq v \in D$ always implies $u \in D$, see Figure 1-4.

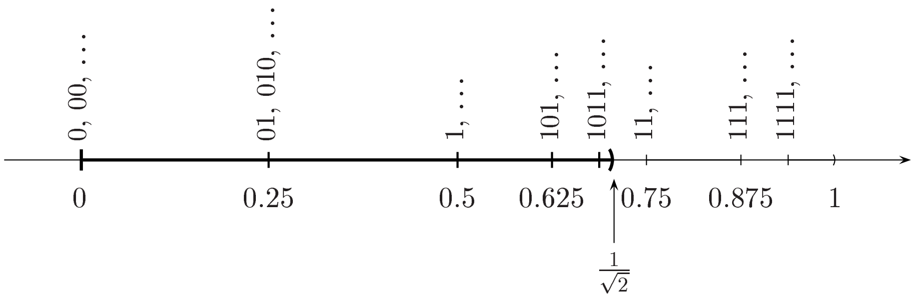


Figure 1-4
Dedekind cut induced by the square root of 1/2. The bitstrings above the line are mapped to the rational numbers below the line. The lexicographical ordering of the bitstrings corresponds to reading the bitstrings from left to right. The bitstrings in the Dedekind cut for $1/\sqrt{2}$ are all those bitstrings which are left of this number. Note, that there exists no bitstring which is interpreted as $1/\sqrt{2}$ directly.

Recall that the *selective family* SEL_2 is the set of all chains in $\mathbb{B}^2$. Selman (1981), Theorem 1 page 329, proved that *every Dedekind cut of the lexicographical ordering is in* $\mathbf{P}[\text{SEL}_2]$. To see this, consider any two words u and v and a machine that checks $u \leq_{\text{lex}} v$. If $u \leq_{\text{lex}} v$, then if $v \in D$ so is $u \in D$ as D is a Dedekind cut. In this case, it is not possible that $\chi_D(u, v) = 01$. Thus, $\{00, 10, 11\}$ is a pool for u and v, if $u \leq_{\text{lex}} v$. Likewise, if $v \leq_{\text{lex}} u$ then the set $\{00, 01, 11\}$ is a pool for u and v. This shows that for any two words u and v the machine can calculate in polynomial time a pool for these words and hence there exists a function computable in polynomial time that maps any pair of words to a pool from SEL_2 for them. But then by definition $D \in \mathbf{P}[\text{SEL}_2]$.

One can map Σ^* in a natural way into the interval $[0, 1)$ by mapping for example 0011 to 0.0011 in binary or 0.1875 in decimal. By a standard argument from calculus, see for example Proposition 5.3 of Forster (1976), every real number in the interval $[0, 1)$ is the supremum of the image of some Dedekind cut; which shows in turn that there are uncountably many Dedekind cuts *and thus* $\mathbf{P}[\mathrm{SEL}_2]$ *is not recursive.* ◻

A Dedekind cut of the lexicographical ordering is also called a *standard left cut* in the literature. The more general *left cuts* have been used to study the computational complexity of real numbers, see for example Selman (1981), where it is also shown that standard left cuts are in $\mathbf{P}[\mathrm{SEL}_2]$. However, the original idea for this proof came from McLaughlin and Martin who showed, see Jockusch (1968), that every Dedekind cut is semirecursive, i. e., in $\mathbf{REC}[\mathrm{SEL}_2]$. As we just saw, their construction works just as well for polynomial time—and also for logarithmic space for that matter.

1.13 Example (Computing partial information for closed chains)
The notion of a Dedekind cut can be generalised, by replacing *linear orderings* with *partial orderings*. For a partial ordering $\leq$ of Σ^* decidable in polynomial time, we can still consider languages L which have the property that $u \leq v \in C$ implies $u \in C$. As pointed out below, such languages are *closed sets* in a natural topology on the partial ordering and will hence be called *closed languages*.

For a closed language and two words u and v we can easily produce a pool from the selective family, if $u \leq v$ or $v \leq u$. Unfortunately, if the words are incomparable with respect to the partial ordering, either word may or may not be an element of the language. However, if we restrict the languages to *chains*, then for incomparable words *at most one word can be an element of a chain*. This can be expressed by asserting that $\{00, 01, 10\}$ is a pool for these words. To sum up, *all closed chains are elements of* $\mathbf{P}\big[\mathrm{SEL}_2 \cup \big\{\{00, 01, 10\}\big\}\big]$.

An important canonical example of a partial ordering of words is the prefix ordering $(\Sigma^*, \sqsubseteq)$ which is best envisioned as a large *tree* with the empty word as root, see Figure 4-2 on page 55. The infinite closed chains in this tree are exactly its *branches*. Such branches will be treated in more detail in Section 4.4, where we prove that there exist branches which are not selective. ◻

The languages just introduced are called *closed* because they are, indeed, closed sets in a topology $\mathcal{T}_{\leq}$ over the set Σ^*. More precisely, on a partial ordering $(S, \leq)$ we define a topology $(S, \mathcal{T}_{\leq})$ as follows: A subbasis is given by all sets of the form $\{\, y \mid x \leq y \,\}$ for some $x \in S$. Then, a closed set in this topology is a set for which $x \leq y \in S$ implies $x \in S$.

The partial information classes treated in the two previous examples were not recursive. While this may appear to be a bit alarming, it turns out that these classes are not as large as it may seem—for example in the next section we will show that neither of them contains $\mathbf{NP}$-hard languages, unless $\mathbf{P} = \mathbf{NP}$.

The next example presents a more well-behaved partial information class—the minimum non-trivial polynomial time partial information class. Despite having been introduced by Nickelsen (1997) as a spin-off of the combinatorial theory of partial information, it appears to be interesting in its own right.

1.14 Example (But one polynomial languages)
Let $\mathcal{B}$ be the 2-family of pools of diameter 1, i. e., let $\mathcal{B} = \big\{\{00, 01\}, \{00, 10\}, \{01, 11\}, \{10, 11\}\big\}$. The family contains all pools consisting of pairs of bitstrings which agree at one position and differ at the other. Let $L \in \mathbf{P}_{\mathrm{dist}}[\mathcal{B}]$ and let

u and v be different words. Then, by definition, it is possible to compute in polynomial time a pool from $\mathcal{B}$ for these two words. In such a pool, the bitstrings agree on one position. Hence, either $\chi_L(u)$ or $\chi_L(v)$ is actually computed.

Consider a constant number of distinct words $w_1, \dots, w_n$. Using $L \in \mathbf{P}_{\text{dist}}[\mathcal{B}]$ we can compute either $\chi_L(w_1)$ or $\chi_L(w_2)$. For the word whose characteristic value we failed to compute, assume it to be w_2, we compute the characteristic value of either $\chi_L(w_2)$ or $\chi_L(w_3)$. Next, we couple the failed word with w_4 and so on. After n steps, all characteristic values *but one* will have been computed. For *any finite set of words* we can *decide all words but one* in time polynomial in the total length of these words. Because of that, such a language will be called *but one in* $\mathbf{P}$.

But one polynomial languages are recursive. This will follow from the theory established in Chapter 4, but can also easily be seen directly: For a given word u simply ask more and more words v. Sooner or later the characteristic value of u will be output; and if it is never output, hardwire this value and the values of all other v become known instead. ▯

Families describe notions of partial information combinatorially. Unfortunately, some concepts studied in the literature cannot be described directly by a single family. For example, while the concept of being *n-approximable* as introduced in Beigel *et al.* (1994) is directly represented by the family APPROX_n, the concept of being *approximable* is not represented by a single family. A language is called approximable, if it is n-approximable for some sufficiently large n. As every n-approximable language is also $(n+1)$-approximable, a language is approximable, iff it is in $\mathbf{P}[\text{APPROX}_n]$ *for almost all n*. Like things happen for many other notions of partial information studied in the literature: A language is *cheatable*, if it is n-cheatable for almost all n, *easily countable*, if it is easily n-countable for almost all n, and so forth. This motivates the following suggestive notation:

1.15 Notation (Families Without an Index)
For families $\text{FAM}_1, \text{FAM}_2, \text{FAM}_3, \dots$ a language is in $\boldsymbol{C}[\text{FAM}]$, if it is in $\boldsymbol{C}[\text{FAM}_n]$ for almost all n.

As $\boldsymbol{C}[\text{FAM}] = \bigcup_{m \in \mathbb{N}} \bigcap_{n > m} \boldsymbol{C}[\text{FAM}_n]$, the notation actually describes the set algebraic *limes inferior* of the sequence $\boldsymbol{C}[\text{FAM}_1], \boldsymbol{C}[\text{FAM}_2], \boldsymbol{C}[\text{FAM}_3], \dots$ of classes.

1.4 Computing Partial Information for Satisfiability

This section examines how much partial information can be computed for the satisfiability problem, which is an $\mathbf{NP}$-complete problem by Cook's Theorem. It turns out, that it is rather hard to compute—unless $\mathbf{P} = \mathbf{NP}$, it is not possible to compute *any* useful partial information about *any* $\mathbf{NP}$-complete problem.

Stated more friendly, for many languages being able to compute useful partial information implies being able to decide the language: The partial information can be 'boosted' to full information, using some clever algorithms.

This section is divided into four examples. Each example uses less partial information than the previous one. While the first example was already included in the very first paper on p-selective sets, see Selman (1979), the last and most powerful example is due to Beigel *et al.* (1994).

1.16 Example (Satisfiability is not p-selective, unless P = NP)
Assume, that for the satisfiability problem we can compute partial information from the selective family in polynomial time, i.e., assume $\mathrm{SAT} \in \mathbf{P}[\mathrm{SEL}_2]$. We claim, that we can then decide satisfiability in deterministic polynomial time.

Consider some word w which is interpreted as a formula ϕ. We must compute $\chi_{\mathrm{SAT}}(w)$ in deterministic polynomial time, i.e., we must check if ϕ is satisfiable. Starting from ϕ we check if ϕ contains at least one variable x. If not, then ϕ can be evaluated in polynomial time and we are done. Otherwise, we construct two new formulas ϕ_0 and ϕ_1, where we replace all occurrences of x inside ϕ with 0 and 1 respectively.

Assuming $\mathrm{SAT} \in \mathbf{P}[\mathrm{SEL}_2]$, we can compute selective partial information for the two formulas ϕ_0 and ϕ_1. More precisely, we can compute a pool P from SEL_2 with $\chi_{\mathrm{SAT}}(\phi_0, \phi_1) \in P$. Such a pool excludes (at least) one of the two bitstrings 01 and 10. Assume that the first bitstring is excluded—the other case is symmetric. Spelled out, this means: 'It is not possible, that the formula ϕ_0 is unsatisfiable, but ϕ_1 is satisfiable.' As ϕ is satisfiable, iff either ϕ_0 or ϕ_1 is satisfiable, we conclude that ϕ is satisfiable, iff ϕ_0 is satisfiable.

Using the postulated selectivity of satisfiability we have reduced—in deterministic polynomial time—the satisfiability problem for the formula ϕ to the satisfiability problem for the formula ϕ_0 which has *one variable less* than ϕ. Iterating this reduction yields a polynomial time algorithm for deciding satisfiability. Hence, we proved the following Theorem 3 of Selman (1979). ∎

1.17 Theorem
If $\mathrm{SAT} \in \mathbf{P}[\mathrm{SEL}_2]$ *then* $\mathbf{P} = \mathbf{NP}$.

1.18 Example (Satisfiability is neither selective nor top, unless P = NP)
In the previous example, we allowed only selective pools to be output. However, there is another pool which would provide us with useful information, namely the pool $\{01, 10, 11\}$. This pool is missing the bitstring 00 only. If this is a pool for the formulas ϕ_0 and ϕ_1 from the last example, then we know that *at least one* of the two formulas is satisfiable and hence ϕ is satisfiable.

If $\mathrm{SAT} \in \mathbf{P}[\mathrm{SEL}_2 \cup \{01, 10, 11\}]$ *then* $\mathbf{P} = \mathbf{NP}$. ∎

1.19 Example (Satisfiability is not COSMC, unless P = NP)
The idea of the previous example can be extended further. Starting from ϕ we get two shorter formulas ϕ_0 and ϕ_1 such that ϕ is satisfiable, iff either of these two formulas is satisfiable. Next, from ϕ_0 we produce two new even simpler

formulas ϕ_{00} and ϕ_{01} by substituting another variable with 0 and 1 respectively. Likewise, from ϕ_1 we produce ϕ_{10} and ϕ_{11}. In the four resulting formulas, we substitute a further variable, yielding eight formulas. We break the expansion when we either no longer have any variables to substitute or if in some level we exceed a fixed number of n formulas. At any point, the original formula ϕ is satisfiable, iff at least one of the generated formulas is satisfiable.

Assume that we know that some generated formula ϕ_b has the property that it is *not the only satisfiable formula generated*. Phrased differently, assume that either ϕ_b is not satisfiable or, if ϕ_b is satisfiable, so is another of the formulas. In any case, we can simply *ignore* the formula ϕ_b, because ϕ is satisfiable, iff at least one formula *other than* ϕ_b is satisfiable.

What we need is a procedure that produces the index b of a generated formula ϕ_b such that ϕ_b is not the only satisfiable formula. This can be expressed in the framework of partial information as follows: Assume that it is possible to compute for any n formulas a pool which misses at least one bitstring of the form 0^*10^*; then the position of the 1 in the missing bitstring denotes the index of a formula which is not the only satisfiable formula. Hence, under this assumption we can always reduce more than n formulas under consideration to only n formulas. It is now quite easy to extend this to an algorithm which decides satisfiability in deterministic polynomial time.

To sum up, consider the closed ball $\mathrm{B}_1(0^n)$ of radius 1 around the bitstring 0^n in the Hamming space $\mathbb{B}^n$. This pool contains all bitstrings of length n which have at most one 1. The algorithm just described will work fine, as long as the closed ball is *not* output; because then we do not know which formula we should drop.

We are interested in the family of all pools which do not contain $\mathrm{B}_1(0^n)$. The complementary notion, i. e., the family of all pools which do not contain $\mathrm{B}_1(1^n)$, has been studied in the literature under the name *strongly membership comparable*, see Definition 5.2 of Köbler (1995). We will denote the families, which describe the classes of strongly membership comparable languages, by SMC_n. This motivates the name COSMC_n for the family of all pools which do not contain $\mathrm{B}_1(0^n)$.

Recalling from Notation 1.15 that $\mathbf{P}[\mathrm{COSMC}]$ is the union of all $\mathbf{P}[\mathrm{COSMC}_n]$, we just argued:

If $\mathrm{SAT} \in \mathbf{P}[\mathrm{COSMC}]$ *then* $\mathbf{P} = \mathbf{NP}$. ☐

The argument just given works for languages other than SAT, too. Actually we only used the property of SAT that for any word w we find 'simpler' words w_0 and w_1 such that w is in SAT, iff either of the simpler words is in SAT. This property is called the disjunctive self-reducibility of the satisfiability problem; and we call an arbitrary language L *disjunctively self-reducible*, if for all words w we can compute—in deterministic polynomial time—two shorter words u and v such that w is in L iff either u or v is in L. For the below corollary, which is due to Hoene and Nickelsen (1993), note that cheatable pools never contain the closed ball $\mathrm{B}_1(0^n)$.

1.20 Theorem
All disjunctively self-reducible languages in $\mathbf{P}[\text{COSMC}]$ *are in* $\mathbf{P}$.

1.21 Corollary
All disjunctively self-reducible languages in $\mathbf{P}[\text{CHEAT}]$ *are in* $\mathbf{P}$.

1.22 Example (Satisfiability is not approximable, unless P = NP)
This concluding example presents Theorem 4.2 of Beigel *et al.* (1994) with a slightly modified argument.

We now only assume, that the satisfiability problem is approximable, i. e., we assume $\text{SAT} \in \mathbf{P}[\text{APPROX}_n]$ for some n. Recall, that this means that for any n words we can exclude at least one of the 2^n possible characteristic strings.

Basically, we will try to reduce this case to the idea of the previous example: Among a set of formulas we will try to find one formula which is not the only satisfiable formula. The basic difference is, that we now start the elimination process, not when we have more than n formulas but only if we have more than 2^n formulas. Note that n and hence 2^n are constants.

Assume that we have constructed 2^n formulas ϕ_1 to ϕ_{2^n}. We now try to find one formula which is not the only satisfiable formula among them. Construct a set S_1 which contains the first half of the formulas. Next, construct a set S_2 which contains the first quarter and the third quarter of the formulas. Let S_3 contain the first, third, fifth and seventh eighth of the formulas. In general, let S_i contain all those formulas whose index's i-th bit is 0. For each of these n sets, let ψ_i denote the logical or of all formulas in S_i. For example, we have $\psi_1 = \phi_1 \vee \dots \vee \phi_{2^{n-1}}$.

Assuming that the satisfiability problem is n-approximable, we can compute a non-trivial pool for the n formulas $\psi_1, \dots, \psi_n$. One bitstring b is excluded for the characteristic string of these formulas. We claim that *this bitstring b induces a formula ϕ_i which is not the only satisfiable formula*. Here, the index i is given by the index of the formula ϕ_i which is contained in all S_k where $b[k] = 1$ and is not contained in all S_k where $b[k] = 0$.

To prove this claim, for the sake of contradiction assume that ϕ_i is the only satisfiable formula. Then, in a set S_k with $\phi_i \in S_k$ there would be a satisfiable formula and hence ψ_k would be satisfiable. Likewise, in a set S_k with $\phi_i \notin S_k$ there would be no satisfiable formulas and hence ψ_k would not be satisfiable. This would imply that indeed $\chi_{\text{SAT}}(\psi_1, \dots, \psi_n) = b$, which is impossible.

As has been shown, we can still apply the algorithm of the previous example. This proves the *en suite* Theorem 4.2 of Beigel *et al.* (1994). ▯

1.23 Theorem
If $\text{SAT} \in \mathbf{P}[\text{APPROX}]$ *then* $\mathbf{P} = \mathbf{NP}$.

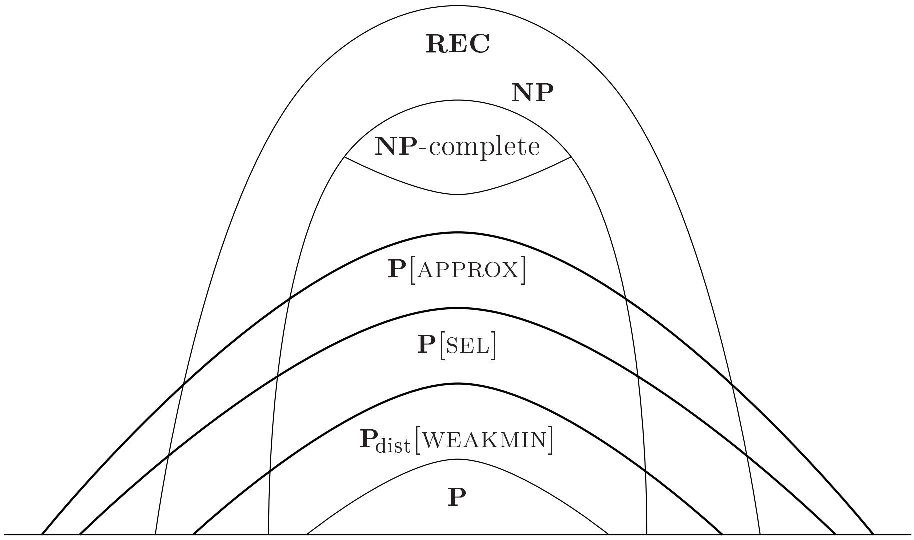


Figure 1-5
Class review for the results presented in this chapter. Assuming $\mathbf{P} \neq \mathbf{NP}$, by Theorem 1.23 no **NP**-complete problem is approximable. Examples 1.12 and 1.14 show that $\mathbf{P}[\text{SEL}]$ is not recursive while $\mathbf{P}_{\text{dist}}[\text{WEAKMIN}]$ is.

1.5 Computing Partial Information for Reachability and Circuit Value

This last section proves two new theorems intended to demonstrate that the usage of partial information in complexity theory need not and perhaps should not be limited to the polynomial time case as has generally been done in the literature. Partial information inside **P** is just as lively as outside.

The reachability problem is the following: Given a *directed* graph and two vertices s and t, does there exist a path from the first vertex to the latter? This problem is **NL**-complete, see Theorem 16.2 of Papadimitriou (1994). The circuit value problem is this: Given a circuit without input gates and with a single output gate, what is its value? This problem is **P**-complete, as show in Ladner (1975).

How much partial information can be calculated for these problems? Just as in the previous section, partial information turns out to be hard to come by unless certain complexity classes coincide. The other way round, some partial information can be boosted to full information for these two problems.

The first theorem states that we cannot compute selective partial information efficiently *in parallel* for the circuit value problem unless $\mathbf{NC} = \mathbf{P}$. The second states that the reachability problem is not selective in logarithmic space unless $\mathbf{L} = \mathbf{NL}$. Special thanks go to Arfst Nickelsen for pointing out that one can use the Immerman-Szelepcsényi Theorem here.

1.24 Theorem
If CIRCUIT-VALUE $\in \mathbf{NC}[\text{SEL}_2]$ *then* $\mathbf{NC} = \mathbf{P}$.

Proof. Assume CIRCUIT-VALUE $\in \mathbf{NC}[\text{SEL}_2]$. Starting from a word w, interpreted as a circuit C without input gates, we construct the negated circuit C' by adding a negation gate at the output. Then, we compute a selective pool P for C and C'. As exactly one of the two circuits evaluates to true and as there is exactly one bitstring in P with exactly one 1, the pool P tells us which circuit evaluates to true. Hence, CIRCUIT-VALUE $\in \mathbf{NC}$. □

The class **NC** is not used in any particular way in the proof and the theorem also holds if we replace **NC** by, say, logarithmic space. The above formulation is simply a 'weakest precondition' formulation.

1.25 Theorem
If REACHABILITY $\in \mathbf{L}[\text{SEL}_2]$ *then* $\mathbf{L} = \mathbf{NL}$.

Proof. Assuming REACHABILITY $\in \mathbf{L}[\text{SEL}_2]$, we argue that REACHABILITY is in $\mathbf{L}$. Let $G = (V, E)$ be a graph and let s and t be vertices from V. We must decide—in logarithmic space—whether t is reachable from s.

By the Immerman-Szelepcsényi Theorem, the *unreachability problem* is in **NL** via some non-deterministic machine M. Let $\mathrm{G}_M(G)$ be the *configuration graph of the machine M on input G*. By definition of the unreachability problem, the accepting state is reachable from the initial state, iff *there exists no path from s to t in G*. Note that $\mathrm{G}_M(G)$ has polynomial size.

Hence, if t is reachable from s in G, then $G \in$ REACHABILITY and $\mathrm{G}_M(G) \notin$ REACHABILITY and the other way round, if t is not reachable from s, then $G \notin$ REACHABILITY and $\mathrm{G}_M(G) \in$ REACHABILITY. In either case, the characteristic string of the tuple $\big(G, \mathrm{G}_M(G)\big)$ contains exactly one 1.

We compute a selective pool for $\big(G, \mathrm{G}_M(G)\big)$. As this pool contains only one bitstring which contains exactly one 1, we know which one is correct. Hence, we can decide REACHABILITY in deterministic logarithmic space. □

Bibliographical Notes

Partial information was first studied in recursion theory—although not under that name. Recursive frequency computations were introduced by Rose already back in 1960. However, the most influential form of partial information—namely selectivity—was introduced by Jockusch in his doctoral dissertation. He introduced *semi-recursive* languages which are exactly the languages in $\mathbf{REC}[\text{SEL}_2]$. Naturally, the original definition of semirecursive languages did not use the framework of pools and families. Rather, semirecursiveness was originally defined as follows:

> **Definition 3.1 from Jockusch (1968).** A set of natural numbers A is *semirecursive* if there is a recursive function f of two variables such that for every x and y
>
> 1 $f(x, y) = x$ or $f(x, y) = y$ and
>
> 2 $(x \in A \text{ or } y \in A) \implies f(x, y) \in A$.

Due to the success in distinguishing reduction closures of the recursively enumerable sets, Selman introduced the notion of the p-selective languages, which are exactly the languages in $\mathbf{P}[\text{SEL}_2]$. The original definition is a direct adaption of Jockusch's definition:

Definition from Selman (1979). Let Σ be a finite alphabet. A set A, $A \subseteq \Sigma^*$, is *p-selective* if there is a function $f\colon \Sigma^* \times \Sigma^* \to \Sigma^*$ so that

1 f is computable in polynomial time,

2 $f(x,y) = x$ or $f(x,y) = y$, and

3 $x \in A$ or $y \in A \implies f(x,y) \in A$.

Polynomially selective languages have since been studied extensively, starting with Selman (1982a,b). Especially their different reduction closures have been investigated—for details please refer to the second part of this thesis, especially Chapter 7.

Later on, many notions were proposed which can be fitted into the framework of partial information classes. In his PhD thesis Richard Beigel introduced the notions of cheatable and non-p-superterse or approximable languages; Hoene and Nickelsen introduced the easily countable languages in 1993; Kummer and Stephan adopted Trakhtenbrot's frequency computations to the polynomial case in 1991. For many more examples, please refer to Nickelsen (1999).

The first steps towards a unified treatment of partial information were taken in Beigel *et al.* (1995a) where the notion of strongly $\mathcal{D}$-verbose languages was introduced, which corresponds to $\mathbf{REC}[\mathcal{D}]$. These ideas were further developed by Nickelsen (1997, 1999). While the definition of pools and families, Definition 1.1 in this thesis, is exactly the same as in Nickelsen's PhD thesis, in Nickelsen's 1997 paper a set of pools is called a family only, if it is a covering of $\mathbb{B}^n$ and all its pools are maximal.

Definition 1.11 of partial information classes is more general than the corresponding Definitions 2.3 to 2.6 in Nickelsen (1999), where only $\mathbf{P}$ and $\mathbf{REC}$ are studied. Nickelsen pointed out quite correctly that the adaption of the concept of partial information to other function classes is often trivial. However, for some more exotic complexity classes like linear space as opposed to quadratic space, the definition of c.c.c. function classes helps to quickly decide whether the basic results on partial information really apply.

In Theorem 1.20, which states that disjunctively self-reducible languages in $\mathbf{P}[\text{COSMC}]$ are already in $\mathbf{P}$, the families COSMC_n may be replaced by several other families and the claim still holds. Many such other families can be found in Hoene and Nickelsen (1993). However, Theorem 7.1 of Beigel *et al.* (1994) shows that in some *relativised world* the family COSMC_2 cannot be replaced by APPROX_2.

While the usage of partial information to decide the satisfiability problem was already proposed by Selman in the same paper where he introduced the p-selective languages (Selman, 1979), to my knowledge partial information has not been used to investigate the complexity of problems in $\mathbf{P}$. Especially, Theorem 1.25—which states that no $\mathbf{NL}$-complete problem is selective in logarithmic space unless deterministic and non-deterministic logarithmic space coincide—has not been studied before.

To conclude, I would like to point out that the definition of partial information based on pools and families is often more fine-grained than the original definitions. For example, approximable and cheatable languages can also be defined in terms of query limited reductions. In the following definitions, F_k^A is the characteristic function of A extended to k input words, i.e., χ_A^k in the notation of this text. The set $\mathrm{PF}_{k\text{-T}}^A$

contains all function computable in polynomial time with k adaptive queries to the language A.

Definition 5.1 from Beigel (1991).

1 A set A is *k-query p-terse* if $\mathrm{F}_k^A \notin \mathrm{PF}_{(k-1)\text{-T}}^A$.

2 A set A is *p-terse* if A is k-query p-terse for all k.

3 A set A is *k-query p-superterse* if $(\forall X)\big[\mathrm{F}_k^A \notin \mathrm{PF}_{(k-1)\text{-T}}^X\big]$.

4 A set A is *p-superterse* if A is k-query p-superterse for all k.

Definition 5.2 from Beigel (1991).

1 An oracle A is *k-cheatable* if $(\exists X)\big[\mathrm{F}_{2^k}^A \in \mathrm{PF}_{k\text{-T}}^X\big]$.

2 An oracle A is *cheatable* if A is k-cheatable for some k.

A language is n-query p-superterse in the above sense, iff it is no element of the partial information class $\mathbf{P}[\mathrm{SIZE}_n(2^{n-1})]$. To see this, note that if a language is in $\mathbf{P}[\mathrm{SIZE}_n(2^{n-1})]$ then for any n words we can compute a pool of size 2^{n-1}. Hence, it takes $n-1$ bits to store the information which bitstring is the correct one, and this information can be coded into an oracle X which must be queried only $n-1$ times. For the other direction, simply note that if we can decide n words with $n-1$ queries to some oracle, we can compute 2^{n-1} possibilities for the characteristic string without querying the oracle at all.

A language that is n-cheatable in the sense above is in $\mathbf{P}\left[\mathrm{CHEAT}_{2^n}\right]$, because here, too, even without knowing the oracle we can always compute 2^n possibilities for the result of an n-Turing reduction. Vice versa, a language $L \in \mathbf{P}[\mathrm{CHEAT}_{2^n}]$ is n-cheatable in the sense above, because here for any n words we can compute a pool of size 2^n and it takes n bits of extra information to pick the correct bitstring from the pool. If we code this information into an oracle X, the function $\chi_L^{2^n}$ is even computable with n non-adaptive queries to X.

In later papers, like Beigel *et al.* (1992), the definitions were modified and formulated without the use of reductions. These modified definitions are the same as the ones used in this thesis.

2

Representation of Resource Bounded Partial Information

H. Greean, P. Smith and D. Jones
in their review of Brown's paper, Wuffle Review, 48,
suggested the name Woffle for any Wuffle other than the non-trivial Wuffle
and conjectured that the total number of Woffles
would be at least as great as the number so far known to exist.

They asked if this conjecture was the strongest possible.

— A. K. Austin, *The Mathematical Gazette*, 51:149–150, 1967

Families are combinatorial *descriptions*, but not necessarily *unique representations* of partial information classes. The trouble is that different families often describe the same partial information class. In order to *represent* partial information, the concept of *normal forms of families* due to Nickelsen (1997) is needed. This chapter presents Nickelsen's theory of normal forms and extends it to all recursively presentable c.c.c. function classes.

After the first section, which introduces the simple but useful concept of *subset closed families*, the second section defines *normal families* which are families in normal form. These are the basic building blocks of partial information. For every family there exists a normal family producing the same partial information classes for all c.c.c. function classes. More importantly, Nickelsen showed that for the function class $\mathbf{FP}$ for every n-family $\mathcal{F}$ there exists *exactly one normal n-family* $\mathcal{G}$ such that $\mathbf{P}[\mathcal{F}] = \mathbf{P}[\mathcal{G}]$. This theorem effectively reduces the equality problem for polynomial time partial information classes to a combinatorial problem, namely the problem of computing normal forms.

Nickelsen pointed out that polynomial time is not the only function class whose partial information classes have unique normal representations, but did not elaborate further on this point. We will make precise Nickelsen's remark in the following way: All partial information classes over *recursively presentable* c.c.c. function classes have unique normal representations. The third section reviews recursively presentable function classes in detail. The fourth section adapts Nickelsen's proof for polynomial time to the general setting.

2.1 Definition of Subset Closed Families

In this section, Definition 2.1 introduces subset closed families which were first proposed in Tantau *et al.* (1998) and which were also adopted in Nickelsen (1999). A family is *subset closed*, if it contains all subpools of all its pools. Lemma 2.2 below tells us that for every family there exists a subset closed family producing the same partial information class.

2.1 Definition (Subset Closure, Subset Closed, Maximal Pool)
The *subset closure* $\overline{\mathcal{F}}$ of a family $\mathcal{F}$ is the family $\{Q \mid \exists P \in \mathcal{F} \colon Q \subseteq P\}$. A family $\mathcal{F}$ is *subset closed* if $\mathcal{F} = \overline{\mathcal{F}}$. A pool in $\mathcal{F}$ is *maximal* if it is not contained in any other pool from $\mathcal{F}$.

2.2 Lemma
Let $\boldsymbol{FC}$ *be c.c.c. and let* $\mathcal{F}$ *be a family. Then* $\boldsymbol{C}[\mathcal{F}] = \boldsymbol{C}[\overline{\mathcal{F}}]$.

Proof. First, we have $\mathcal{F} \subseteq \overline{\mathcal{F}}$ and hence $\boldsymbol{C}[\mathcal{F}] \subseteq \boldsymbol{C}[\overline{\mathcal{F}}]$.

For the other direction, consider a language $L \in \boldsymbol{C}[\overline{\mathcal{F}}]$ and a function $f \in \boldsymbol{FC}$ that witnesses its membership. Whenever this function outputs a pool Q for a word tuple, by definition there exists a pool $P \in \mathcal{F}$ with $Q \subseteq P$. By the composition property of $\boldsymbol{FC}$, the function g which is identical to f, except that it outputs the pool P whenever f outputs the pool Q, is also an element of $\boldsymbol{FC}$. Hence, g witnesses $L \in \boldsymbol{C}[\mathcal{F}]$. □

Whenever we have a family $\mathcal{F}$ and a pool $P \in \mathcal{F}$, we get the same partial information class whether we add or remove any proper subpool of P. Only the *maximal* pools of a family are important with respect to the partial information class. The other way round, it does not hurt to add all pools that are contained in some maximal pool.

2.3 Example (*But one* languages revisited)
Recall the 2-family $\mathcal{B}$ of all pools of exact diameter 1 from Example 1.14. It consists of all pairs of bitstrings which agree on one position and disagree on the other. This family contains only maximal pools and we have

$$\mathcal{B} = \big\{\{00,01\},\{00,10\},\{10,11\},\{01,11\}\big\}\,,$$
$$\overline{\mathcal{B}} = \big\{\emptyset,\{00\},\{01\},\{10\},\{11\},\{00,01\},\{00,10\},\{10,11\},\{01,11\}\big\}\,.$$

By Lemma 2.2 these families produce the same partial information class. The subset closed family $\overline{\mathcal{B}}$ is exactly $\textsc{weakmin}_2$. ▯

In Tantau *et al.* (1998) and Nickelsen (1999) subset closed families are called subset *complete*. However, the subset closed families are, indeed, exactly the topologically closed sets with respect to the topology $(\mathcal{P}(\mathbb{B}^n), \mathcal{T}_\subseteq)$ introduced in the remark after Example 1.13. Furthermore, the *subset closure* of a family is the topological closure of the family considered as a subset of $\mathcal{P}(\mathbb{B}^n)$, and the set of the maximal pools of a family is exactly the *smallest dense subset* of the family. 'Closed' and 'complete' both being topological terms, these observation appear to justify the small change in terminology.

2.2 Definition of Normal Families

In this section, the important concept of normal families due to Nickelsen is introduced. Normal families are obtained from subset closed families by removing 'superfluous' maximal pools. Examples 2.4 and 2.5 first demonstrate how superfluous pools may be detected. The ideas developed in these examples are then used to construct Algorithm 2-1 which will never output any superfluous pools. Finally, at the end of the section we define normal families and prove that for any family there exists a normal family that produces the same partial information classes.

To fix notations, in this section $\mathcal{F}$ will denote a subset closed n-family and L a language with $L \in \boldsymbol{C}[\mathcal{F}]$ via some $f \in \boldsymbol{FC}$.

Given n words $w_1, \dots, w_n$, the function f yields a pool $P \in \mathcal{F}$ for these words, i. e., $\chi_L(w_1, \dots, w_n) \in P$. Often it is possible to compute another pool Q for the words $w_1, \dots, w_n$ using an algorithm different from the one used by f. The important point is, that if P is a pool for the words and Q also, then so is $P \cap Q$ which is typically a smaller pool and hence contains more information. In general, in order to obtain a pool as small as possible for some words, we will produce a large number $P_1, \dots, P_k$ of pools for these words using a number of different algorithms and then intersect all these pools to obtain a—hopefully optimal—pool $\bigcap_i P_i$. This will sometimes by referred to as the *intersection trick* in the following.

2.4 Example (Producing new pools by permuting the input)
Given some words $w_1, \dots, w_n$ and a function f we might try to compute a pool R for a *permutation* of the words. Such a pool for the permuted words induces a pool Q for the original words, obtained by *inverting* the permutation on the bitstrings in R.

For example, if we have two words w_1 and w_2 with $\chi_L(w_1, w_2) = 01$ then $P = \{01, 11\}$ might be the original pool generated by f. Then f applied to the word tuple $\langle w_2, w_1 \rangle$ might produce the pool $R = \{00, 10\}$. The pool Q would be given by $\{00, 01\}$; and we would hence *know* that the correct characteristic string is 01 as $P \cap Q = \{01\}$. Naturally, f might also have produced $\{00, 10, 11\}$ as a pool for the words w_2, w_1, which would not have helped at all. ▯

2.5 Example (Producing new pools by adding known words to the input)
A second way to produce a pool Q is to choose two fixed 'known' words $k_0 \notin L$ and $k_1 \in L$, which is always possible for non-trivial languages. Then one can apply f to the tuple $\langle k_0, w_2, \dots, w_n \rangle$. This, too, yields a pool R where we can ignore all bitstrings which do not begin with 0. For all bitstrings which *do* begin with 0, we get possible values for the characteristic string of the words $w_2, \dots, w_n$. Thus, the pool $Q := \{\, 0b \mid 0b \in R \,\} \cup \{\, 1b \mid 0b \in R \,\}$ is a pool for the words $w_1, \dots, w_n$.

For example, let w_1, w_2 and w_3 be words with $\chi_L(w_1, w_2, w_3) = 100$. Applying f to the tuple $\langle k_0, w_2, w_3 \rangle$ might produce the pool $R = \{000, 010, 100, 101\}$. Then, knowing $\chi_L(k_0) = 0$, we know that the last two elements of R cannot be the correct bitstrings. Hence $\chi_L(w_2, w_3) \in \{00, 10\}$ and hence $\chi_L(w_1, w_2, w_3) \in \{000, 100, 010, 110\} = Q$. Again, whether $P \cap Q$ is actually smaller than P depends on $\mathcal{F}$ and f. □

Algorithm 2-1

Generation of a small pool for given words. The two sets K_0 and K_1 are used to denote the positions in the tuple $(x_1, \dots, x_n)$ which we set to k_0 and k_1 respectively. Then, the pool R is calculated using f. The pool Q is constructed from R by taking the 'inverse image' with respect to $i_1, \dots, i_n$ and the sets K_0 and K_1.

$$
\begin{array}{l}
\textbf{\textit{input}}\ w_1, \dots, w_n\,; \\
P := \mathcal{P}(\mathbb{B}^n)\,; \\
\textbf{\textit{forall}}\ i_1, \dots, i_n \in \mathbb{N}_n^* \ \textbf{\textit{do}} \\
\quad \textbf{\textit{forall}}\ K_0, K_1 \subseteq \mathbb{N}_n^* \ \textit{with}\ K_0 \cap K_1 = \emptyset \ \textbf{\textit{do}} \\
\qquad (x_1, \dots, x_n) := (w_{i_1}, \dots, w_{i_n})\,; \\
\qquad \textbf{\textit{forall}}\ i \in K_0 \ \textbf{\textit{do}} \\
\qquad\quad x_i := k_0\,; \\
\qquad \textbf{\textit{forall}}\ i \in K_1 \ \textbf{\textit{do}} \\
\qquad\quad x_i := k_1\,; \\
\qquad R := f(x_1, \dots, x_n)\,; \\
\qquad Q := \big\{ b \in \mathbb{B}^n \ \big| \ \exists b' \in R \colon \big(b\,[i_1, \dots, i_n] = b', \\
\qquad\qquad\qquad\qquad\qquad\quad \forall i \in K_0 \colon b'\,[i] = 0, \\
\qquad\qquad\qquad\qquad\qquad\quad \forall i \in K_1 \colon b'\,[i] = 1 \big) \big\}\,; \\
\qquad P := P \cap Q\,; \\
\textbf{\textit{output}}\ P\,;
\end{array}
$$

The pool produced by Algorithm 2-1 might be smaller than the pool generated by simply applying the function f to the words $w_1, \dots, w_n$. As a matter of fact, consider a maximal pool S such that for some permutation σ we have $S\,[\sigma(1), \dots, \sigma(n)] \notin \mathcal{F}$. In this case, the pool output by the algorithm will *never* be S, because applying the inverse permutation to the bitstrings of the pool for the tuple $\langle w_{\sigma(1)}, \dots, w_{\sigma(n)} \rangle$ will not yield a superpool of S, and hence the intersection P of all the pools Q produced in the algorithm is a proper subpool of S.

As $\boldsymbol{FC}$ is c.c.c. and as $f \in \boldsymbol{FC}$, there exists a function $g \in \boldsymbol{FC}$ which implements Algorithm 2-1. This function will witness $L \in \boldsymbol{C}[\mathcal{F}]$, but it will never output a pool P for which we have $P\,[i_1, \dots, i_n] \notin \mathcal{F}$ for some indices $i_1, \dots, i_n \in \mathbb{N}_n^*$ or $\pi(P) \notin \mathcal{F}$ for some projection $\pi \colon \mathbb{B}^n \to \mathbb{B}^n$, i.e., for some composition $\pi = \pi_{i_1}^{b_1} \circ \dots \circ \pi_{i_k}^{b_k}$ of elementary projections. These observations motivate the following definition and prove the *en suite* theorem due to Nickelsen (1997).

2.6 Definition (Normal Family)

An n-family $\mathcal{F}$ is *closed under selections*, if $P \in \mathcal{F}$ implies $P\,[i_1, \dots, i_n] \in \mathcal{F}$ for all indices $i_1, \dots, i_n \in \mathbb{N}_n^*$. It is *closed under projections*, if $P \in \mathcal{F}$ implies $\pi(P) \in \mathcal{F}$ for all projections $\pi \colon \mathbb{B}^n \to \mathbb{B}^n$. A family is *in normal form*, or just *normal*, if it is subset closed, closed unter selections and closed under projections.

2.7 Theorem
Let $\boldsymbol{FC}$ be c.c.c. and let $\mathcal{F}$ be a subset closed n-family. Let $\mathcal{G}$ denote the largest normal n-family contained in $\mathcal{F}$. Then $\boldsymbol{C}[\mathcal{F}] = \boldsymbol{C}[\mathcal{G}]$.

For the class $\boldsymbol{C}_{\text{dist}}[\mathcal{F}]$ the claim of the theorem is too strong: The problem is that we cannot copy a word in our reduction process as we may not ask the same word more than once. Nickelsen introduced the term *weakly normal* for a family which is closed under projections and *permutations*. Then, the theorem is also correct in the following form for $\boldsymbol{C}_{\text{dist}}[\mathcal{F}]$:

2.8 Theorem
Let $\boldsymbol{FC}$ be c.c.c. and let $\mathcal{F}$ be a subset closed n-family. Let $\mathcal{G}$ denote the largest weakly normal n-family contained in $\mathcal{F}$. Then $\boldsymbol{C}_{\text{dist}}[\mathcal{F}] = \boldsymbol{C}_{\text{dist}}[\mathcal{G}]$.

2.3 Definition of Recursively Presentable Function Classes

This section reviews the widely studied notion of *recursively presentable function classes*. In the next section we will show that *partial information classes over recursively presentable c.c.c. function classes have unique normal representations*.

We will not use recursively presentable function classes directly in the main proof of the next section. Rather, we will show that the existence of *universal functions with resource help*, introduced in Definition 2.11 below, is a sufficient condition for partial information classes to have unique normal representations. Lemma 2.12 below states that every recursively presentable c.c.c. function class contains such a universal function.

2.9 Definition (Recursively Presentable Function Class)
A function class $\boldsymbol{FC}$ is *recursively presentable*, if there exists an effective enumeration $M^0, M^1, M^2, \ldots$ of Turing machines which halt on all inputs, such that $\boldsymbol{FC} = \{ f^i \mid i \in \mathbb{N} \}$, where f^i is the function computed by M^i.

This definition is taken from Landweber and Robertson (1972), but with the addition that the machines must halt on all inputs, which restricts the function classes to recursive functions. This addition is also used in Balcázar *et al.* (1988). An enumeration is *effective*, if the mapping of indices i to the Turing machines M^i is recursive—more formally, if the mapping taking the binary representation of i to a standard coding of M^i is recursive.

2.10 Example (Polynomial time is recursively presentable)
The standard enumeration of the class $\mathbf{FP}$ is given by mapping i to a simulator of the i-th Turing machine. However, this simulator does not only stop when the simulated Turing machine stops, but also after $n^i + i$ steps, where n is the length of the input. First, this definition ensures that only functions computable in polynomial time will be listed. Furthermore, if we have any Turing machine M which is guaranteed to stop after $p(n)$ steps on inputs of length n for some polynomial p, then there exists an infinite number of indices i such that the function computed by the i-th Turing machine is the same as the

function computed by M. Hence, for some sufficiently large such i we have $p(n) \leq n^i + i$ for all n and the simulator will compute exactly the function computed by M. ∎

A prominent example of a function class which is *not* recursively presentable are the recursive functions, as a simple diagonalisation argument shows.

Recursively presentable function classes are commonly used in diagonalisation arguments, most noticeably in the Uniform Diagonalisation Theorem due to Schöning (1982). In the next section, we will also use a diagonalisation argument to prove the uniqueness of normal representations.

However, recursively presentable function classes will not be used directly in the diagonalisation. Rather, for the purposes of this chapter a new intermediate notion of *universal functions with resource help* is introduced below, which also allow diagonalisation arguments and which appear to be better suited for the proofs in the next section.

Universal *partial* functions are an important concept from recursion theory, see for example Odifreddi (1989). A function is *universal* for a set of functions, if it can somehow 'simulate' all other functions of the function class. More precisely, in recursion theory a partial function u is said to be universal for a set $\{ f^i \mid i \in \mathbb{N} \}$ of partial functions, if $u(i,x) = f^i(x)$ for all $i \in \mathbb{N}$ and all x.

Function classes like polynomial time cannot contain functions which are universal for polynomial time. In essence, there cannot exist a function computable in polynomial time which can simulate *all* other functions computable in polynomial time. *However*, there exist universal functions which get as input the number of the function, a word and some kind of *resource help*. Definition 2.11 below makes this precise.

2.11 Definition (Universal Function with Resource Help)
A function $u\colon \Sigma^* \to \Sigma^*$ is *universal with resource help* $r\colon \Sigma^* \to \Sigma^*$ for a function class $\{ f^i\colon \Sigma^* \to \Sigma^* \mid i \in \mathbb{N} \}$, if

1 the graph of r is in $\mathbf{L}$,

2 we have $2^{|w|} \leq |r(w)|$ for all words w,

3 for each $i \in \mathbb{N}$, we have $u\bigl(\langle 0^i, w, r(w)\rangle\bigr) = f^i(w)$ for almost all words w.

The first two properties ensure that the resource help is 'well-behaved'. The first condition ensures that we can 'invert' the resource help effectively. The second condition ensures that once the resource help has been inverted, the resulting words are so short that we can do complicated operations in space logarithmic in the length of the original input.

Instead of $u\bigl(\langle 0^i, w, r(w)\rangle\bigr)$, the more liberal notation $u(0^i, w, r(w))$ will also be used in the following.

Recursively presentable function classes and universal functions are closely linked, as the following lemma shows.

2.12 Lemma

For every recursively presentable function class $\boldsymbol{FC}$ there exists a function $u \in \mathbf{FL}$ and a function $r\colon \Sigma^* \to \Sigma^*$ such that u is universal for $\boldsymbol{FC}$ with resource help r.

Proof. Let $M^0, M^1, M^2, \ldots$ be an effective enumeration of Turing machines which halt on all inputs, such that $\boldsymbol{FC} = \{ f^i \mid i \in \mathbb{N} \}$ where f^i is the function computed by the Turing machine M^i.

First, there exists *some* universal function u for $\boldsymbol{FC}$, not necessarily computable in logarithmic space, even without any resource help. We simply pick the function computed by a Turing simulator S which on input $\langle 0^i, w\rangle$ computes the value $f^i(w)$ by simulating M^i on input w, which is no problem since the enumeration of the Turing machines is effective and all of them stop on all inputs.

The time and space taken by the Turing simulator is not limited *a priori* in any way. Fortunately, we did not refer to the resource help as yet; defining $r(w)$ large enough, we can ensure that the simulation needs very little resources in terms of its input.

We replace the universal Turing simulator by a new simulator which on input $\langle 0^i, w, r(w)\rangle$ does the same as S on input $\langle 0^i, w\rangle$ but stops whenever it uses more space than the logarithm of $|r(w)|$. Obviously, the function u computed by this simulator is in $\mathbf{FL}$. All we need to ensure *is that for each i for almost all words w the simulator S needs less space than the logarithm of $|r(w)|$*. And furthermore, we must ensure that the graph of r is also decidable in logarithmic space. If we can ensure this, the function u is universal for $\boldsymbol{FC}$ in the sense of Definition 2.11.

We define r as follows: It is the function computed by a Turing machine R which maps words w to 2^s many 0's, where s is the *maximum space used by the simulations of the machines M^j with $j = 0, \ldots, |w|$ on input w*. This definition ensures that, indeed, for each i for almost all words w—namely for all words of length greater than i—the new simulator yields correct results.

The graph of this resource help r is not necessarily in $\mathbf{L}$. However, it is not too difficult to see that every recursive function is *dominated* by some function whose graph is in $\mathbf{L}$. More precisely, there exists a function $r'\colon \Sigma^* \to \Sigma^*$ such that for all words w we have $|r(w)| \leq |r'(w)|$ and the graph of r' is in $\mathbf{L}$. To see this, first note that the graph of the function $w \mapsto 1^{t(w)}$ is in $\mathbf{P}$, where $t(w)$ is the number of steps needed by R on input w. Second, note that we can then take as r' the function which maps a word w to $2^{t(w)}$ many 1's. The graph of r' is in $\mathbf{L}$, because in order to check whether $w' = r(w)$ for some pair (w, w'), we must simply take the logarithm of the length of w' and check if this logarithm is exactly the number of steps taken by R on input w.

Note, that we cannot reapply this argument to obtain an r'' whose graph is in an ever smaller class like logarithmic logarithmic space, as even counting the length of the input takes logarithmic space. □

2.4 Proof of the Unique Normal Form Theorem

This section presents the proof of the Unique Normal Form Theorem, Corollary 2.15 below. The following theorem was first stated as Theorem 8 in Nickelsen (1997) for the case $\boldsymbol{FC} = \mathbf{FP}$.

2.13 Theorem
Let $\boldsymbol{FC} = \{ f^k \mid k \in \mathbb{N} \}$ be c.c.c. and let $u \in \boldsymbol{FC}$ be universal for $\boldsymbol{FC}$ with resource help r. Then for every normal n-family $\mathcal{F}$ there exists a language $L \in \boldsymbol{C}[\mathcal{F}]$ which is not an element of $\boldsymbol{C}[\mathcal{G}]$ for all subset closed n-families $\mathcal{G}$ with $\mathcal{F} \not\subseteq \mathcal{G}$.

Proof. Let $\mathfrak{G}$ denote the set of all subset closed n-families $\mathcal{G}$ with $\mathcal{F} \not\subseteq \mathcal{G}$. Note, that $\mathfrak{G}$ is finite. For each $\mathcal{G} \in \mathfrak{G}$ let $P_{\mathcal{G}}$ denote a pool in $\mathcal{F} \setminus \mathcal{G}$.

The proof proceeds in several steps. First, we construct L. Second, for all $\mathcal{G} \in \mathfrak{G}$ we show $L \notin \boldsymbol{C}[\mathcal{G}]$, which will follow easily from the construction as it diagonalises against all functions which could possibly witness $L \in \boldsymbol{C}[\mathcal{G}]$. Third, we show that indeed $L \in \boldsymbol{C}[\mathcal{F}]$.

Let $\beta\colon \mathbb{N} \to \mathbb{N} \times \mathfrak{G}$ be a surjection computable in logarithmic space such for each pair $(k, \mathcal{G}) \in \mathbb{N} \times \mathfrak{G}$ there are an infinite number of $i \in \mathbb{N}$ which are mapped to it.

Construction of the language *L*

The language L is constructed stepwise by a diagonalisation argument. In step i, a subset of the words $W^i := \{w_1^i, \ldots, w_n^i\}$ is added to L. No words other than words from these subsets will be added to the language. For a fixed step i, the words w_ℓ^i are all very much alike—they are all equal to some word w^i tagged with the number ℓ, i. e., we define $w_\ell^i := \langle w^i, 0^\ell \rangle$. If we let $t^i := \langle w_1^i, \ldots, w_n^i \rangle$ denote the tuple consisting of the words in W^i, the words w^i are defined inductively by $w^0 := \lambda$ and $w^{i+1} := \langle t^i, r(t^i) \rangle$. This definition will ensure that later on if we have words at step $i + 1$ as input we always *have enough resources to do simulations at step i and below.*

We must fool each function f^k which could possibly witness $L \in \boldsymbol{C}[\mathcal{G}]$. We will not fool the function f^k directly in step k. Rather, we fool it indirectly by fooling the universal function u. As the universal function u yields the same results as f^k on input 0^k *almost everywhere*, we can fool f^k by fooling u on *an infinite number of inputs.*

Given a step number i, compute $\beta(i) := (k, \mathcal{G})$. The definition of β ensures that we return to each pair $(k, \mathcal{G})$ infinitively often. We now try to ensure that L is not in $\boldsymbol{C}[\mathcal{G}]$ via the function f^k. Let Q denoted the pool coded by $u(0^k, t^i, r(t^i))$ or the empty set if no pool is coded by it. If $Q \in \mathcal{G}$ there exists a bitstring $b \in P_{\mathcal{G}} \setminus Q$ and we *choose the characteristic string of the words $w_1^i, \ldots, w_n^i$ with respect to the language L to be b.* If $Q \notin \mathcal{G}$ we choose the characteristic string arbitrarily, but still in $P_{\mathcal{G}}$. This concludes the construction of the language L.

The constructed language is not in any *C* [*G*]

We claim $L \notin \boldsymbol{C}[\mathcal{G}]$ for any $\mathcal{G} \in \mathfrak{G}$. To see this, assume that we did indeed have $L \in \boldsymbol{C}[\mathcal{G}]$ via some function $f^k \in \boldsymbol{FC}$. As $u(0^k, w, r(w)) = f^k(w)$ for almost all words w, among the infinitely many indices i with $\beta(i) = (k, \mathcal{G})$ there must be one with $u(0^k, t^i, r(t^i)) = f^k(t^i)$. Recall that t^i was just a shorthand for $\langle w_1^i, \dots, w_n^i \rangle$. By construction of the language L, *the pool* $f^k(w_1^i, \dots w_n^i)$ *is not a pool for the words* $w_1^i, \dots, w_n^i$ *which would have to be the case, if* f^k *did indeed witness* $L \in \boldsymbol{C}[\mathcal{G}]$.

The constructed language is in *C* [*F*]

To conclude the proof we must show $L \in \boldsymbol{C}[\mathcal{F}]$. We must find a function $g \in \boldsymbol{FC}$ that, given n words $v_1, \dots, v_n$, yields a pool from $\mathcal{F}$ for them.

The first step is to compute *in which* W^{i_j}*, if any, the words* v_j lie. Fortunately, this problem can be solved in logarithmic space. Starting with a word v_j we first check if it is of the correct form $\langle x, 0^\ell \rangle$ where 0^ℓ is a tag with $1 \le \ell \le n$—recall that all words in the language are of that form. If so and if $x = \lambda$, we stop. Otherwise, we check if x is of the form $\langle y, z \rangle$. If so, we check if $\langle y, z \rangle$ is in the graph of r, i. e., we check if $z = r(y)$. By the first property from Definition 2.11 of universal functions, this check can be done in logarithmic space. If we do indeed have $x = \langle y, r(y) \rangle$, we check if y is of the form $\langle x_1, \dots, x_n \rangle$. If so, we check if all x_ℓ are of the form $\langle x', 0^\ell \rangle$. If so, we restart the whole process with x' instead of x. *As* $|x'| \le \log |x|$ *by the second property from Definition 2.11, all following iterations can easily be done in logarithmic space.* If we reach the word λ, the number of iterations we made is the correct value for i_j. If we do not reach the word λ, we have $v_j \notin W^i$ for all $i \in \mathbb{N}$ and hence $v_j \notin L$.

Now, having obtained the indices i_j, we can check if we happen to be asked the tuple $w_1^i, \dots, w_n^i$ for some index i. In this case *we can easily produce a pool from* $\mathcal{F}$: The pool $P_\mathcal{G}$ is correct because we carefully setup L in such a way that $P_\mathcal{G}$ is correct in this situation.

Next, consider the more complicated case of a permutation $w_{\sigma(1)}^i, \dots, w_{\sigma(n)}^i$. As $\mathcal{F}$ is normal, we can output $P[\sigma(1), \dots, \sigma(n)] \in \mathcal{F}$. Likewise, if we are given a selection of these words we can still easily produce the desired pool using the normal form properties.

The tricky part comes, when we are not given a subset of words from the same level, but arbitrary words. Let i^* denote the largest index such that there exists a j with $v_j \in W^{i^*}$. If no such index exists, none of the words is in L and we can trivially produce a pool for them.

We will show that there exists a function in $\boldsymbol{FC}$ which maps v_j to $\chi_L(v_j)$ for all v_j with $v_j \notin W^{i^*}$. If this claim holds, we can produce a pool for $v_1, \dots, v_n$ by first taking an appropriate selection of the pool $P_\mathcal{G}$ for the words in W^{i^*} and then projecting the positions of the other words to the known values. As $\mathcal{F}$ is normal, this still produces a pool from $\mathcal{F}$.

Deciding words from lower construction steps

We must now compute $\chi_L(v_j)$ for $v_j \notin W^{i^*}$. If v_j is not element of some other W^i we know $\chi_L(v_j) = 0$ and we are done. Otherwise, assume $v_j \in W^i$ for some $i < i^*$—remember that i^* was chosen to be maximal.

As a byproduct of the calculation of the index i^*, we will also have obtained the values t^{i_j} and more importantly $\langle t^{i_j}, r(t^{i_j})\rangle$. *Note that we do not have to compute $r(t^{i_j})$ from a known t^{i_j}—which would typically not be possible quickly—but we 'come across' the value $\langle t^{i_j}, r(t^{i_j})\rangle$ during the iteration process.*

But now, we can compute $u(0^k, t^{i_j}, r(t^{i_j}))$ and knowing this value we can easily compute the correct value for $\chi_L(v_j)$. As we need to invoke the function u at most n times and as all other calculations can be carried out in logarithmic space we get $g \in \boldsymbol{FC}$. □

2.14 Corollary
Let $\boldsymbol{FC}$ be c.c.c. and recursively presentable and let $\mathcal{F}$ and $\mathcal{G}$ be normal n-families. Then $\boldsymbol{C}[\mathcal{F}] \subseteq \boldsymbol{C}[\mathcal{G}]$, iff $\mathcal{F} \subseteq \mathcal{G}$.

Proof. If $\mathcal{F} \subseteq \mathcal{G}$ then we trivially have $\boldsymbol{C}[\mathcal{F}] \subseteq \boldsymbol{C}[\mathcal{G}]$. Otherwise, let $\mathcal{F} \not\subseteq \mathcal{G}$. As $\boldsymbol{FC}$ is recursively presentable, Lemma 2.12 tells us that there exists a universal function $u \in \mathbf{FL}$ for the function class $\boldsymbol{FC}$. As $\mathbf{FL} \subseteq \boldsymbol{FC}$ the universal function is included in $\boldsymbol{FC}$. Hence, by Theorem 2.13 we have $\boldsymbol{C}[\mathcal{F}] \not\subseteq \boldsymbol{C}[\mathcal{G}]$. □

2.15 Corollary (Unique Normal Form Theorem)
Let $\boldsymbol{FC}$ be c.c.c. and recursively presentable. Then for every n-family there exists exactly one normal n-family which produces the same partial information class over $\boldsymbol{FC}$.

Bibliographical Notes

The first steps towards a *description theory* of partial information were taken in Beigel *et al.* (1995a). The authors used sets of pools as *descriptions*, but not as *representations*, of notions of partial information over the class of recursive functions. Families had no combinatorial life of their own. This was remedied in Nickelsen (1997) where normal forms for families were introduced. Both there as well as in his PhD thesis, Nickelsen proved the Unique Normal Form Theorem only for polynomial time, but pointed out that it also holds for other kinds of resource bounded complexity classes.

Subset closed families, i. e., families which contain with any pool also all subpools, were first used in Tantau *et al.* (1998).

A general machine independent way to identify resource bounds has been proposed by Blum (1967) who introduced a general definition of *complexity measures*. The close, but sometimes involved, relationship between complexity measures and recursively presentable function classes is discussed in Landweber and Robertson (1972). A most useful application of recursively presentable classes is the *Uniform Diagonalisation Theorem* due to Schöning (1982). Unfortunately, this theorem cannot be used to prove the Unique Normal Form Theorem as it relies heavily on the considered classes' being recursive—which is normally not the case for partial information classes.

3

Structure of Normal Families

T. Brown in "A collection of 250 papers on Woffle Theory
dedicated to R. S. Green on his 23rd Birthday"
defined a Piffle to be an infinite multi-variable sub-polynormal Woffle
which does not satisfy the lower regular Q-property.

He stated, but was unable to prove,
that there were at least a finite number of Piffles.

— A. K. Austin, *The Mathematical Gazette*, 51:149–150, 1967

Normal families form the backbone of the analysis of partial information. In the previous chapter, we showed that for all c.c.c. function classes and any n-family $\mathcal{F}$, there exists a normal n-family $\mathcal{G}$ producing the same partial information class. If the function class is furthermore recursively presentable, the normal family $\mathcal{G}$ is even uniquely determined.

This chapter studies the combinatorics of normal families. The first two sections investigate the 'inner structure' of normal families. Borrowing from linear algebra, we introduce generating systems and bases of families. We show that bases of normal families are almost uniquely determined. In the third section, we prove that every base of a normal family is a representative system of an antichain in the so-called unit poset. Bases and antichains provide a new efficient way of describing families which has not been studied before.

The fourth section studies *upward translations* of families. For an m-family $\mathcal{E}$ we can always find an n-family $\mathcal{F}$ for $n \geq m$ which produces the same partial information class. This was first proved in Nickelsen (1997).

Upward translation is a purely combinatorial process, which enables us to *compare partial information classes for differing input numbers combinatorially*. The fifth section gives numerous examples of upward translations of special families like the selective families—thereby giving combinatorial proofs for a number of results due to Selman (1979) and Beigel (1987).

In the sixth section we put all results of this chapter together to prove that *inclusion is well-founded on cheatable partial information classes*. This makes progress on a remark of Nickelsen (1999) who conjectured that inclusion is well-founded on partial inclusion classes.

This chapter concludes with a figure section where all normal 2-families are depicted as well as the unit poset for $n = 3$. Furthermore, the number of units in all unit posets for $n \le 5$ are listed.

3.1 Definition of Generating Systems

It is often awkward to write down a subset closed family. Its like writing down all elements of a finite group, where writing down its generating elements would be sufficient. Nickelsen (1999) proposed the following useful definition:

3.1 Definition (Generating System)
Let $\mathcal{G}$ be a set of n-pools. We define $\langle \mathcal{G} \rangle$ as the smallest normal n-family that contains $\mathcal{G}$. We call $\mathcal{G}$ a *generating system for* $\langle \mathcal{G} \rangle$.

Note, that there does, indeed, always exist a smallest normal n-family as the intersection of two normal families is still normal.

Generating systems can also be defined constructively. Consider a generating system $\mathcal{G}$ for a family $\mathcal{F}$. Then $\mathcal{F}$ contains exactly those pools which can be generated from some pool in $\langle \mathcal{G} \rangle$ by applying selections or projections and reducing the size of the pool. For example, the pool $\{1000, 1110\}$ can be generated from the pool $\{0000, 0001\}$ by swapping the positions 3 and 4, copying the resulting position 3 to position 2 and the projecting the first position to 1. Hence we have $\{1000, 1110\} \in \langle \{0000, 0001\} \rangle$.

3.2 Notation (Stacking Notation)
Swapping of positions and copying can be done more easily, if we stack the different bitstrings instead of listing them alongside each other. In this notation $\{1000, 1110\}$ becomes $\{{1000 \atop 1110}\}$ and $\{0000, 0001\}$ becomes $\{{0000 \atop 0001}\}$. Using this stacking notation, it makes sense to talk about the *i-th column* of a pool.

3.3 Example (Generating systems of the minimal family)
The minimal non-trivial normal 2-family

$$\begin{aligned}\textsc{min}_2 = \big\{&\emptyset, \{00\}, \{01\}, \{10\}, \{11\},\\ &\{00, 01\}, \{00, 10\}, \{10, 11\}, \{01, 11\}, \{00, 11\}\big\}\end{aligned}$$

has the generating system $\big\{\{00, 01\}\big\}$. Another such system is $\big\{\{10, 11\}, \{00\}\big\}$, whereas $\big\{\{00\}\big\}$ is no generating system. ▯

3.4 Definition (Eigenpool)
Let P be a pool. The *eigenpool* of P is obtained by removing from P all constant columns and for every duplicate column all but the first occurrence of the column.

If two pools have the same eigenpool, they can be generated from each other by permuting and copying columns as well as projecting some columns. *Hence, solely the eigenpools of the pools of a generating system determine the generated family.*

3.5 Example (Eigenpools)
The pool $\{100, 011\}$ has the eigenpool $\{10, 01\}$. The pools $\left\{\begin{smallmatrix}1000\\1110\end{smallmatrix}\right\}$ and $\left\{\begin{smallmatrix}0000\\0001\end{smallmatrix}\right\}$ have the same eigenpool, namely the pool $\{0, 1\}$. □

3.2 Definition of Bases

Having defined a generating system, it is natural enough to define a basis. Surprisingly, a basis of a family is nearly uniquely determined.

3.6 Definition (Basis)
A set $\mathcal{B}$ of n-pools is a *basis* of a normal family $\mathcal{F}$, if $\mathcal{B}$ is a minimal generating system of $\mathcal{F}$, i. e., $\langle\mathcal{B}\rangle = \mathcal{F}$ and $\langle\mathcal{B}'\rangle \neq \mathcal{F}$ for all proper subsets $\mathcal{B}' \subsetneq \mathcal{B}$.

3.7 Theorem
Let $\mathcal{F}$ be a normal family and $\mathcal{B}_1$ a basis of $\mathcal{F}$. Then $\mathcal{B}_2$ is also a basis of $\mathcal{F}$, iff there exists a bijection $\beta\colon \mathcal{B}_1 \to \mathcal{B}_2$ such that for all $P \in \mathcal{B}_1$ the pools P and $\beta(P)$ have the same eigenpools up to permutation.

Proof. If such a bijection exists, from our definition of eigenpools we easily deduce $\langle\mathcal{B}_1\rangle = \langle\mathcal{B}_2\rangle$.

For the other direction, let $\mathcal{B}_2$ be a basis of $\mathcal{F}$. As $\mathcal{B}_1$ is a generating system of $\mathcal{F}$ and $\mathcal{B}_2 \subseteq \mathcal{F}$, all pools in $\mathcal{B}_2$ can be generated from pools in $\mathcal{B}_1$. This yields a function $\beta_2\colon \mathcal{B}_2 \to \mathcal{B}_1$ which assigns each pool $P_2 \in \mathcal{B}_2$ a pool in $P_1 \in \mathcal{B}_1$, such that P_2 can be generated from P_1 by selection and projection. Likewise we get a function $\beta_1\colon \mathcal{B}_1 \to \mathcal{B}_2$ with analogous properties.

Consider the function $\beta_2 \circ \beta_1\colon \mathcal{B}_1 \to \mathcal{B}_1$. For a pool $P \in \mathcal{B}_1$ we get a pool $\beta_1(P) \in \mathcal{B}_2$ from which we can generate P. From $\beta_1(P)$ we get a pool $\beta_2\bigl(\beta_1(P)\bigr) \in \mathcal{B}_1$ from which we can generate $\beta_1(P)$. Thus, we can generate P from $\beta_2\bigl(\beta_1(P)\bigr)$. However, $\mathcal{B}_1$ is a basis. In this case, P can only be generated from another pool $Q \in \mathcal{B}_1$, if $P = Q$. This implies $P = \beta_2\bigl(\beta_1(P)\bigr)$ and hence $\beta_2 \circ \beta_1 = \mathrm{id}$. Obviously, we also get $\beta_1 \circ \beta_2 = \mathrm{id}$. Hence, $\beta := \beta_1$ is bijective.

The pools P and $\beta(P)$ have the same eigenpool up to permutation, as P can be generated from $\beta(P)$ which in turn can be generated from P. □

3.8 Example (Bases of special families)

1 The family $\textsc{sel}_2$ has the two bases $\left\{\left\{\begin{smallmatrix}00\\01\\11\end{smallmatrix}\right\}\right\}$ and $\left\{\left\{\begin{smallmatrix}00\\10\\11\end{smallmatrix}\right\}\right\}$.

2 The family $\textsc{min}_n$ has—among many others—the basis $\left\{\left\{\begin{smallmatrix}0\ldots00\\0\ldots01\end{smallmatrix}\right\}\right\}$. Here, the eigenpool is given by the last column.

3 The family $\textsc{cheat}_2$ has the basis $\left\{\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}\right\}$.

4 The family $\textsc{approx}_2$ has two bases, namely $\left\{\left\{\begin{smallmatrix}00\\01\\11\end{smallmatrix}\right\}, \left\{\begin{smallmatrix}00\\01\\10\end{smallmatrix}\right\}, \left\{\begin{smallmatrix}01\\10\\11\end{smallmatrix}\right\}\right\}$ as well as $\left\{\left\{\begin{smallmatrix}00\\10\\11\end{smallmatrix}\right\}, \left\{\begin{smallmatrix}00\\01\\10\end{smallmatrix}\right\}, \left\{\begin{smallmatrix}01\\10\\11\end{smallmatrix}\right\}\right\}$. □

3.3 Representing Normal Families by Antichains

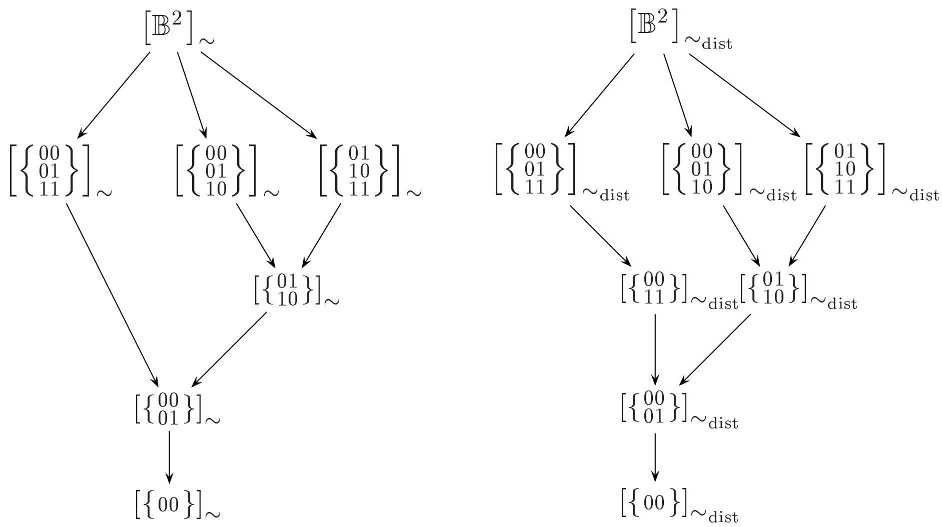
In this section, Definition 3.9 below introduces the *unit poset*. A unit is a set of pools which all have the same eigenpool up to permutation. A partial ordering is induced on such units by inclusion of the families they generate.

The name *units* is once more borrowed from linear algebra, namely from unit vectors. The motivation is that with each basis a unique set of units can be associated. More precisely, Theorem 3.10 below states that there exists a bijection between normal families and antichains in the unit posets. Using this theorem, we can easily enumerate all normal 2-families as we only have to enumerate all antichains in a small poset.

By virtue of Theorem 3.7 two pools generate the same family, iff their eigenpools are permutations of each other. The other way round, any family generated by a single pool P is generated exactly by all those pools whose eigenpools are permutations of the eigenpool of P.

3.9 Definition (Unit, Unit Poset)
Two n-pools are *equivalent*, written $P \sim Q$, if $\langle\{P\}\rangle = \langle\{Q\}\rangle$. An n-*unit* is an equivalence class of an n-pool, i. e., an element of the partition $\mathfrak{U}_n := \mathcal{P}(\mathbb{B}^n)/\!\sim$. We define a *partial ordering of n-units* by $\mathcal{U} \preceq \mathcal{V}$, iff $\langle\mathcal{U}\rangle \subseteq \langle\mathcal{V}\rangle$. This yields the *unit posets* $(\mathfrak{U}_n, \preceq)$.

The partial ordering of units can be expressed in terms of representatives. For pools P and Q we have $[P]_\sim \preceq [Q]_\sim$, iff $\big\langle\{P\}\big\rangle \subseteq \big\langle\{Q\}\big\rangle$.



The poset $(\mathfrak{U}_2, \preceq)$ **The poset $(\mathfrak{W}_2, \preceq)$**

Figure 3-1
Unit poset and its weak counterpart for $n = 2$; for $n = 3$ see Figure 3-8 on page 44. The units are shown as equivalence classes of one of their elements. The relation $\sim$ is defined in Definition 3.9. For the weak units, there exists a corresponding relation which we denote $\sim_{\text{dist}}$.

Consider an antichain $\mathfrak{A}$ in $(\mathfrak{U}_n, \preceq)$. Recall that an antichain in a partial ordering is a set, for which no two elements are comparable with respect to the partial ordering. Let $\mathcal{A}$ be a representative system of $\mathfrak{A}$, i. e., for each unit $\mathcal{U} \in \mathfrak{A}$ let there be exactly one pool $P \in \mathcal{A}$ with $P \in \mathcal{U}$. As elements of an antichain are incomparable, this representative system is a basis of $\langle \mathcal{A} \rangle$. The other way round, for any set of units that is no antichain, no representative system $\mathcal{G}$ is a basis, as $\mathcal{G}$ necessarily contains two pools P and Q, such that $[P]_\sim \subseteq [Q]_\sim$ and hence $\langle \mathcal{G} \rangle = \langle \mathcal{G} \backslash \{P\} \rangle$. To sum up, a set $\mathfrak{A}$ of units is an antichain, iff one and then all representative systems of $\mathfrak{A}$ are bases. All such representative systems of an antichain produce the same family.

Consider an arbitrary normal family $\mathcal{F}$. Then every basis $\mathcal{B}$ of $\mathcal{F}$ is a representative system of a set of units, namely of $\{[P]_\sim \mid P \in \mathcal{B}\}$. We summarise these results in the following theorem:

3.10 Theorem (Antichain Theorem)
There exists a bijection β between the normal n-families and the non-empty antichains in $(\mathfrak{U}_n, \preceq)$ such that the bases of a normal n-family $\mathcal{F}$ are exactly the representative systems of $\beta(\mathcal{F})$.

It is straightforward to extend these definitions to weakly normal families. We can define a weak generating system of a weakly normal family $\mathcal{F}$ as a set $\mathcal{G}$ such that $\mathcal{F}$ is the least weakly normal family which contains all of $\mathcal{G}$. Likewise, we define a weak basis of $\mathcal{F}$ and finally the weak unit posets $(\mathfrak{W}_n, \preceq)$.

At the end of this chapter in Section 3.7, all antichains in $(\mathfrak{U}_2, \preceq)$ and $(\mathfrak{W}_2, \preceq)$ are shown—and hence all (weakly) normal families of index 2.

Table 3-2
To the left, the sizes and numbers of antichains in the *unit posets* for $n \leq 5$ are shown, to the right the same is done for the *weak unit posets*. The number of antichains increases swiftly, and starting from $n = 4$ this number is no longer computable using counting algorithms.

The lower bounds for the number of antichains for $n = 4$ are obtained from Table 3-9 on page 45 by noting that for $n = 4$ there are 727 units of size 8. All of these are incomparable with respect to the partial ordering induced on units. Hence, every subset of these units is an antichain—and there are $2^{727} \approx 10^{218}$ such subsets.

Poset	Units	Antichains
$(\mathfrak{U}_1, \preceq)$	2	2
$(\mathfrak{U}_2, \preceq)$	7	12
$(\mathfrak{U}_3, \preceq)$	60	2 323 341
$(\mathfrak{U}_4, \preceq)$	3 777	$> 2^{727}$
$(\mathfrak{U}_5, \preceq)$	37 316 578	$> 2^{5182324}$

Poset	Weak Units	Antichains
$(\mathfrak{W}_1, \preceq)$	2	2
$(\mathfrak{W}_2, \preceq)$	8	17
$(\mathfrak{W}_3, \preceq)$	68	5 098 786
$(\mathfrak{W}_4, \preceq)$	3 904	$> 2^{727}$
$(\mathfrak{W}_5, \preceq)$	37 329 264	$> 2^{5182324}$

3.4 Upward Translation of Normal Families

Given an m-family $\mathcal{E}$ and a new index n with $n > m$, does there exist an n-family $\mathcal{F}$ *which produces the same partial information class as* $\mathcal{E}$? If so, $\mathcal{F}$ can be thought of as the *index n version* or the *upward translation* of the m-family $\mathcal{E}$.

In this section, Lemma 3.11 demonstrates that upward translation is always possible, as was first noted in Nickelsen (1997). While this lemma solves the upward translation problem in principle, a stronger result is needed to ensure that the upward translation of a normal family yields a normal family once more. This result is presented in Theorem 3.12 below. We first proved this theorem in Tantau *et al.* (1998), but the present refined form is due to Nickelsen (1999).

Upward translation is the last missing piece necessary for the representation theory of partial information by normal families. Corollary 3.14 below summarises the results obtained up to now on the relationship between partial information classes and families.

This section concludes with a short look at the problem of *downward translation*. Lemma 3.15 below was first proved in Tantau *et al.* (1998), but a more thorough treatment of this subject can be found in Nickelsen (1999).

3.11 Lemma
Let $\boldsymbol{FC}$ be c.c.c. and $\mathcal{E}$ an m-family. For $n \geq m$ define the n-family $\mathcal{F}$ by

$$\mathcal{F} := \{\, P \subseteq \mathbb{B}^n \mid P\texttt{[}1,\dots,m\texttt{]} \in \mathcal{E} \,\}.$$

Then $\boldsymbol{C}[\mathcal{E}] = \boldsymbol{C}[\mathcal{F}]$.

Proof. Assume $L \in \boldsymbol{C}[\mathcal{E}]$. Let $w_1,\dots,w_n$ be any n words. We construct a function that witnesses $L \in \boldsymbol{C}[\mathcal{F}]$: For the first m words we obtain an m-pool Q such that $\chi_L(w_1,\dots,w_m) \in Q \in \mathcal{E}$. We extend this pool to an n-pool by setting $P := \{\, b \in \mathbb{B}^n \mid b\texttt{[}1,\dots,m\texttt{]} \in Q \,\}$, i. e., we simply extend all bitstrings in Q in all possible ways. Then P is a pool for $w_1,\dots,w_n$ and $P \in \mathcal{F}$ by definition, as $P\texttt{[}1,\dots,m\texttt{]} = Q \in \mathcal{E}$.

For the other direction, assume $L \in \boldsymbol{C}[\mathcal{F}]$. For any m words $w_1,\dots,w_m$ we add arbitrary words $w_{m+1},\dots,w_n$ and compute a pool P for $\chi_L(w_1,\dots,w_n)$. Extracting the first m columns yields $Q := P\texttt{[}1,\dots,m\texttt{]}$ with $Q \in \mathcal{E}$ by definition. But Q is a pool for the first m words. □

The lemma shows that for increasing n the partial information classes produced by n-families get more and more fine-grained. Unfortunately, the construction in Lemma 3.11 is asymmetric and it is easy to see that in general the construction does not yield a normal family, even when $\mathcal{E}$ is normal. Naturally, we could compute the normal form of the family produced by the lemma. Hence, the lemma solves the upward translation problem *in principle*; but it is ill-suited for direct combinatorial arguments. In the following, we remedy this shortcoming and construct a normal n-family directly, which produces the same partial information class as $\mathcal{E}$.

The basic idea is to proceed as in Section 2.2 where we constructed the normal form of a family by producing many pools for some words and then intersecting these pools. Given any n words, we can compute a pool $R_{i_1,\dots,i_m}$ for *any selection of m words at distinct indices i_1 to i_m*. Each such pool can be extended to a pool for all n words, by setting $Q_{i_1,\dots,i_m} := \{\, b \in \mathbb{B}^n \mid b[i_1,\dots,i_m] \in R_{i_1,\dots,i_m} \,\}$, i. e., by allowing the columns not selected to range over all possible values. For example, if we have $m = 2$ and $n = 4$ and select the first and the fourth column and if we have $R_{1,4} = \{00\}$, then $Q_{1,4} = \{0000, 0010, 0100, 0110\}$.

3.12 Theorem (Upward Translation)
Let $\boldsymbol{FC}$ be c.c.c. and let $\mathcal{E}$ be an m-family. For $n \geq m$ let $\mathcal{F}$ denote the n-family of all pools P for which for all distinct indices $i_1,\dots,i_m \in \mathbb{N}_n^*$ we have $P[i_1,\dots,i_m] \in \mathcal{E}$.

1 *If $\mathcal{E}$ is normal, so is $\mathcal{F}$ and $\boldsymbol{C}[\mathcal{E}] = \boldsymbol{C}[\mathcal{F}]$.*

2 *If $\mathcal{E}$ is weakly normal, so is $\mathcal{F}$ and $\boldsymbol{C}_{\mathrm{dist}}[\mathcal{E}] = \boldsymbol{C}_{\mathrm{dist}}[\mathcal{F}]$.*

Proof. We prove only the first point. The second can then be obtained easily as the construction of $\mathcal{F}$ only refers to *distinct* indices.

The family $\mathcal{F}$ is a subfamily of $\{\, P \subseteq \mathbb{B}^n \mid P[1,\dots,m] \in \mathcal{E} \,\}$ by construction. Hence, by Lemma 3.11 we can conclude $\boldsymbol{C}[\mathcal{F}] \subseteq \boldsymbol{C}[\mathcal{E}]$.

For the other direction, assume $L \in \boldsymbol{C}[\mathcal{E}]$ via some function f. Then, there exists some function $g \in \boldsymbol{FC}$ which implements Algorithm 3-3 as $\boldsymbol{FC}$ is c.c.c. The algorithm will always output a pool $P \in \mathcal{F}$ as, indeed, every selection of m distinct columns of P is a pool from $\mathcal{E}$. Furthermore, P is also a pool for the input words. Hence, $L \in \boldsymbol{C}[\mathcal{F}]$.

Algorithm 3-3
An algorithm for the generation of an n-pool for given words.

input $w_1,\dots,w_n$;
$P := \mathcal{P}(\mathbb{B}^m)$;
forall *distinct* $i_1,\dots,i_m \in \mathbb{N}_n^*$ ***do***
  $R := f(w_{i_1},\dots,w_{i_m})$;
  $Q := \{\, b \in \mathbb{B}^n \mid \exists b' \in R\colon b[i_1,\dots,i_m] = b' \,\}$;
  $P := P \cap Q$;
output P;

We still need to show that $\mathcal{F}$ is normal. It is obvious that $\mathcal{F}$ is subset closed. Furthermore, it is also closed under selections, for consider some $P \in \mathcal{F}$ and arbitrary indices $j_1,\dots,j_n \in \mathbb{N}_n^*$. Being closed under selections means that then $P[j_1,\dots,j_n] \in \mathcal{F}$. By definition of the family $\mathcal{F}$ this is the case, if for any distinct indices $i_1,\dots,i_m \in \mathbb{N}_n^*$ we have $P[j_1,\dots,j_n][i_1,\dots,i_m] \in \mathcal{E}$. Phrased differently, for a pool $P \in \mathcal{F}$ we must check if $P[j_{i_1},\dots,j_{i_m}] \in \mathcal{E}$.

If the indices $j_{i_1},\dots,j_{i_m}$ are distinct, we are done by definition. If not, consider new distinct indices $j'_{i_1},\dots,j'_{i_m}$ obtained from the old ones by replacing repetitions of indices by arbitrary fresh indices. As the new indices are distinct we have $P[j'_{i_1}, \dots, j'_{i_m}] \in \mathcal{E}$. Next, we use an appropriate selection of the columns of the pool $P[j'_{i_1}, \dots, j'_{i_m}]$ which *overwrites* all positions where we used a fresh index with the value we actually wanted for that position, hence yielding the pool $P[j_{i_1}, \dots, j_{i_m}]$. As $\mathcal{E}$ is normal, this yields a pool from $\mathcal{E}$ once more. But then $P[j_{i_1}, \dots, j_{i_m}] \in \mathcal{E}$.

A like argument can be used to show that $\mathcal{F}$ is also closed under projections because $\mathcal{E}$ is closed under projections. Thus, $\mathcal{F}$ is normal. □

3.13 Notation (Upward Translation)
We will denote the family $\mathcal{F}$ defined in Theorem 3.12 by $\lceil\mathcal{E}\rceil_n$. It will be called the *upward translation* of $\mathcal{E}$.

The following corollary summarises the consequences of the main combinatorial results obtained up to now on the structure of normal families.

3.14 Corollary (Summary of Results)
Let $\boldsymbol{FC}$ be c.c.c. and recursively presentable and let $\mathcal{F}$ be a family. Let $m \in \mathbb{N}$ be minimal, such that there exists an m-family $\mathcal{E}$ with $\boldsymbol{C}[\mathcal{E}] = \boldsymbol{C}[\mathcal{F}]$. Then, for all $n \geq m$ the following propositions hold:

1 *There exists exactly one normal n-family $\mathcal{F}_n$ with $\boldsymbol{C}[\mathcal{F}] = \boldsymbol{C}[\mathcal{F}_n]$.*

2 *The family $\mathcal{F}_n$ is the intersection of all subset closed n-families $\mathcal{G}$ for which we have $\boldsymbol{C}[\mathcal{F}] \subseteq \boldsymbol{C}[\mathcal{G}]$.*

3 *There exists a unique antichain in $(\mathfrak{U}_n, \preceq)$ such that one and then all representative systems of the antichain are bases of $\mathcal{F}_n$.*

4 *We have $\mathcal{F}_n = \lceil\mathcal{F}_m\rceil_n$.*

Proof. For the first point from Lemma 3.11 we know that for each $n \geq m$ there exists *some* family which produces the same partial information class as $\mathcal{F}$; and by the Unique Normal Form Theorem there exists a unique normal n-family with that property. The remaining points are direct consequences of Theorems 2.7, 3.10 and 3.12, respectively. □

To finish this section, the following lemma tells us how to *reduce* the index of families. This lemma is only correct for the resource bounded case.

3.15 Lemma (Downward Translation)
Let $\boldsymbol{FC}$ be c.c.c. and recursively presentable and let $\mathcal{F}$ be a normal n-family. Then for $m \leq n$ the m-family $\mathcal{E} := \{\, P[1, \dots, m] \mid P \in \mathcal{F} \,\}$ is normal and it is the smallest subset closed m-family whose partial information class contains $\boldsymbol{C}[\mathcal{F}]$.

Proof. Obviously, $\mathcal{E}$ is normal. Furthermore, we have $\boldsymbol{C}[\mathcal{F}] \subseteq \boldsymbol{C}[\mathcal{E}]$, because $\mathcal{F}$ is contained in the family $\{\, P \subseteq \mathbb{B}^n \mid P[1, \dots, m] \in \mathcal{E} \,\}$ and by our first result on upward translation, Lemma 3.11, this family produces the same partial information class as $\mathcal{E}$.

For the minimality, let $\mathcal{G}$ be a normal m-family with $\boldsymbol{C}[\mathcal{F}] \subseteq \mathbf{C}[\mathcal{G}] \subseteq \boldsymbol{C}[\mathcal{E}]$. We prove $\mathcal{G} = \mathcal{E}$. As $\lceil\mathcal{G}\rceil_n$ and $\lceil\mathcal{E}\rceil_n$ produce the same partial information classes as $\mathcal{G}$ and $\mathcal{E}$, by the Unique Normal Form Theorem we conclude

$\mathcal{F} \subseteq \lceil \mathcal{G} \rceil_n \subseteq \lceil \mathcal{E} \rceil_n$. Now, as $\mathcal{E} = \{\, P\texttt{[}1,\ldots,m\texttt{]} \mid P \in \mathcal{F} \,\}$ and likewise $\mathcal{E} = \{\, P\texttt{[}1,\ldots,m\texttt{]} \mid P \in \lceil \mathcal{E} \rceil_n \,\}$ and finally also $\mathcal{G} = \{\, P\texttt{[}1,\ldots,m\texttt{]} \mid P \in \lceil \mathcal{G} \rceil_n \,\}$, we get $\mathcal{E} \subseteq \mathcal{G} \subseteq \mathcal{E}$ and thus $\mathcal{G} = \mathcal{E}$. $\square$

3.5 Upward Translation of Special Families

This section studies five examples which show how the most important families behave upon upward translation.

3.16 Example (Upward translation of the selective families)
A most important upward translation is $\lceil \mathrm{SEL}_2 \rceil_n$. Recall that we defined SEL_n as the set of all chains in $\mathbb{B}^n$. It turns out that SEL_n and $\lceil \mathrm{SEL}_2 \rceil_n$ coincide as the following combinatorial argument shows. A non-combinatorial proof which argues in the language domain is due to Selman (1979).

First, every chain in $\mathbb{B}^n$ is in $\lceil \mathrm{SEL}_2 \rceil_n$. The reason is that, indeed, every selection of two positions from a chain in $\mathbb{B}^n$ is a chain in $\mathbb{B}^2$ as selections are monotone functions.

All pools $P \in \lceil \mathrm{SEL}_2 \rceil_n$ are chains in $\mathbb{B}^n$. To see this, assume $P \notin \mathrm{SEL}_n$. Order the bitstrings of P in such a way that $P = \{b_1, \ldots, b_k\}$ and the number of 1's in b_i increases with i. Now, this is an ordering of P and $P \notin \mathrm{SEL}_n$ and thus there exists an index i and a position j such $b_i\texttt{[}j\texttt{]} = 1$ and $b_{i+1}\texttt{[}j\texttt{]} = 0$. However, as the number of 1's in b_{i+1} is at least equal to the number of 1's in b_i there *exists also some different position* k such that $b_i\texttt{[}k\texttt{]} = 0$ and $b_{i+1}\texttt{[}k\texttt{]} = 1$. Then $P\texttt{[}j,k\texttt{]} \supseteq \{01, 10\} \notin \mathrm{SEL}_2$ and thus $P \notin \lceil \mathrm{SEL}_2 \rceil_n$.

Recalling Notation 1.15 for families without an index, we have just proved $\boldsymbol{C}[\mathrm{SEL}_2] = \boldsymbol{C}[\mathrm{SEL}_n] = \boldsymbol{C}[\mathrm{SEL}]$ for $n \geq 2$. $\square$

3.17 Example (Upward translation of the minimal families)
The smallest non-trivial normal n-family is denoted MIN_n. We show that for $n \geq 2$ we have $\mathrm{MIN}_n = \lceil \mathrm{MIN}_2 \rceil_n$ as pointed out in Nickelsen (1999).

Theorem 3.10 states that every normal family is uniquely identified by a non-empty antichain in the unit poset. Hence, it suffices to prove that bases of MIN_n and $\lceil \mathrm{MIN}_2 \rceil_n$ have the same units. Consider a basis $\mathcal{B}$ of MIN_n. As MIN_n is minimal, this basis consists of a single pool P whose eigenpool is just $\mathbb{B}$. Likewise, consider a basis $\mathcal{B}'$ of $\lceil \mathrm{MIN}_2 \rceil_n$. Let $P' \in \mathcal{B}'$. If P' had an eigenpool with at least two different columns found at positions i and j in P', we would have $P\texttt{[}i,j\texttt{]} \notin \mathrm{MIN}_2$, as every pool in MIN_2 has an eigenpool with at most one column. Thus, every pool in $\mathcal{B}'$ has the eigenpool $\mathbb{B}$. By Theorem 3.10 this proves the claim. $\square$

3.18 Example (Upward translation of bottom and top families)
The bottom family BOTTOM_n is generated by the pool $\mathrm{B}_1(0^n)$. Trivially, a basis of BOTTOM_n is $\{\mathrm{B}_1(0^n)\}$. Using the same techniques as in the preceding examples, for $n \geq 2$ one can prove $\lceil \mathrm{BOTTOM}_2 \rceil_n = \mathrm{BOTTOM}_n$. Likewise, one proves $\lceil \mathrm{TOP}_2 \rceil_n = \mathrm{TOP}_n$. These results are also due to Nickelsen (1999). $\square$

3.19 Example (Upward translation of the cheatable families)

Recall, that we defined $\text{CHEAT}_n = \text{SIZE}_n(n)$. The upward translation of CHEAT_m is *not* CHEAT_n. Instead, what happens is that we get the same size family for the new index: $\lceil \text{CHEAT}_m \rceil_n = \text{SIZE}_n(m)$ for $n \geq m$. From this, we can easily deduce that for $m \leq n \leq k$ one generally has $\lceil \text{SIZE}_n(m) \rceil_k = \text{SIZE}_k(m)$. We will now give a combinatorial proof for this, which is different from the proof of Theorem 5.4 in Beigel (1991), where a slightly stronger claim is proved using Owings's Separation Lemma.

We prove $\text{SIZE}_{n+1}(m) = \lceil \text{SIZE}_n(m) \rceil_{n+1}$ for $n \geq m$. As any pool in $\text{SIZE}_{n+1}(m)$ has at most m bitstrings, so has any selection of columns from such a pool. This proves inclusion from left to right. For the other direction, we must show that if every selection of n columns from an $(n+1)$-pool P has a size at most m, then so has P a size at most m. The following lemma proves this and hence the claim. ∎

3.20 Lemma

Let $m \leq n$ and let P be an $(n+1)$-pool such that for all distinct $i_1, \dots, i_n \in \mathbb{N}^*_{n+1}$ we have $|P[i_1, \dots, i_n]| \leq m$. Then $|P| \leq m$.

Proof. For the sake of contradiction assume $|P| > m$. Then $P \supseteq \{b_1, \dots, b_{m+1}\}$ for some pairwise different b_i. Consider an undirected graph with vertices b_i and the edges defined as follows: As $|P[1, \dots, j-1, j+1, \dots, n+1]| \leq m$ for every j, for each j there exist indices k_j and l_j such that b_{k_j} and b_{l_j} are equal on all positions but position j, where they differ. For each j, we connect the vertices b_{k_j} and b_{l_j} by an edge labelled j.

Now, we have constructed a graph with $m+1$ vertices and $n+1 \geq m+1$ different edges. Then there must exist some cycle $b_{i_1}, \dots, b_{i_p}$ in the graph. Let j be the label of the first edge of the cycle. Then b_{i_1} and b_{i_2} must differ on position j. But b_{i_2} and b_{i_3} must agree on position j as they are labelled differently. Also, b_{i_3} and b_{i_4} must agree on position j and so forth. But as b_{i_p} and b_{i_1} must also agree on position j, so must b_{i_1} and b_{i_2}—a contradiction. □

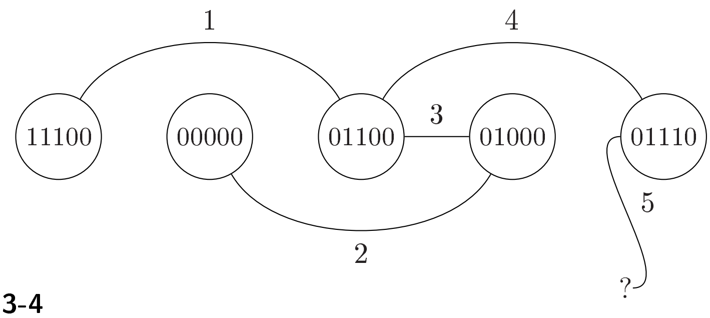


Figure 3-4
Example for the construction of Lemma 3.20 for $m = n = 4$. As described in the lemma, the edges are labelled with the position where connected vertices differ. The pool $P = \{11100, 00000, 01100, 01000, 01110\}$ cannot have the property, that every selection of four columns contains only four bitstrings. Although for the first four selection this is the case, the last selection must fail as this would create a cycle in the graph.

The name *cheatable* is perhaps best explained by the following anecdote. Professor Solomon intends to create a mean exam for her students: There are one hundred questions which must be answered with 'yes' or 'no'. The professor knows, that her students only possess polynomial time thinking capabilities and hence intends to pick the questions from a language known to be no element of polynomial time—thus ensuring that the students have really studied the topic. Having read about Theorem 2.13, Professor Solomon happily picks a five-cheatable language which is not in **P**.

The students, however, quickly note that the questions are taken from a cheatable language. Hence, they can compute *five possibilities for the correct answers to the one hundred questions* in polynomial time, and they can compute the four interesting questions, such that if they know the answers to these questions they know the answers to all questions. If the students succeed in *cheating on four questions*, they can pass the whole exam.

3.21 Example (Upward translation of the approximable families)
Translating the family $\text{APPROX}_m = \text{SIZE}_m(2^m - 1)$ upward does not yield a size family again. However, Beigel (1987) proved that for $n \geq m$ we have $\tau_{\lceil \text{APPROX}_m \rceil_n} \leq S(n,m) := \sum_{i=0}^{m-1} \binom{n}{i}$. In the following, a combinatorial proof is given for the slightly stronger result $\tau_{\lceil \text{APPROX}_m \rceil_n} = S(n,m)$. Spelled out, this means

1 $\lceil \text{APPROX}_m \rceil_n \subseteq \text{SIZE}_n(S(n,m))$,

2 $\lceil \text{APPROX}_m \rceil_n \nsubseteq \text{SIZE}_n(S(n,m) - 1)$.

The second point can be seen as follows: Consider the pool $\text{B}_{m-1}(0^n)$ which consists of all bitstrings containing at most $m-1$ bits 1. Then every selection of m distinct columns from $\text{B}_{m-1}(0^n)$ does not contain the bitstring 1^m and is hence an element of APPROX_m. Thus, $\text{B}_{m-1}(0^n) \in \lceil \text{APPROX}_m \rceil_n$. As $\text{B}_{m-1}(0^n)$ contains exactly one bitstring with no 1, $\binom{n}{1}$ bitstrings with exactly one 1, $\binom{n}{2}$ bitstrings with exactly two 1's and so forth, $|\text{B}_{m-1}(0^n)| = S(n,m)$.

The first point can be proved by showing that the pool $\text{B}_{m-1}(0^n)$ is in some sense the 'worst case'. The idea is to prove that every pool $Q \in \lceil \text{APPROX}_m \rceil_n$ has the property $|Q| \leq |\text{B}_{m-1}(0^n)|$.

Let $Q \in \lceil \text{APPROX}_m \rceil_n$ and for any distinct indices $i_1, \dots, i_m \in \mathbb{N}_n^*$ let $q_{i_1,\dots,i_m}$ be a bitstring missing from $Q\texttt{[}i_1, \dots, i_m\texttt{]}$. For example, for $\text{B}_{m-1}(0^n)$ such a bitstring is 1^m for all indices. Let $r_{i_1,\dots,i_m}$ be the same as $q_{i_1,\dots,i_m}$, except that we project the bit at the position j with $i_j = 1$ to 1, if such a position exists. Let R be the largest pool such that $R\texttt{[}i_1, \dots, i_m\texttt{]} \subseteq \mathbb{B}^m \setminus \{r_{i_1,\dots,i_m}\}$ for all distinct indices $i_1, \dots, i_m$. Then we obviously have $R \in \lceil \text{APPROX}_m \rceil_n$ as, indeed, every selection of distinct columns from R misses at least one bitstring. We now prove $|Q| \leq |R|$ by showing that the following mapping $\gamma \colon Q \to R$ is injective:

$$\gamma(b) := \begin{cases} b & \text{if } b \in R, \\ \pi_1^0(b) & \text{otherwise.} \end{cases}$$

First, we show that the range of γ is contained in R. If $b \in R$ then $\gamma(b) \in R$ by construction. So, assume $b \notin R$. Then we necessarily have $b\texttt{[1]} = 1$ by construction of the pool R. Now, if $\pi_1^0(b) \notin R$ then once more by construction we have $\pi_1^0(b)\,[i_1,\dots,i_m] = r_{i_1,\dots,i_m}$ for some appropriate indices. As the first position of $\pi_1^0(b)$ is trivially 0, none of the indices can select this first position—for we explicitly defined the bitstring $r_{i_1,\dots,i_m}$ to be 1 for such positions. But then $b\,[i_1,\dots,i_m] = \pi_1^0(b)\,[i_1,\dots,i_m] = r_{i_1,\dots,i_m} = q_{i_1,\dots,i_m}$. But as $b \in Q$ we have $b\,[i_1,\dots,i_m] \in Q\,[i_1,\dots,i_m]$ which is impossible since $q_{i_1,\dots,i_m}$ was explicitly setup to be no element of $Q\,[i_1,\dots,i_m]$. This shows $\gamma(b) = \pi_1^0(b) \in R$.

Second, we argue that γ is injective. It suffice to prove that if $b \notin R$ then $\pi_1^0(b) \notin Q$. Once more, there exist appropriate indices with $b\,[i_1,\dots,i_m] = r_{i_1,\dots,i_m}$. As $b \in Q$ we have $b\,[i_1,\dots,i_m] \neq q_{i_1,\dots,i_m}$. Hence, $r_{i_1,\dots,i_m}$ and $q_{i_1,\dots,i_m}$ differ. By construction this means that for some j we have $i_j = 1$ and in $r_{i_1,\dots,i_m}$ at this position there is a 1, while in $q_{i_1,\dots,i_m}$ there is a 0. But then $\pi_1^0(b)\,[i_1,\dots,i_m] = q_{i_1,\dots,i_m}$ and hence $\pi_1^0(b) \notin Q$.

We have now shown $|Q| \leq |R|$, where R has the property that any selection, which selects the first column of R, misses a bitstring with a 1 at that position. We now repeat the whole argument to obtain a pool R_2 with $|R| \leq |R_2|$ and for which any selection, which selects the first *two* columns of R_2, misses a bitstring with 1's at these positions. Repeating this process for all positions yields $|Q| \leq |R_n|$ where R_n is the largest pool satisfying $1^m \notin R\,[i_1,\dots,i_m]$ for all distinct indices. But that pool is $\mathrm{B}_{m-1}(0^n)$. ∎

3.6 Well-Foundedness of Inclusion on Cheatable Partial Information

A relation is *well-founded* if every non-empty subset of the relation has a minimal element or—equivalently—if there do not exist infinite descending chains. In this section we investigate, whether the inclusion relation is well-founded on the set of all partial information classes. This general problem, raised by Nickelsen (1999), is not yet solved fully. As a first step towards a solution, we show that the inclusion relation is well-founded on the set of all partial information classes which are subsets of $\boldsymbol{C}[\mathrm{CHEAT}]$.

Currently, no infinite descending chain in the set of all partial information classes in known. For some chains, the proof that they become stationary at some point is by no means simple, see for example Hinrichs and Wechsung (1997). It would be most satisfactory to prove in general that inclusion is well-founded on partial information classes.

3.22 Theorem
Let $\boldsymbol{FC}$ be c.c.c. Then inclusion is well-founded on $\{\, \boldsymbol{C}[\mathcal{F}] \mid \boldsymbol{C}[\mathcal{F}] \subseteq \boldsymbol{C}[\mathrm{CHEAT}] \,\}$.

Proof. We show that for each m the class $\boldsymbol{C}[\mathrm{CHEAT}_m]$ *has only a finite number of partial information subclasses*. That means, we claim that the cardinality of the set $\{\, \boldsymbol{C}[\mathcal{F}] \mid \boldsymbol{C}[\mathcal{F}] \subseteq \boldsymbol{C}[\mathrm{CHEAT}_m] \}$ is finite. This implies the claim as any infinite descending chain would have to start somewhere and there are no infinite descending chains in a finite relation.

Let $\boldsymbol{C}[\mathcal{F}] \subseteq \boldsymbol{C}[\text{CHEAT}_m]$ for some n-family $\mathcal{F}$. Note, that n may be much larger than m, but still Example 3.19 tells us $\boldsymbol{C}[\mathcal{F}] \subseteq \boldsymbol{C}[\text{SIZE}_n(m)]$ and hence $\boldsymbol{C}[\mathcal{F}] = \boldsymbol{C}[\mathcal{F} \cap \text{SIZE}_n(m)]$. Thus, for every class $\boldsymbol{C}[\mathcal{F}]$ there exists an n-family $\mathcal{G} := \mathcal{F} \cap \text{SIZE}_n(m)$ with $\boldsymbol{C}[\mathcal{F}] = \boldsymbol{C}[\mathcal{G}]$ and $\tau_{\mathcal{G}} \leq m$.

Theorem 3.10 states that there exists an antichain $\mathfrak{A}$ in $(\mathfrak{U}_n, \preceq)$, such that the bases of $\mathcal{G}$ are the representative systems of $\mathfrak{A}$. By definition, all pools in a unit from $\mathfrak{A}$ have the same eigenpool up to permutation. Hence, all of these eigenpools have the same size which is limited by m.

An eigenpool has the property that it has no duplicate columns and no constant columns. Hence, an eigenpool can be described by giving a list of its columns which may not contain duplicate columns or the constant columns. Furthermore, the 'height' of these columns is limited by the size of the pool, which is m in our case. *But there are only $2^m - 2$ different non-constant columns of height m.*

The number of different possible eigenpools of size m is *independent of n*. Hence, also the number of units in $(\mathfrak{U}_n, \preceq)$ whose pools have a size limited by m depends only on m, not on n. Finally, this implies that the number of antichains which are made up of such units is depended only on m. This proves the claim. □

Bibliographical Notes

Generating systems for families were first proposed in Nickelsen (1999). However, bases and antichains are a novel approach to the description of families.

Upward translation of specific families has been studied intensively in the literature, albeit often in disguised forms. For example, the upward translation of the selective family SEL_2 was implicitly given already by Selman (1979). Likewise the effects of upward translation on the cheatable and the approximable families have been studied by Beigel (1987).

The first general theorem on upward translation was proved in Nickelsen (1997), namely Lemma 3.11 which states that for any family for all greater indices there do always exist families which produce the same partial information classes. We first proved Theorem 3.12, which states the same but preserves normality of families in the upward translation process, in Tantau *et al.* (1998), however without the restriction that the indices must be pairwise different. This useful addition is due to Nickelsen (1999).

Well-founded relations are widely used in set theory. As a matter of fact, one of the axioms of Zermelo-Fraenkel set theory—the Axiom of Regularity—simply states that the relation $\in$ is well-founded. For more details on well-founded relations and set theory in general, see Jech (1997). The question whether inclusion is well-founded on partial information classes was first raised in Nickelsen (1999).

In Hinrichs and Wechsung (1997) it is shown that for all r the descending chain $\mathbf{P}_{\text{dist}}\big[\text{FREQ}_1(r)\big] \supseteq \mathbf{P}_{\text{dist}}\big[\text{FREQ}_2(r)\big] \supseteq \mathbf{P}_{\text{dist}}\big[\text{FREQ}_3(r)\big] \supseteq \dots$ becomes stationary at some point. This is an example of a chain outside the cheatable languages which becomes stationary.

3.7 Figures and Tables

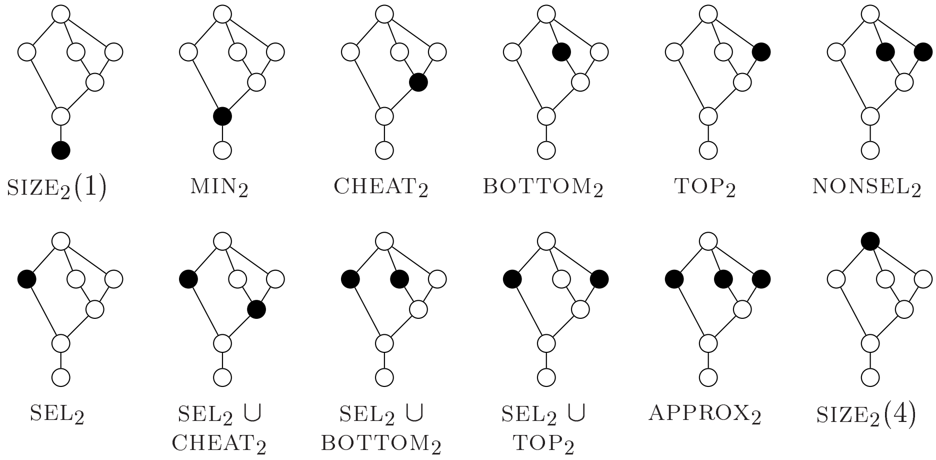


Figure 3-5
The twelve antichains in $(\mathfrak{U}_2, \preceq)$, see also Figure 3-1 on page 32. The partial information classes over recursive function classes of the families in the top line are recursive while those of the bottom line are not, see Fact 4.14.

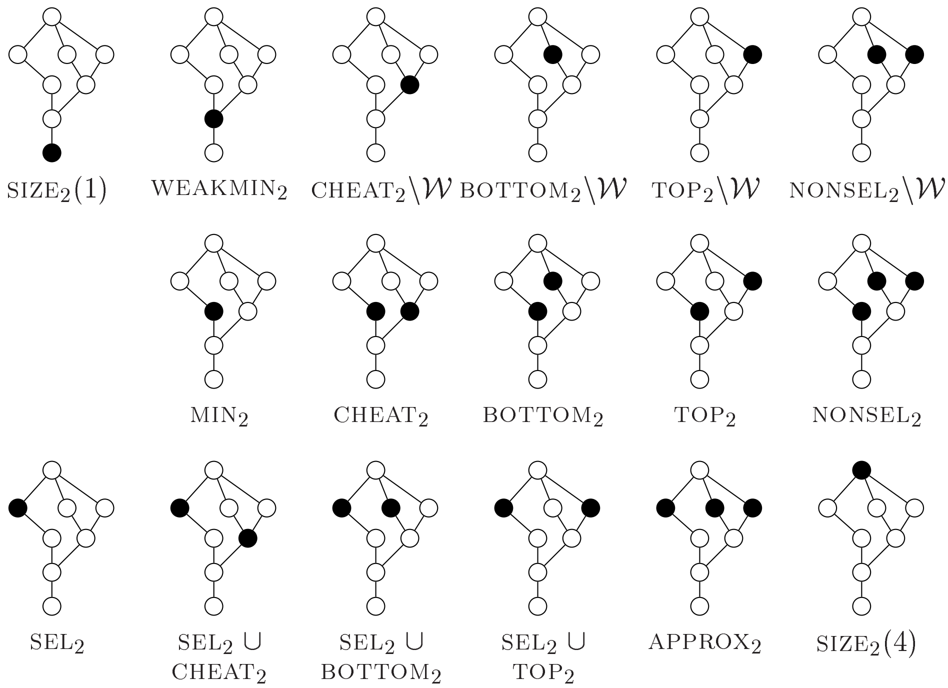


Figure 3-6
The seventeen antichains in $(\mathfrak{W}_2, \preceq)$. The first line contains, except for the trivial family, only families which are only weakly normal and miss $\mathcal{W} := \big\{\{00, 11\}\big\}$. The first two lines are recursive, the bottom line is not.

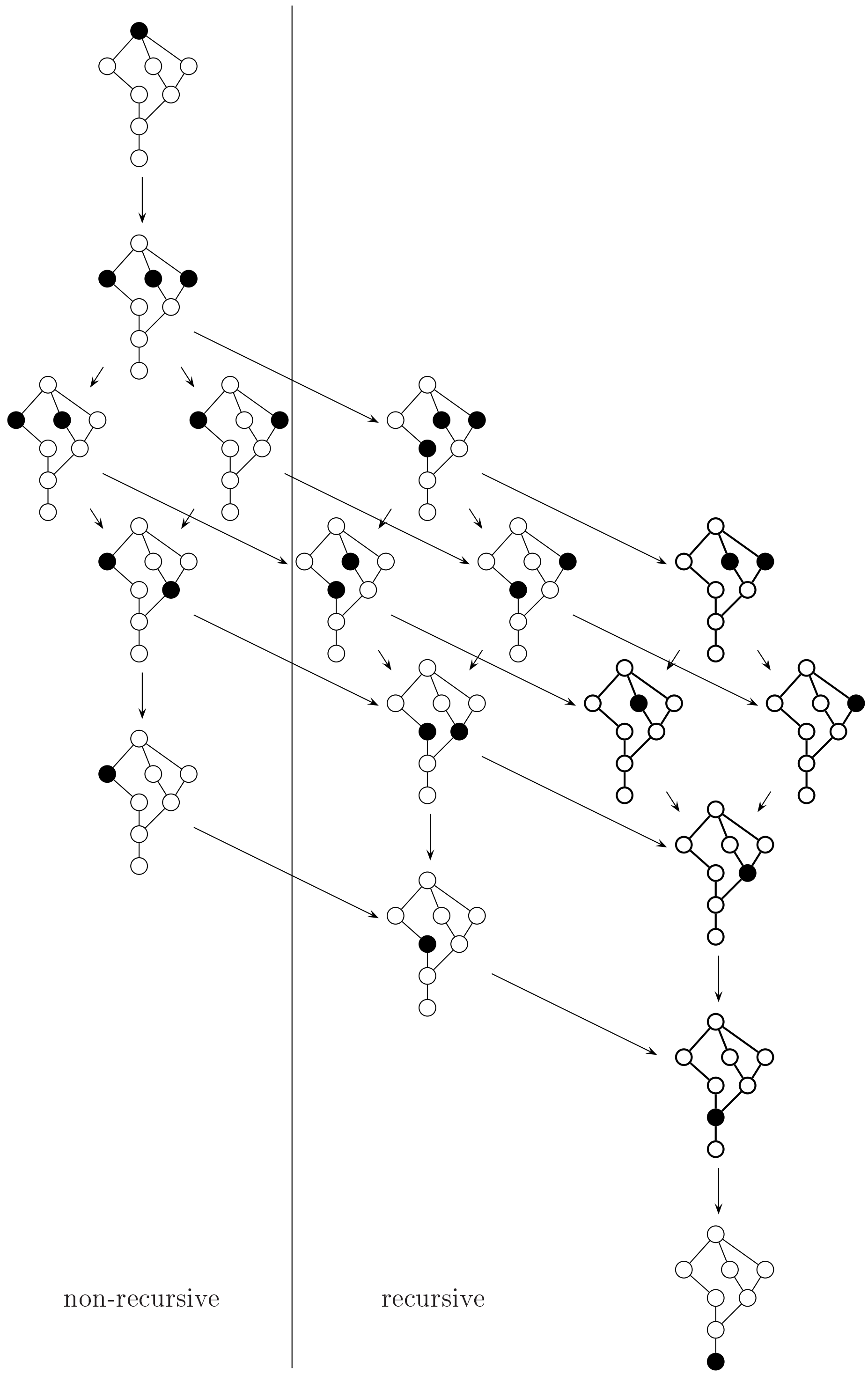


Figure 3-7
The weakly normal 2-families. The families shown in bold are weakly normal but not normal. An arrow from $\mathcal{F}$ to $\mathcal{G}$ indicates $\mathcal{G} \subsetneq \mathcal{F}$. As discussed in the next chapter, the partial information classes produced by the families to the right are recursive, the ones to the left are not.

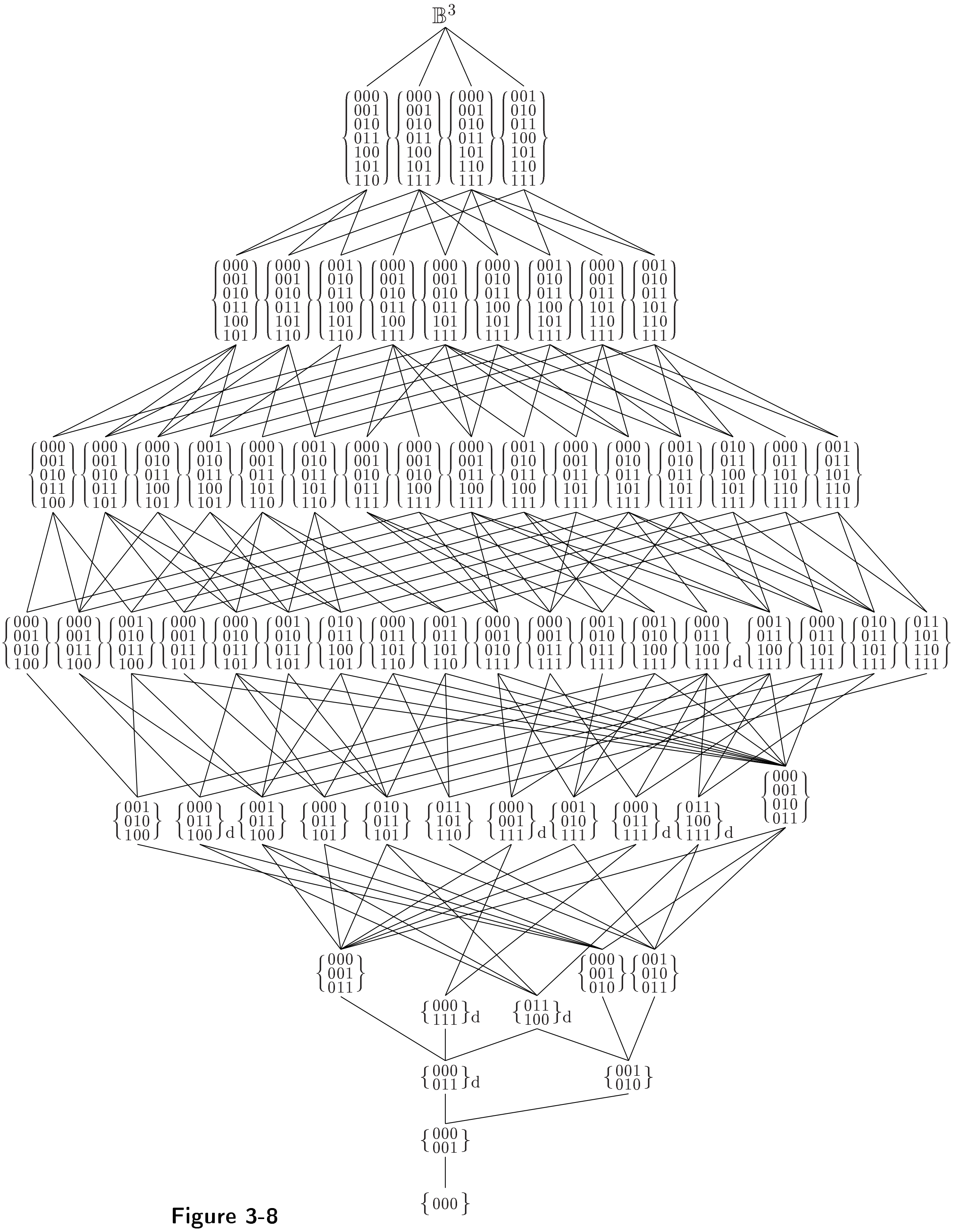


Figure 3-8

Visualisation of the poset $(\mathfrak{W}_3, \preceq)$ by representatives. For connected pools, the pool further down can be produced from the pool further up. Pools with subscript 'd' have duplicate non-constant columns.

Table 3-9

This table lists the numbers of units for $n \leq 5$ to the left and of weak units to the right. In a unit, all pools have the same size and the eigenpools are equal up to permutation. For example, the number 6 in the third line tells us, that there are six units, whose pools contain exactly three bitstrings and whose eigenpools have three columns. A typical such unit is $\{\{001, 010, 100\}\}$.

It took a C++ program about three hours on a 300 MHz CPU to compute the below numbers using an optimised brute-force counting algorithm.

	Units			Weak units		
Pool	Columns of Eigenpool			Columns of Eigenpool		
Size	Three	Four	Five	Three	Four	Five
1						
2				2	3	3
3	6	4	2	10	23	46
4	17	59	115	18	97	373
5	16	218	1 344	16	252	2 122
6	9	441	7 732	9	459	9 250
7	4	641	30 730	4	647	32 924
8	1	727	96 642	1	728	99 118
9		655	254 566		655	256 782
10		477	576 672		477	578 254
11		284	1 138 622		284	1 139 522
12		136	1 973 763		136	1 974 166
13		52	3 016 750		52	3 016 890
14		17	4 077 005		17	4 077 043
15		5	4 881 074		5	4 881 082
16		1	5 182 323		1	5 182 324
17			4 881 092			4 881 092
18			4 077 077			4 077 077
19			3 016 994			3 016 994
20			1 974 438			1 974 438
21			1 140 090			1 140 090
22			579 208			579 208
23			258 092			258 092
24			100 577			100 577
25			34 230			34 230
26			10 195			10 195
27			2 674			2 674
28			625			625
29			134			134
20			28			28
31			6			6
32			1			1
Sum	**53**	**3 717**	**37 312 801**	**60**	**3 836**	**37 325 360**

4

Structure of Stable Families

T. Smith, L. Jones, R. Brown and A. Green in their collected works
"A short introduction to the classical theory of the Piffle", Piffle Press, 6 gns.,
showed that all bi-universal Piffles were strictly descending
and conjectured that to prove a stronger result would be harder.

It is this conjecture which motivated the present paper.

— A. K. Austin, *The Mathematical Gazette*, 51:149–150, 1967

Originally, partial information classes were studied in recursion theory only. The first definition of a *resource bounded* partial information class—the p-selective languages—was given in 1979 by Selman, and the systematic study of such classes only began in the last decade. Surprisingly, while the inclusion problem for resource bounded partial information classes is completely solved by Theorem 2.13, *for recursively computable partial information deciding inclusion is still an open problem.* Using the following definition, this can be rephrased: *No algorithm is known for deciding whether a family is stable.*

4.1 Definition (Stable Family)
A family $\mathcal{F}$ is in *stable normal form*, or just *stable*, if it is subset closed and there exists no subset closed family $\mathcal{F}' \subsetneq \mathcal{F}$ such that $\mathbf{REC}[\mathcal{F}] = \mathbf{REC}[\mathcal{F}']$.

This chapter makes some progress on the question which families are stable. The first section introduces the simple but useful concept of *hard tuples*. Based on this concept, the second section introduces *hard remainder pools*. Such pools are obtained by shrinking the size of a given pool using the knowledge that some input words are hard—the hard remainder pool then contains all bitstrings which could not be eliminated.

In the third section we prove that for all languages either no tuple is hard or hard remainder pools can be computed for all words. As a corollary we obtain the *Generalised Non-Speedup Theorem* due to Beigel, Kummer and Stephan.

At the end, the fourth section identifies all stable 2-families.

4.1 Definition of Hard Tuples

This section introduces *hard tuples*. A language for which *no* word tuple is hard is relatively, well, easy to decide. Fortunately, even if some tuple is hard for a language, *hard remainder pools* defined in the next section can still be computed.

To fix notations, *in the following the hard tuples will always be m-tuples. The families $\mathcal{E}$ and $\mathcal{F}$ will always be m- and n-families, respectively, with $m < n$.*

Consider a language $L \in \boldsymbol{C}[\mathcal{F}]$ via some function $f \in \boldsymbol{FC}$. Given words $w_1, \dots, w_m$ we might try *to produce a pool from $\mathcal{E}$ for these words* as follows: We fix a set $S \subseteq \Sigma^*$ and then pick some words $w_{m+1}, \dots, w_n \in S$ and compute $f(w_1, \dots, w_n) =: P \in \mathcal{F}$. *The first m columns of this pool, call them Q, may happen to be an element of $\mathcal{E}$.* But then, we have computed a pool $Q \in \mathcal{E}$ for the given words. If we did not succeed, i. e., if $Q \notin \mathcal{E}$, we simply try some other extra words from the set S.

If $\mathcal{E}$ is not too small and the set S sufficiently large, the algorithm may succeed sooner or later for a large number of tuples. Naturally, for some tuples the algorithm may fail no matter how hard we try, but in this case we have every right to call the words $w_1, \dots, w_m$ *hard* over S via f.

4.2 Definition (Hard Tuple)
Let $\boldsymbol{FC}$ be c.c.c. and $L \in \boldsymbol{C}[\mathcal{F}]$ via $f \in \boldsymbol{FC}$ and let $S \subseteq \Sigma^*$. A tuple $w_1, \dots, w_m$ of words is *$\mathcal{E}$-hard over S via f*, if there do not exist words $w_{m+1}, \dots, w_n \in S$ such that the first m columns of $f(w_1, \dots, w_n)$ form a pool from $\mathcal{E}$.

4.3 Example (Hard tuples for the trivial family)
Consider the family $\mathcal{E} := \text{CHEAT}_1$. Here, $m = 1$ and 'tuples' become single words. A word w is CHEAT_1-hard via a function f over a set S, if for no words $w_2, \dots, w_n$ in S the pool $f(w, w_2, \dots, w_m)$ has *a constant first column*. Hence, for a hard word every pool produced by the function f for w and words from S *has a non-constant first column*, i. e., contains both 0 and 1. ▯

When dealing with the class of all recursive functions as $\boldsymbol{FC}$, as we will generally do in this chapter, we may pick a fairly large set S, namely $S = \Sigma^*$.

4.4 Lemma
Let $L \in \mathbf{REC}[\mathcal{F}]$ via f and let no m-tuple be $\mathcal{E}$-hard over Σ^* via f. Then we also have $L \in \mathbf{REC}[\mathcal{E}]$.

Proof. Let $\big(\langle w^i_{m+1}, \dots w^i_n\rangle\big)_{i\in\mathbb{N}}$ be an enumeration of all $(n-m)$-tuples of words. Given words $w_1, \dots, w_m$ we repeatedly compute $f(w_1, \dots, w_m, w^i_{m+1}, \dots, w^i_n)$ until the first m columns are an element of $\mathcal{E}$. As no tuple is $\mathcal{E}$-hard, the computation necessarily finishes after some finite time. Thus $L \in \mathbf{REC}[\mathcal{E}]$. ☐

Note, that if $L \notin \mathbf{REC}[\mathcal{E}]$ there must exist an $\mathcal{E}$-hard tuple over Σ^* via f. Example 4.5 below shows that this has a number of interesting consequences.

4.2 Definition of Hard Remainder Pools

Lemma 4.4 from the previous section showed how to compute partial information for some language, if no tuple is hard. The following example demonstrates how we can compute partial information if there do exist hard tuples. Definition 4.9 generalises the obtained results by defining *hard remainder pools*.

4.5 Example (Recursively cheatable languages are recursive)
Let L be a recursively, say, 5-cheatable language, i. e., let $L \in \mathbf{REC}[\text{CHEAT}_5]$ via some recursive function f. We claim that L is also recursively 4-cheatable. The family $\mathcal{E}$ will be the family CHEAT_4.

If no tuple is CHEAT_4-hard over Σ^* via f, Lemma 4.4 states $L \in \mathbf{REC}[\text{CHEAT}_4]$ and we are done. So, assume that there exists a CHEAT_4-hard tuple $w_1, \dots, w_4$.

Let $b := \chi_L(w_1, \dots, w_4)$. This is a fixed bitstring. For any given word w compute the pool $P := f(w_1, \dots, w_4, w)$. *As the tuple is* CHEAT_4*-hard, the first four columns of* P*, call them* Q*, are no element of* CHEAT_4. By definition, this means $|Q| > 4$ and as $P \in \text{CHEAT}_5$ we even have $|Q| = 5$. Among the five different bitstrings in Q only one can be correct, namely b. But then, we also know the characteristic value of the last word w. Hence, L is even recursive.

As we can reapply the argument, all recursively cheatable languages are recursive. This fact was first proved by Richard Beigel is his PhD thesis where he labelled it the *Non-Speedup Theorem*. The funny name is explained in the remark after Corollary 4.13 below. ▯

4.6 Theorem (Non-Speedup Theorem)
We have $\mathbf{REC} = \mathbf{REC}[\text{CHEAT}]$.

4.7 Corollary
No non-trivial family contained in a cheatable family is stable.

Let's analyse the argument of Example 4.5 from an abstract point of view. First, we checked if no tuple of m words from a set S was $\mathcal{E}$-hard over S via the function f. If so, we could easily produce a pool from $\mathcal{E}$ for all tuples.

Second, if there existed an $\mathcal{E}$-hard tuple, we acquired its characteristic string. For any $n - m$ words we prepended the hard tuple and produced a pool from $\mathcal{F}$. Knowing the correct value for the first m words, we deleted all bitstrings from this pool which were incorrect for these positions, yielding what will be called a *remainder pool*.

The important point is, that *we can guarantee to delete several bitstrings, because the first* m *columns are no pool from* $\mathcal{E}$. Because the words for which we knew the characteristic string formed a hard tuple, the remainder pool will be called a *hard remainder pool*. Note that the below definitions of hard remainder pools is purely combinatorial—it neither refers to sets S nor to functions f.

4.8 Definition (Remainder Pool)

Let P be an n-pool and $b \in \mathbb{B}^m$ be a bitstring. The *remainder of P knowing b* is the $(n-m)$-pool $\{c \in \mathbb{B}^{n-m} \mid bc \in P\}$. A pool is *a remainder pool of P*, if it is contained the remainder of P knowing some bitstring b.

4.9 Definition (Hard Remainder Pool)

A remainder pool of a pool P is an *$\mathcal{E}$-hard remainder pool of P*, if the first m columns of P form no pool from $\mathcal{E}$. The set of all $\mathcal{E}$-hard remainder pools of pools from a family $\mathcal{F}$ will be denoted $\mathcal{F} : \mathcal{E}$.

4.10 Lemma (Basic Properties of Hard Remainder Pools)

1 *If $\mathcal{F} \subseteq \mathcal{F}'$ then $\mathcal{F} : \mathcal{E} \subseteq \mathcal{F}' : \mathcal{E}$, and if $\mathcal{E} \subseteq \mathcal{E}'$ then $\mathcal{F} : \mathcal{E} \supseteq \mathcal{F} : \mathcal{E}'$,*

2 *$(\mathcal{F} \cup \mathcal{F}') : \mathcal{E} = \mathcal{F} : \mathcal{E} \cup \mathcal{F}' : \mathcal{E}$ and $(\mathcal{F} \cap \mathcal{F}') : \mathcal{E} \subseteq \mathcal{F} : \mathcal{E} \cap \mathcal{F}' : \mathcal{E}$,*

3 *$\mathcal{F} : (\mathcal{E} \cap \mathcal{E}') = \mathcal{F} : \mathcal{E} \cup \mathcal{F} : \mathcal{E}'$ and $\mathcal{F} : (\mathcal{E} \cup \mathcal{E}') \subseteq \mathcal{F} : \mathcal{E} \cap \mathcal{F} : \mathcal{E}'$,*

4 *$\lceil \mathcal{F} \rceil_{n+m} : \mathcal{E} \subseteq \mathcal{F}$.*

Proof. As all but the last property follow directly from the definition, we argue only for the last claim. The selection of the last $(n+m)-m = n$ many columns of any pool from $\lceil \mathcal{F} \rceil_{n+m}$ is a pool from $\mathcal{F}$ by definition of upward translation. Hence, no matter what pools are hard or what bitstrings we know, remainder pools of pools from $\lceil \mathcal{F} \rceil_{n+m}$ are always in $\mathcal{F}$. □

$$\begin{tikzcd}
 & \left\{\begin{smallmatrix}00\ 00\\ 00\ 01\\ 00\ 11\end{smallmatrix}\right\} \arrow[r] & \{00, 01, 11\} =: R_1 \\
P = \left\{\begin{smallmatrix}00\ 00\\ 00\ 01\\ 00\ 11\\ 01\ 11\\ 11\ 11\end{smallmatrix}\right\} \arrow[ur, "b = 00"] \arrow[r, "b = 01"] \arrow[dr, "b = 11"'] & \{01\ 11\} \arrow[r] & \{11\} =: R_2 \\
 & \{11\ 11\} \arrow[r] & \{11\} =: R_3
\end{tikzcd}$$

Figure 4-1

The pools R_1 to R_3 are the remainders of P knowing the different b's. The pools R_i are $\textsc{bottom}_2$-hard remainder pools of P, but not $\textsc{sel}_2$-hard remainder pools of it. The reason is that the first two columns of P are an element of $\textsc{sel}_2$, but no element of $\textsc{bottom}_2$.

4.11 Example (Hard remainder pools of size families)

A useful equation is $\textsc{size}_n(k) : \textsc{size}_m(h) = \textsc{size}_{n-m}(k-h)$. To see this, consider some $\textsc{size}_m(h)$-hard remainder pool $Q = \{c \in \mathbb{B}^{n-m} \mid bc \in P\}$ of a pool $P \in \textsc{size}_n(k)$. By definition, the first m columns of P are no element of $\textsc{size}_m(h)$. But then, these columns must contain *at least $h+1$ different bitstrings*. Hence, the number of bitstrings in P that begin with b cannot be more than $k-h$ which implies $Q \in \textsc{size}_{n-m}(k-h)$. The other way round, it is not too hard to see that all pools in $\textsc{size}_{n-m}(k-h)$ are indeed $\textsc{size}_m(h)$-hard remainder pools of pools from $\textsc{size}_n(k)$. □

Hard remainder pools are exactly those pools which can be computed if the characteristic string of some hard words is known.

4.3 Computing Partial Information using Hard Tuples

This section studies applications of hard tuples to the computation of partial information. Theorem 4.12 below states that for any language either no tuple is $\mathcal{E}$-hard or pools from $\mathcal{F} : \mathcal{E}$ can be computed for all words. As an application, we prove the Generalised Non-Speedup Theorem.

At the end of this section, we show that hard tuples are not only useful in the recursive case. Theorems 4.18 and 4.19 give two applications of hard tuples in the resource bounded case.

Hard Tuples and Recursively Computable Partial Information

4.12 Theorem
We have $\mathbf{REC}[\mathcal{F}] \subseteq \mathbf{REC}[\mathcal{F} : \mathcal{E}] \cup \mathbf{REC}[\mathcal{E}]$.

Proof. Let $L \in \mathbf{REC}[\mathcal{F}]$ via a recursive function f. First, if no n-tuple is $\mathcal{E}$-hard by Lemma 4.4 we have $L \in \mathbf{REC}[\mathcal{E}]$.

If there exists some $\mathcal{E}$-hard tuple $w_1, \dots, w_m$ let $b := \chi_L(w_1, \dots, w_m)$. Given any words $w_{m+1}, \dots, w_n$ we can compute a pool $P := f(w_1, \dots, w_n)$. The first m columns of P *are not an element of* $\mathcal{E}$ *as the tuple is* $\mathcal{E}$*-hard*. As we know the correct value for the first m bits of the bitstrings in P, namely b, we can delete all bitstrings from P which do not begin with b, yielding the remainder of P knowing b, which is a pool for the given words. As the words $w_1, \dots, w_m$ were $\mathcal{E}$-hard, this remainder pool is even an $\mathcal{E}$-hard remainder pool and is hence an element of $\mathcal{F} : \mathcal{E}$. □

By Example 4.11, we have $\text{SIZE}_n(k) : \text{SIZE}_m(h) = \text{SIZE}_{n-m}(k-h)$. Hence, we get the following corollary, which was first proved in Beigel *et al.* (1992).

4.13 Corollary (Generalised Non-Speedup Theorem)
Let $m < n$ *and* $h < k$. *Then*

$$\mathbf{REC}[\text{SIZE}_n(k)] \subseteq \mathbf{REC}[\text{SIZE}_{n-m}(k-h)] \cup \mathbf{REC}[\text{SIZE}_m(h)].$$

The name *Generalised* Non-Speedup Theorem stems from the following argument: If we set $m = h = 1$, we get $\mathbf{REC}[\text{CHEAT}_n] \subseteq \mathbf{REC}[\text{CHEAT}_{n-1}]$ and reapplying the argument yields $\mathbf{REC}[\text{CHEAT}_n] = \mathbf{REC}$, i.e., the Non-Speedup Theorem.

But why *non-speedup*? Consider some non-recursive language L. In order to decide such a language with a Turing machine, the machine must obviously be an oracle Turing machine equipped with a powerful non-recursive oracle X. Given n words, we can easily decide membership of these words with respect to L with n queries by taking $X = L$ and simply querying the input words. But is it possible to make do with *less* queries for some other, clever, oracle?

Assume that there did exist some oracle X such that we could make do with $\log_2 n$ adaptive queries. In this case, L *would be recursively cheatable*. To see this, note that each of the $2^{\log_2 n} = n$ different answers of the oracle induces one possible characteristic string. That would mean, that the language were recursively cheatable and hence recursive.

Thus, the Non-Speedup Theorem asserts that for *any non-recursive language* and *any oracle* the characteristic string of n given words *cannot be computed with less than* $\lfloor \log_2 n \rfloor + 1$ *queries* to the oracle, in general.

Can the Non-Speedup Theorem be strengthened further? More precisely, which families above the cheatable families have recursive partial information classes? Phrased differently, which families above the cheatable families are instable and collapse to the trivial family? By our very first example of a partial information class, Example 1.12, such a family cannot contain the selective pools.

Fortunately, the question has been solved in a most satisfactory way: *All* subset closed families which do not contain the selective family produce recursive partial information classes. Hence, selectivity is the exact dividing line between recursive and non-recursive partial information. A proof for this can be found in Nickelsen (1997) where an earlier result of Kummer (1992) known as the Cardinality Theorem is extended.

If we introduce the name NONSEL_n for the largest n-family that does not contain the selective family, we get:

4.14 Fact
We have $\mathbf{REC} = \mathbf{REC}[\text{NONSEL}]$.

4.15 Corollary
No non-trivial normal family missing a maximal selective pool is stable.

As a second application of Theorem 4.12 we show that for recursive selectivity, allowing cheatable pools to be output does not change the partial information class.

4.16 Theorem
For $n \geq 2$ *we have* $\mathbf{REC}[\text{SEL}_n \cup \text{CHEAT}_n] = \mathbf{REC}[\text{SEL}]$.

Proof. First, $(\text{SEL}_n \cup \text{CHEAT}_n) : \text{CHEAT}_1 = \text{SEL}_{n-1} \cup \text{CHEAT}_{n-1}$ by Lemma 4.10. Hence, we easily get that $\mathbf{REC}[\text{SEL}_n \cup \text{CHEAT}_n]$ is contained in $\mathbf{REC}[\text{SEL}_2 \cup \text{CHEAT}_2]$. The upward translation of $\text{SEL}_2 \cup \text{CHEAT}_2$ is $\text{SEL}_3 \cup \text{SIZE}_3(2)$. This can be seen by checking that in Figure 3-8 on page 44, indeed, no pools other than the pools in $\text{SEL}_3 \cup \text{SIZE}_3(2)$ have the property that all selections of two different columns yields either a selective or a cheatable pool. But then, as $\bigl(\text{SEL}_3 \cup \text{SIZE}_3(2)\bigr) : \text{CHEAT}_1 = \text{SEL}_2$ we get the claim. □

Hard Tuples and Resource Bounded Partial Information

Hard tuples are also useful when examining partial information classes over function classes much smaller than the class of recursive functions. However, if we consider function classes like $\mathbf{FP}$, it certainly no longer makes sense to have $S = \Sigma^*$. Instead, for a word w we will generally consider the set $\Sigma^{\leq w}$ of words of maximum length $|w|$. Naturally, even for this set we cannot hope to examine it fully in deterministic polynomial time—but in *non-deterministic* polynomial time we can.

The following theorem demonstrates that hard tuples can also be used to identify the amount of *advice* necessary to decide a partial information class. The notion of advice classes is due to Karp and Lipton (1980). For more details on results concerning advice bounds and partial information classes, please see the bibliographical notes.

4.17 Definition (Advice Class, Advice Function)
Let $\boldsymbol{K}$ be a class of languages and $\boldsymbol{F}$ be a class of functions from $\mathbb{N}$ to $\mathbb{N}$. Then L is in the *advice class* $\boldsymbol{K}/\boldsymbol{F}$, if there exists some *advice function* $h\colon \mathbb{N} \to \Sigma^*$ such the mapping $n \mapsto |h(n)|$ is in $\boldsymbol{F}$ and the language $\{\,\langle w, h(|w|)\rangle \mid w \in L\,\}$ is in $\boldsymbol{K}$.

4.18 Theorem
We have $\mathbf{P}[\text{MIN}] \subseteq \Delta_2\mathbf{P}/1$.

Proof. Before we start the specialised proof, consider once more the general situation where we have $L \in \mathbf{P}[\mathcal{F}]$ via some polynomial time computable function f and where $\mathcal{E}$ is some arbitrary m-family. In this situation, let $w_1, \dots, w_m$ be words of maximum length l.

1 The set X_1 of m-tuples which are not $\mathcal{E}$-hard over $\Sigma^{\leq l}$ via f is in $\mathbf{NP}$: For an input tuple we simply 'guess' some further words $w_{m+1}, \dots, w_n \in \Sigma^{\leq l}$ and then compute the pool $f(w_1, \dots, w_n)$. If the first n columns are a pool from $\mathcal{E}$ we accept. Otherwise we reject.

2 It is an $\mathbf{FP}^{\mathbf{NP}}$-problem to compute a pool from $\mathcal{E}$ for a tuple $w_1, \dots, w_m$ which is not $\mathcal{E}$-hard over $\Sigma^{\leq l}$ via f. To see this, consider a machine which enumerates all pools in $\mathcal{E}$ and for each $P \in \mathcal{E}$ it queries an oracle X_2 about the word $\langle w_1, \dots, w_m, P\rangle$. The first pool for which the oracle answers 'yes' for the query is output. The oracle X_2 contains a tuple $\langle w_1, \dots, w_m, P\rangle$, iff the first m columns of $f(w_1, \dots, w_n)$ are P for some additional words $w_{m+1}, \dots, w_n \in \Sigma^{\leq l}$. By the same argument as above, we have $X_2 \in \mathbf{NP}$. Note, that as the input tuple is not $\mathcal{E}$-hard, *some* pool P is indeed output.

We now specialise our argument to $\mathcal{F} = \text{MIN}_2$ and $\mathcal{E} = \text{CHEAT}_1$. Let L be some language in $\mathbf{P}[\text{MIN}_2]$. Note, that by Example 3.17 we have $\mathbf{P}[\text{MIN}_2] = \mathbf{P}[\text{MIN}]$. We must prove $L \in \Delta_2\mathbf{P}/1$, i.e., we must show that the language L can be decided by a deterministic polynomial time algorithm that may query an $\mathbf{NP}$ oracle and which may use one bit of advice per word length. The $\mathbf{NP}$ oracle will be the join of X_1 and X_2.

To decide L, upon input w we first check, using X_1, if this word is not CHEAT_1-hard for $\Sigma^{\leq |w|}$. If it is not, we use X_2 to compute a pool from $\mathcal{E} = \text{CHEAT}_1$, thus deciding the word w.

The family MIN_2 has a special property: Any two words which are hard over the same set $\Sigma^{\leq l}$ have the same characteristic value. To see this, note that the only pool in MIN_2 that does not have a constant column is the pool $\{00, 11\}$. Hence, to decide a *hard* word, apart from the queries to the **NP**-oracles, we only need one further bit of knowledge for each length l. This extra bit tells us whether the hard words of length l are in the language or not. □

The astute reader may have noticed that the last argument is not entirely precise. In order to ensure that $\{00, 11\}$ is indeed a pool for any two hard words, the words must acutally not only be hard via f but also via f' where $f'(u, v) := f(v, u)$. Naturally, the proof can be modified to take care of this problem also.

The next theorem addresses the following question: *Given a c.c.c. function class $\boldsymbol{FC}$, is every language $\boldsymbol{C}[\text{CHEAT}_2]$ the symmetric difference of languages in $\boldsymbol{C}[\text{MIN}_2]$?* We answers this question affirmatively for polynomial space. It is not clear how the following proof might be adopted to polynomial time.

4.19 Theorem

Every language in $\mathbf{PSPACE}[\text{CHEAT}_2]$ is the symmetric difference of a language in $\mathbf{PSPACE}[\text{MIN}_2]$ and a language in $\mathbf{PSPACE}$.

Proof. Let $L \in \mathbf{PSPACE}[\text{CHEAT}_2]$ via some function f computable in polynomial space. We must construct two languages $M \in \mathbf{PSPACE}[\text{MIN}_2]$ and $N \in \mathbf{PSPACE}$ such that $L = M \triangle N$. Like in the previous proof, hardness of words will once more refer to CHEAT_1-hardness in the following.

The key idea of the proof is the following definition of the *lookup word l_w* of a word w. For words w which are not CHEAT_1-hard over $\Sigma^{\leq |w|}$, the lookup word is w itself. However, if w is hard, we define the lookup word as *the lexicographically smallest word which is hard over $\Sigma^{\leq |w|}$ via f*.

The lookup word is computable in polynomial space, as we can easily check whether a word is hard in polynomial space; and if it is, we only have to search for the smallest word u in $\Sigma^{\leq |w|}$ such that for all other words v in $\Sigma^{\leq |w|}$ the pool $f(u, v)$ does not have a constant first column, thus obtaining the smallest word CHEAT_1-hard over $\Sigma^{\leq |w|}$.

Using lookup words, we can now define the languages M and N. For $w \in \Sigma^*$ let

$$\chi_M(w) := \chi_L(l_w),$$
$$\chi_N(w) := \chi_L(l_w) \oplus \chi_L(w).$$

Note, that we have $\chi_N(w) = \chi_M(w) \oplus \chi_L(w)$ and hence $L = M \triangle N$.

Deciding *N* in polynomial space

In order to show $N \in \mathbf{PSPACE}$, let w be an input word. First, we compute the lookup word l_w which can be done in polynomial space as argued above. We must compute $\chi_L(w) \oplus \chi_L(l_w)$. If $l_w = w$ we are done. Otherwise, both w and l_w are hard over all of $\Sigma^{\leq |w|}$. Specifically, both $f(l_w, w)$ and $f(w, l_w)$ have non-constant first columns. From this, we can deduce whether $\{00, 11\}$ or $\{01, 10\}$ is a pool for $\chi_L(l_w, w)$. In the first case, $\chi_L(w) \oplus \chi_L(l_w) = 0$ and in the second case $\chi_L(w) \oplus \chi_L(l_w) = 1$.

Computing minimal partial information for *M*

Next, given words u and v we must compute a pool for them from MIN_2 for the language M in polynomial space. Once more, we first compute the lookup words l_u and l_v.

We must compute a pool for the characteristic string $\chi_M(u,v)$. By definition, we have $\chi_M(u,v) = \chi_L(l_u,l_v)$. But then, we only need to compute a pool for l_u and l_v *with respect to L*. If $l_u = l_v$ this is trivial, as then $\{00,11\} \in \mathrm{MIN}_2$ is such a pool. Otherwise, assume l_u is the lexicographically smaller word—the other case is symmetric. As l_v is the smallest word which is hard over all of $\Sigma^{\leq |v|}$ by definition, the word l_u cannot also be hard over $\Sigma^{\leq |v|}$. But then, we can compute $\chi_L(l_u)$ in polynomial space and can hence output a pool from MIN_2 for $\chi_L(l_u,l_v)$. □

4.4 Computing Partial Information for Branches

After the small excursion at the end of the previous section, this last section returns to the original problem of identifying stable families. This section identifies all stable 2-families. We have already eliminated a large number of candidates. Except for the trivial family, only the family SEL_2 can be stable and families beyond $\mathrm{SEL}_2 \cup \mathrm{CHEAT}_2$. There are only four such other families which are also normal, namely $\mathrm{SEL}_2 \cup \mathrm{BOTTOM}_2$, $\mathrm{SEL}_2 \cup \mathrm{TOP}_2$, APPROX_2 and finally $\mathrm{SIZE}_2(4)$. All of these families are stable as Corollary 4.21 below proves.

In our second example of a partial information class, Example 1.13, we showed that the *closed chains* of in partial ordering are in $\mathbf{REC}[\mathrm{SEL}_2 \cup \mathrm{BOTTOM}_2]$. Furthermore, we gave a canonical example of interesting closed chains in a partial ordering: All *branches* in the *standard tree* $(\Sigma^*, \sqsubseteq)$ are closed chains. We now prove that *some branches are not recursively selective*. Special thanks once more to Arfst Nickelsen for pointing out the idea of the proof.

4.20 Theorem
There exists a branch in the standard tree that is not an element of $\mathbf{REC}[\mathrm{SEL}]$.

Proof. We construct a branch B as follows: Let M^i be an enumeration of all machines that could possibly witness $B \in \mathbf{REC}[\mathrm{SEL}_2]$. For each $i \in \mathbb{N}$ we will put exactly one word of length $i+1$ into B. Assume, that B is already constructed up to level i, i. e., we have constructed a path $w^0, \ldots, w^i$ to some node w^i of length i. For the two successors w^i0 and w^i1 of w^i compute $M^i(w^i0, w^i1)$. If the machine fails to produce a pool from SEL_2, we extend the path arbitrarily—for example by choosing $w^{i+1} := w^i0$. Otherwise, the machine outputs a pool from SEL_2. One of the two bitstrings 01 and 10 is not in this pool. In the first case set $w^{i+1} := w^i1$, in the other case set $w^{i+1} := w^i0$. In either case, the machine M^i errs on the words w^i0 and w^i1. Thus $B := \{\, w^i \mid i \in \mathbb{N} \,\}$ is a branch, but $B \notin \mathbf{REC}[\mathrm{SEL}_2]$. □

We have seen that closed chains in a partial ordering are in $\mathbf{P}[\mathrm{SEL}_2 \cup \mathrm{BOTTOM}_2]$ and that some closed chains are not selective.

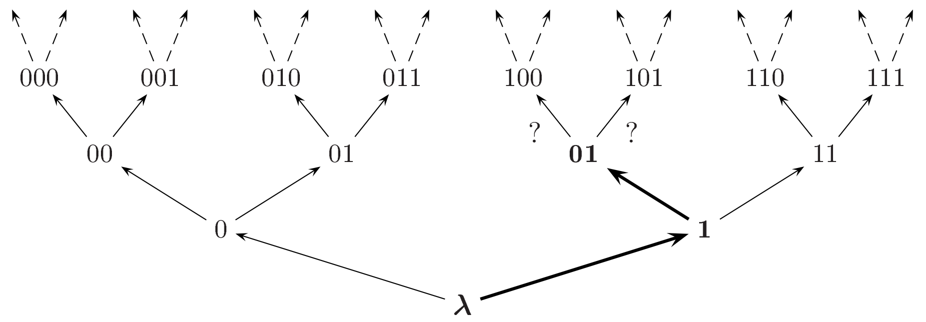


Figure 4-2
The standard tree $(\Sigma^*, \sqsubseteq)$ used in the construction of Theorem 4.20. In the depicted situation, the words $w^0 = \lambda$, $w^1 = 1$ and $w^2 = 10$ have already been constructed. The path is continued via 100 or via 101 such that the machine M^2 errs for these two words, i. e., we ensure that if the machine outputs a selective pool, this pool it incorrect.

4.21 Corollary
The following proper inclusions hold:

$$\begin{array}{ccc} & \mathbf{REC}[\text{APPROX}_2] & \\ \subsetneq & & \supsetneq \\ \mathbf{REC}[\text{SEL}_2 \cup \text{BOTTOM}_2] & \neq & \mathbf{REC}[\text{SEL}_2 \cup \text{TOP}_2] \\ \supsetneq & & \subsetneq \\ & \mathbf{REC}[\text{SEL}_2] & \end{array}$$

Proof. By Theorem 4.20 recursive selectivity is a proper subset of the class to the left. As the class to the right is just the complement of the left class, neither can the right class be equal to recursive selectivity. As $\mathbf{REC}[\text{SEL}_2]$ is the intersection of the two classes above it, these classes cannot be equal. Lastly, neither the left nor the right class can be equal to the class at the top, as $\mathbf{REC}[\text{APPROX}_2]$ is closed under complement which the left and right classes are not. $\square$

Bibliographical Notes

Despite the fact that recursively computable partial information has been studied for quite some time, no general procedure is known for deciding stability of families

A first result on the collapse of instable families is the *Non-Speedup Theorem* which Richard Beigel proved in his PhD thesis. He also conjectured that a stronger result holds, namely that all *recursively easily countable languages* are recursive. A language L is easily n-countable, if for any n words $w_1, \dots, w_n$ we can *exclude* one possibility for the cardinality of $L \cap \{w_1, \dots, w_n\}$. It quite straightforward to see that the easily countable languages form a partial information class over an appropriate family, called $\text{CARD}_n(n)$, and that we have $\text{CHEAT}_n \subsetneq \text{CARD}_n(n)$.

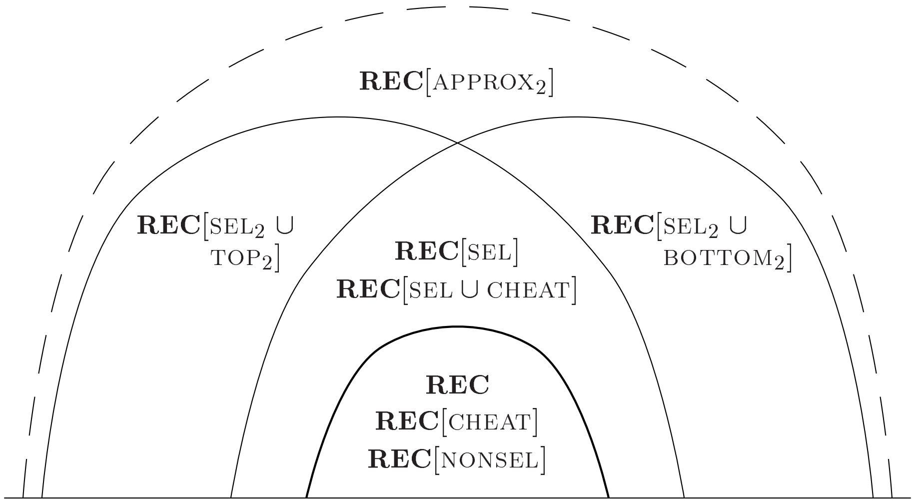


Figure 4-3
Class review for the results established in this chapter. The bordering line for APPROX$_2$ is dashed, because while we know that **REC**[APPROX$_2$] does not coincide with any of the other classes, it is unclear whether it is not perhaps simply the union of the lower classes.

Beigel's conjecture, also known as the *Cardinality Conjecture*, was first proved for the case $n = 2$ by Owings (1989). The general case was proved by Martin Kummer in 1992 in the paper *A Proof of Beigel's Cardinality Conjecture*. In the paper, Kummer already pointed out that the theorem can be strengthened in different ways. In essence, he showed that the theorem also holds for several families between $\text{CARD}_n(n)$ and the family NONSEL_n. However, lacking the framework of pools and families he did not to give the exact characterisation which is due to Nickelsen. In his 1997 paper, Nickelsen finally proved Fact 4.14 which states that **REC** = **REC**[NONSEL] and this theorem cannot be strengthened any further—at least not within the framework of pools and families.

Beyond selectivity, the picture is not so clear. The *Generalised Non-Speedup Theorem* was first proved in Beigel *et al.* (1992). Having a close look at the proof given there, Theorem 4.12 is a restating of the Generalised Non-Speedup Theorem in the framework of pools and families. However Theorem 4.16, which states **REC**[SEL] = **REC**[SEL ∪ CHEAT], shows that once more the formulation in the framework of pools and families is strictly more powerful than the standard formulation.

In the same paper of Beigel, Kummer and Stephan, decision procedures are presented for the inclusion and the equality problems of partial information classes over size families. They propose the following definition, where $g_P(k)$ for some pool P is the maximum size of selections of k positions from P:

> **Definition 4.1 from Beigel *et al.* (1992).** A nonempty set $V \subseteq \{0,1\}^s$ is called (m,n)-*good* iff for every partition $n_1 + n_2 + \cdots + n_k = n$ with $1 \le n_j \le s$ and $k \ge 1$, $g_V(n_1) + \cdots + g_V(n_k) \le m + k - 1$.

Their main Theorem 4.3 states that $\mathbf{REC}[\text{SIZE}_n(m)] \subseteq \mathbf{REC}[\text{SIZE}_{n'}(m')]$, iff every

(m,n)-good n'-pool has maximum size m'. As this is a purely combinatorial characterisation, inclusion of recursively computable partial information classes over size families is decidable. However, this does not identify the *stable* size families for there might exist an instable size family that collapses to some non-size family.

As the authors point out, (m,n)-goodness of a pool $P \subseteq \mathbb{B}^s$ is a strengthening of the assertion $g_P(n) \leq m$. Note, that this weaker assertion is equivalent to $P \in \lceil \text{SIZE}_n(m) \rceil_s$. Hence, the theorem is also a strengthened version of the fact that if $\lceil \text{SIZE}_n(m) \rceil_s \subseteq \text{SIZE}_s(r)$ we have $\mathbf{REC}[\text{SIZE}_n(m)] \subseteq \mathbf{REC}[\text{SIZE}_s(r)]$.

Another kind of families for which the inclusion problem has been studied are the frequency families. Recall that $\text{FREQ}_n(r)$ is the family consisting of all n-pools that are contained in closed balls of radius r. The study of these families—more precisely of their partial information classes—began as early as 1960. In McNaughton (1961) page 393, Myhill asked if $\mathbf{REC} = \mathbf{REC}[\text{FREQ}_n(r)]$ for small r. An affirmative answer to this was obtained by Trakhtenbrot (1963) who showed that for $r < n/2$ this is correct—which is now also quite easy to see by noting that selective pools have a diameter of n and cannot be contained in closed balls of diameter less than n. For a survey of results concerning the inclusion problem and the equality problem for $r \geq n/2$ please see Kummer and Stephan (1995).

Advice classes were introduced by Karp and Lipton (1980). The first results on the advice complexity of partial information classes are due to Ko (1983), who showed $\mathbf{P}[\text{SEL}] \subseteq \mathbf{P}/O(n^2)$. A result due to Hemaspaandra and Torenvliet (1996) states that for the p-selective languages $n+1$ bits of advice are necessary for any resource bound, in general, but that for $\mathbf{NP}$ this is also sufficient.

In his Master's thesis *Upper and Lower Bounds for Token Advice for Partial Information Classes*, Ronneburger shows that cheatable languages can be decided with a *constant* amount of advice using a $\Sigma_4\mathbf{P}$-machine. It is not known whether this can be improved to $\mathbf{NP}$ or even $\mathbf{P}$. Theorem 4.18 is currently the best result in this regard.

Part II

Truth-Table Closures of Partial Information Classes

§ 8 Forderungen wenn sie von Nutzen seyn sollen

Wenn aber auch die Rechen-Maschinen nicht das Denken, sondern blos das Gedächtnis erleichtern können, so kann dieser Vortheil dennoch manchmal sehr gros seyn und bei denjenigen Dank verdienen, welche sich öfters anhaltend dem unangenehmen Geschäft des Addirens, Multiplicirens oder Dividirens großer Zahlen unterziehen müssen. Könnte hierdurch noch dem öftern Irren vorgebaut, theils auch Zeit erspart werden, so wäre ihr Nutzen gewiß entschieden: daß dieses aber noch die wenigsten leisten wird die Folge lehren.

Eine Rechenmaschine, die nicht blos Spielwerk seyn soll, muß meines Erachtens folgende Eigenschaften haben.

1 Muß sie so einfach als möglich seyn. Eine Erfordernis, welche theils Dauer, theils Wohlfeile zum Grunde hat.

2 Muß sie untrüglich seyn, d.i. sie muß, wenn gehörig verfahren wird, nie abweichende Resultate geben.

3 Sie muß das Ausrechnen verwickelter und großer Exempel beträchtlich erleichtern.

4 Sie muß dem Arbeitenden Zeit ersparen, leicht zu stellen und leicht zu bewegen seyn; denn sonst würde ihr Nutzen sehr gering seyn, und ein Rechnungskenner würde lieber auf die gewöhnliche Art rechnen, als zu einer Maschine seine Zuflucht nehmen, welche ihm nur eine Schwierigkeit mit der andern vertauscht und endlich

5 keine besondere Anstrengung des Gedächtnisses erfordern.

— Johann Paul Bischoff, *Versuch einer Geschichte der Rechenmaschine*, Ansbach 1804

5

Introduction to Reductions

Der Inhalt eines Begriffes nimmt ab, wenn sein Umfang zunimmt;
wird dieser allumfassend, so muß der Inhalt ganz verloren gehen.

— Gottlob Frege, *Die Grundlagen der Arithmetik*, Seite 40, Breslau 1884

A problem like the halting problem is—in a very natural sense—more difficult than any problem solvable in polynomial time. Complexity classes describe the complexity of languages by asserting that languages *in* the complexity class are somehow simpler than the languages *outside*. However, this simple *yes or no* assertion is often too coarse to describe how much more difficult a language is than some other language. For example, the unsatisfiability and satisfiability problems are obviously closely related, the first being the complement of the latter. However, if **NP** is not closed under complement, the unsatisfiability problem is no element of **NP** while the satisfiability problem is.

A more fine-grained comparison of languages can be achieved by checking which *reductions* are possible between them, i. e., by checking how difficult it is to decide a language when another language is somehow 'known'. If a language is reducible to some other language with, say, five queries, but not with four or less queries, the number five *quantifies* the relative difficulty of the languages. Reductions form the backbone of *structural complexity analysis*.

A reduction of a language to another language is determined by four basic parameters, all of which can be varied independently, which results in a large number of possible reductions:

1. The *number* of times the other language is consulted can be restricted.
2. It can be restricted how the answers to the queries may be *used*.
3. The *time or space* consumed by the reduction computation can be restricted.
4. It can be specified whether *adaptive* or *non-adaptive* queries are used.

This chapter starts with a review of the most important reductions studied in the literature, namely many-one reductions, Turing reductions and positive reductions. The second section introduces *general truth-table reductions* and combinatorial descriptions thereof, called *evaluation types*. The third section shows how different forms of truth-table reduction studied in the literature fit into the framework. Finally, the last section proves the somewhat surprising result that *all bounded reductions, including bounded Turing reductions*, can be represented by appropriate evaluation types.

5.1 Review of Many-One, Turing and Positive Reductions

This section reviews three important reductions studied in the literature. The most restrictive reduction is the many-one reduction. The most general reduction is the Turing reduction. Positive reductions are an appropriate kind of reduction for analysing the stability of non-deterministic classes. Specifically, while **NP** is presumably not closed under Turing reductions, Selman (1982b) proved Theorem 5.2 below, which states that **NP** is closed under positive Turing reductions.

Many-One Reductions

A language L is *many-one* reducible to a language K, if there exists a Turing machine M such that a word w is in L, iff $M(w)$ is in K. An important special case are many-one reductions that need only logarithmic space. In this case, the four parameters of a reduction are instantiated as follows:

1 Just one question is asked.

2 The answer to the query must be returned 'as is'.

3 The space used is limited to logarithmic space.

4 For a single query, adaptiveness is irrelevant.

Logarithmic space many-one reductions are commonly used in completeness arguments, because they are one of the most restrictive forms of reductions possible. If L is many-one reducible to K and we can decide K in some satisfactory way, we can also decide L just as quickly. Hence, if a language is *complete with respect to logarithmic space many-one reductions* for some class of languages, there is no language in the class which is 'more difficult' than the given language. Table 5-1 on the facing page lists a selection of the many known examples of complete problems for classic complexity classes.

Although logarithmic space many-one reduction is a restrictive reduction, it is by no means the most restrictive possible. More restrictive reductions become necessary, if one wishes to study problems which *are complete for logarithmic space itself.* For example, the directed trees reachability problem is complete for logarithmic space with respect to *parallel logarithmic time*—for more details please confer Jones *et al.* (1976) and Cook and McKenzie (1987).

Turing Reductions

Turing reductions are quite the opposite of many-one reductions. A language L is Turing reducible to a language K, if there exists a Turing machine which de-

Table 5-1
Examples of problems complete with respect to logarithmic space many-one reductions. For many more examples, please refer to Papadimitriou (1994).

Complexity Class	Complete Problems
L	*all languages in* **L** *except for* $\emptyset$ *and* Σ^*
NL	reachability, 2-satisfiability
polyL	*none*
P	circuit value problem, monotone circuit value problem
NP	satisfiability, travelling salesman problem, Hamilton path, clique
PSPACE	quantified satisfiability, game Go
REC	*none*

cides L with the oracle K. For the important special case of a *polynomial time Turing reduction*, the four reduction parameters are instantiated as follows:

1 As many questions may be asked as desired.

2 The answers may be used in any desired way.

3 The used time is limited to polynomial time.

4 The questions may be asked adaptively.

While Turing reductions are less useful with respect to completeness arguments than many-one reductions, they are important with respect to closure properties. A class is closed under Turing reductions, if every language reducible to some language in the class is already an element of the class. Hence, being closed under Turing reductions means that the class is *stable*—even very general operations on languages in the class do not lead outside. A well-known example of a complexity class closed under polynomial time Turing reductions is **P**.

A most intriguing question is, is **NP** closed under Turing reductions? If this is not the case, then $\mathbf{P} \neq \mathbf{NP}$ as **P** is closed under Turing reductions. The other way round, if **NP** is closed under Turing reductions, then $\mathbf{NP} = \mathbf{coNP}$.

Positive Reductions

5.1 Definition (Positive Turing Machine)
An oracle Turing machine is *positive*, if, whenever a word is accepted for some oracle, it is also accepted for all larger oracles.

Phrased differently, an oracle Turing machine M is *positive*, if the mapping $\mathrm{L}(M, .)$ which maps oracles to accepted languages is monotone with respect to inclusion. The importance of positive reductions for polynomial time lies in the following fact, taken from Theorem 7 of Selman (1982b).

5.2 Theorem
The class $\mathbf{NP}$ *is closed under positive polynomial time Turing reductions.*

Proof. Assume that L is reducible via a positive Turing machine R to a language K and assume $K \in \mathbf{NP}$ via some machine M_K. We construct a non-deterministic Turing machine M_L which decides L and runs in polynomial time, hence proving $L \in \mathbf{NP}$.

For an input word w let M_L simulate the reduction machine R. Whenever the reduction machine produces a query q for its oracle, the simulation of R is interrupted and the machine M_L starts a non-deterministic simulation of M_K on input q. If a sub-simulation path answers 'yes', the simulation of R is continued with the assumption $q \in K$; if a sub-simulation path answers 'no', the simulation of R is also continued but with the assumption $q \notin K$. Note, that if, indeed, $q \in K$, there exists some path which resumes the simulation of R with the correct assumption $q \in K$; and if $q \notin K$, *all* paths resume the simulation of R with the correct assumption $q \notin K$. Hence, there always exists at least one simulation path for which the assumptions are correct.

As the correct assumptions are always made on some path, we must only argue that it is not possible that a word is *inadvertently* accepted, presumably because we make wrong assumptions on some path. But wrong assumptions can only be made for queries with $q \in K$. So, assume that M_L makes wrong assumptions on queries $q_1, \dots, q_k \in K$. If R does not accept the word, neither will it with the smaller oracle $K \backslash \{q_1, \dots, q_k\}$. But with respect to this oracle the assumptions on the path were correct—and hence this path rejects. □

5.2 Definition of Truth-Table Reductions

In this section, a general framework for the study of truth-table reductions is introduced. After an introduction to truth-table reductions in general, Definition 5.3 introduces a unified way of *describing truth-table reductions combinatorially*. General bounded truth-table reductions are defined in Definition 5.4. A detailed overview how notions of truth-table reducibility studied in the literature fit into the presented framework can be found in the next section.

For a truth-table reduction, all queries must be made non-adaptively. Consider once more two languages L and K, where the language K is assumed to be 'known' in some way; for example if K is the satisfiability problem, we might have a very expensive and fast computer for solving the problem like a DNA computer. If the language L is Turing reducible to K, then there exists a machine which produces a query to K, then produces another query based on the answer, then yet another, and so forth. Hence, a Turing reduction might be envisioned as a kind of *search* for information inside the language K.

It might be more useful to pose questions not sequentially, but rather *in parallel*. For example, we might have ten DNA computers available. Then, if the queries are posed in parallel, all ten answers can be computed in parallel which might save time. However, if we have to ask questions in parallel, we can no longer really search inside the language. Rather, what we do is a sophisticated *lookup* inside the language K. In general, a truth-table reduction of a language L to a language K works as follows:

1 Given an input word w, a function called the *generator* produces a tuple $q_1, \dots, q_k$ of queries to the language K.

2 A second function called the *evaluator* also gets the word w as input, *but also the characteristic string of the queries produced by the generator*. The evaluator, presumably using the extra information, must then decide whether $w \in L$ holds.

For each fixed input word, the behaviour of the evaluator *in terms of the characteristic bits* can be modelled by a Boolean function; hence the name *truth-table* reductions. The different ways the evaluator *uses* the characteristic bits can be described by the Boolean functions computed by the evaluator for different input words. This motivates the following definition, where Φ^k denotes the set of all k-ary Boolean functions.

5.3 Definition (Evaluation Type)
A *k-ary evaluation type* is a subset of Φ^k containing a non-constant function.

Based on this definition, we can now make precise the notion of truth-table reductions with a specific evaluation type. Definition 5.4 below extends the definition of Ladner *et al.* (1975) of truth-table reductions by allowing them to be parametrised over evaluation types.

5.4 Definition (Ψ-Reduction)
Let $\boldsymbol{FC}$ be c.c.c. and let Ψ be a k-ary evaluation type. A language L is *Ψ-reducible to a language K*, written $L \leq_{\Psi}^{\boldsymbol{C}} K$, if there exists a *generator* $g \in \boldsymbol{FC}$ and an *evaluator* $e \in \boldsymbol{FC}$ such that for all input words $w \in \Sigma^*$

1 we have $e_w \in \Psi$ where $e_w \colon \mathbb{B}^k \to \mathbb{B}$ is the Boolean function computed by the evaluator with its first input fixed to the word w, i. e., $e_w(b) := e\big(\langle w, b\rangle\big)$,

2 we have $\chi_L(w) = e_w\big(\chi_K(q_1), \dots, \chi_K(q_k)\big)$ where $q_i := \mathrm{p}_i\big(g(w)\big)$ denotes the *i-th query* produced by the generator for the input word.

From time to time, the evaluator will want to reject or accept some words without looking at the extra bits it is given. For example, the evaluator might *reject* some ill-formed words outright. If we wish to allow outright acceptance or rejection, the evaluation type must contain the *verum* and *falsum* functions. Fortunately, even if the evaluation type does not contain these functions, a small trick allows us to put them into the evaluation type without changing the reduction power.

5.5 Lemma
Let Ψ be an evaluation type and let K be a non-trivial language, i. e., $K \neq \emptyset, \Sigma^*$. Then for every language L we have $L \leq_{\Psi}^{\boldsymbol{C}} K$, iff $L \leq_{\Psi \cup \{\bot, \top\}}^{\boldsymbol{C}} K$.

Proof. We obviously only need to show that $L \leq^{\boldsymbol{C}}_{\Psi \cup \{\bot,\top\}} K$ via some g and e implies $L \leq^{\boldsymbol{C}}_{\Psi} K$ via some g' and e'. Let $u_0 \notin K$ and $u_1 \in K$ be words, which exist by assumption. By Definition 5.3 of evaluation types, there exists some non-constant Boolean function $\psi \in \Psi$. Let $\psi(b_0) = 0$ and $\psi(b_1) = 1$.

We construct the generator g' as follows: Upon input w, it checks, if e_w is *verum* or *falsum*. If not, it outputs $g(w)$. Otherwise, if $e_w = \bot$, the generator g' outputs $\langle u_{b_0\texttt{[1]}}, \dots, u_{b_0\texttt{[}k\texttt{]}} \rangle$; and if $e_w = \top$, it outputs $\langle u_{b_1\texttt{[1]}}, \dots, u_{b_1\texttt{[}k\texttt{]}} \rangle$. The evaluator e' does the same as the old evaluator e, except if $e_w = \bot$ or $e_w = \top$, where it uses the function ψ instead.

To see that g' and e' witness $L \leq^{\boldsymbol{C}}_{\Psi} K$, simply note that if e_w is *verum* or *falsum*, then $e'_w = \psi$ and the new evaluator is fed with the appropriate bitstring b_0 or b_1 by construction. □

For a language or a class of languages and a Ψ-reduction, one is often interested in the class of all languages which are reducible to one of the given languages. This class is denoted the *reduction closure* of the language or set of languages.

5.6 Definition (Reduction Closure)
The Ψ-*closure* of a class $\boldsymbol{K}$ of languages, written $\mathrm{R}^{\boldsymbol{C}}_{\Psi}(\boldsymbol{K})$, is the class of all languages Ψ-reducible to some language in $\boldsymbol{K}$.

Every evaluation type induces a special kind of truth-table reduction. Due to the large number of evaluation types, this may seem a bit alarming—the definition might be too general to allow our proving anything useful about evaluation types *in general*. Fortunately, this is not the case; rather the next chapter shows that *a decision procedure exists for deciding closure under arbitrary Ψ-reductions for partial information classes.*

While the above definitions are restricted to *fixed* numbers of queries produced by the generator, Definition 5.4 can easily be adapted to *non-constant* number of queries. The number of queries should then be some well-behaved function of the input length. For the polynomial case, Definition 5.7 gives a useful notion of 'well-behavedness' due to Krentel (1988), page 493.

In the following, we will nevertheless almost exclusively consider the case where the number of queries is fixed, *because only in this case a combinatorial theory is currently available*. The only exceptions are Theorems 7.8, 7.9 and 8.4 which are formulated explicitly for polynomial time.

5.7 Definition (Smooth Function)
A monotone function $s\colon \mathbb{N} \to \mathbb{N}$ is *smooth*, if the mapping $w \mapsto 1^{s(|w|)}$ is in $\mathbf{FP}$.

5.3 Representing Bounded Truth-Table Reductions by Evaluation Types

This section demonstrates that different notions of truth-table reducibility studied in the literature, see for example Ladner *et al.* (1975), can be described using appropriate evaluation types. Examples 5.8 to 5.11 show that the well-known forms of normal, positive, disjunctive and conjunctive truth-table reductions are Ψ-reductions for appropriate Ψ. Examples 5.12 and 5.13 demonstrate the same for two more specialised reductions, namely parity and cardinality reductions.

5.8 Example (Many-one reduction)
A many-one reduction can be modelled by the evaluation type $\Psi = \{\mathrm{id}\}$ containing only the unary identity function. In this case, the evaluator does nothing with the answer to the generator's query. Conversely, if a language L is Ψ-reducible to another language K, it is also many-one reducible to K via the generator—for the evaluator *must* use the identity function. ∎

5.9 Example (Bounded truth-table reduction)
If we do not restrict in any way how the evaluator may use the extra information, the evaluation type is given by Φ^k, the set of all k-ary Boolean functions. In the literature, $L \leq_{\Phi^k} K$ is usually denoted $L \leq_{k\text{-tt}} K$. ∎

5.10 Example (Positive bounded truth-table reduction)
In a positive reduction, the evaluator is required to behave like a positive machine. This means, that if the evaluator yields the result 1 when applied to the characteristic string of some words with respect to some oracle K, it must also yield 1 when fed with the characteristic string of the words with respect to *larger oracles*.

The evaluation type of a positive truth-table reduction is given by a set of *monotone* Boolean functions, for every monotone function ψ does indeed have the property that $b \leq_{\mathrm{pw}} b'$ implies $\psi(b) \leq \psi(b')$. If we let Δ^k denote the set of all monotone k-ary Boolean functions, i. e., $\Delta^k := \{\, \psi \mid \psi \colon \mathbb{B}^k \to \mathbb{B} \text{ monotone} \,\}$, the standard notation for $L \leq_{\Delta^k} K$ found in the literature is $L \leq_{k\text{-ptt}} K$. ∎

5.11 Example (Disjunctive and conjunctive truth-table reductions)
Two special cases of monotone functions are the logical and the logical or of k bits. These functions give rise to the sets $\Sigma^k = \{\vee_k\}$ and $\Pi^k = \{\wedge_k\}$. The standard notation for the disjunctive case is $L \leq_{k\text{-dtt}} K$ and for the conjunctive case $L \leq_{k\text{-ctt}} K$. ∎

The above examples treated the most prominent forms of truth-table reduction studied in the literature. However, evaluation types are just as useful for describing less commonly used reductions. The reductions presented in the following examples are taken from Agrawal *et al.* (1996).

5.12 Example (Parity reduction)
For parity reductions, the evaluator must compute either the k-ary parity function $\oplus_k$, which counts the number of 1's in its input modulo 2, or the negation of the parity function, or it may accept or reject the word outright. The corresponding evaluation type Ξ^k is the set $\{\oplus_k, \neg \circ \oplus_k, \top, \bot\}$.

The only action to be taken by the evaluator is to decide which function to use for a specific input word. Phrased differently, the evaluator must *decide the input word knowing only the parity of the queries.*

To appreciate the importance of this reduction, note that it follows from the results of Wechsung (1985) that *any language, which is k-truth-table reducible to the satisfiability problem, is already k-parity reducible to the satisfiability problem*. Phrased differently, Φ^k-reducibility to SAT implies Ξ^k-reducibility to SAT. We will show in Theorem 7.9 that *satisfiability shares this property with all p-selective languages*. ∎

5.13 Example (Cardinality reductions)
A reduction less restrictive than parity reduction is the cardinality reduction. While for the parity reduction the evaluator has to decide membership knowing only the *parity* of the number of words in the oracle language, for a cardinality reduction the evaluator knows the *cardinality itself*. Phrased differently, the evaluator *must* first apply a counting function to the characteristic string but may then do whatever it wants with the obtained number.

If we let $\#_1 \colon \mathbb{B}^k \to \mathbb{N}$ denote the function that counts the number of 1's in its input, the k-cardinality reduction is represented by the evaluation type $\{\, \eta \circ \#_1 \mid \eta \colon \mathbb{N} \to \mathbb{B} \,\}$. ∎

5.4 Representing Bounded Turing Reductions by Evaluation Types

This last section demonstrates that *bounded Turing reductions* can be modelled by appropriate evaluation types. This result is perhaps surprising as truth-table reductions and Turing reductions are generally considered to be incompatible notions.

First, we prove Lemma 5.14 below which shows how a Turing reduction with k queries can be simulated by a truth-table reduction with $2^k - 1$ queries. The converse of this lemma does not hold in general, but we show in Example 5.15 how this can be remedied by *shrinking the evaluation type*. As a generalisation of the ideas used in the example, Definitions 5.16 and 5.17 introduce the notions of *navigation paths* and *Turing evaluation types* Υ_k. Finally, Theorem 5.18 shows that Turing evaluation types model exactly Turing reducibility.

Consider a language L which is Turing reducible to a language K with a maximum of k queries via a machine R. In order to describe the Turing reduction using a truth-table reduction, we construct a generator as follows: On any given input, it simulates *all paths* which could possibly be taken by the reduction machine R. Along each path, k queries may be asked. In total there may be up to $2^k - 1$ many different queries. These are the queries generated by the generator.

The evaluator can now decide the input word by simply simulating R once more. However, whenever the oracle is queried by R the evaluator can lookup the correct value in its extra answer vector.

5.14 Lemma

A language which is k-Turing reducible to a language K is also $(2^k - 1)$-truth-table reducible to K.

The $(2^k - 1)$-truth-table reductions are strictly more powerful than k-Turing reductions in general. The reason is, in essence, that *the constructed evaluator does not use all $(2^k - 1)$-ary Boolean functions*. The following example demonstrates which functions are not used for the case $k = 2$.

5.15 Example (Turing reduction with two queries)

Consider for the case $k = 2$ two languages L and K with $L \leq_{2\text{-T}} K$ via R. For an input word w three questions are generated as described above, call them q_λ, q_0 and q_1. The query q_λ is the first query produced during the reduction. The query q_0 is produced, if the answer to q_λ was 'no', the query q_1 is produced, if the answer was 'yes'.

Let's trace the calculation of the evaluator. As input, the evaluator gets a word w and a bitstring $b =: b_\lambda b_0 b_1$ of length three. It starts a simulation of R and whenever the oracle is queried, the evaluator does a lookup in b. Now, assume $b_\lambda = 0$. Then the evaluator will next look at b_0 *and will not bother about b_1 any more*. Phrased differently, the construction of the evaluator guarantees $e_w(0, b_0, 0) = e_w(0, b_0, 1)$. Like things happen, if $b_\lambda = 1$. Hence, the evaluator will satisfy the following conditions for all words w:

$$e_w(0,0,0) = e_w(0,0,1), \quad e_w(0,1,0) = e_w(0,1,1),$$
$$e_w(1,0,0) = e_w(1,1,0), \quad e_w(1,0,1) = e_w(1,1,1).$$

The other way round, assume that a generator is given which produces three queries and an evaluator which uses only functions fulfilling the above equations. This reduction can be replaced by a 2-Turing reduction as follows: We start by querying the first word q_λ produced by the generator. If the answer is 'no', we ask q_0 and, getting the answer b_0, we output $e_w(0, b_0, ?)$, where the question mark denotes an arbitrary value—the value does not matter anyhow, by assumption. Likewise, if the answer to the first question was 'yes', we query q_1 and getting the answer b_1 we output $e_w(1, ?, b_1)$. In any case, this Turing reduction will give exactly the same outputs as the truth-table reduction. ▯

The idea of the previous example can be generalised to larger numbers of queries. The key observation is that the evaluator *navigates through* the answer vector. For any two answer vectors which are navigated in the same way, the evaluator must output the same value.

Assume that the answers to the queries are given in the order produced by a preorder traversal, i. e., first the answer to the root question, then all answers to the left subtree and then all answers to the right subtree. How does the evaluator navigate this bitstring? It will first look at the answer to the first query. If this bit is 0, the search continues inside the left subtree, i. e., inside the $2^{k-1}-1$ many bits following the first bit. If the bit is 1, the search continues inside the right subtree given by the $2^{k-1} - 1$ last bits.

5.16 Definition (Navigation Functions)
The *navigation functions* $\mathrm{nav}_k\colon \mathbb{B}^{2^k-1} \to \mathbb{B}^k$ are defined as follows:

$$\mathrm{nav}_k(b_1,\dots,b_{2^k-1}) := \begin{cases} \lambda & \text{if } k=0,\\ 0\ \mathrm{nav}_{k-1}\big(b_2,\dots,b_{2^k/2-1}\big) & \text{if } b_1=0,\\ 1\ \mathrm{nav}_{k-1}\big(b_{2^k/2},\dots,b_{2^k-1}\big) & \text{if } b_1=1.\end{cases}$$

With this definition, we can now define the evaluation type employed by the k-Turing reduction. Recall that the kernel of a function is the smallest equivalence relation on the function's domain such that non-equivalent elements are mapped to different values.

5.17 Definition (Turing Evaluation Type)
The k-Turing evaluation type Υ_k consists of all (2^k-1)-ary Boolean functions whose kernels include the kernel of nav_k, i. e., of all functions $\psi\colon \mathbb{B}^{2^k-1} \to \mathbb{B}^k$ with $\psi(b)=\psi(c)$ whenever $\mathrm{nav}_k(b)=\mathrm{nav}_k(c)$.

The following theorem, which states that Υ_k-reductions faithfully describe k-Turing reductions, is formulated for polynomial time only. The reason is, that it is not immediately clear how Turing reductions over arbitrary function classes should be defined. We need not elaborate further on this *for Υ_k-reductions can be used to define a notion of bounded Turing reducibility for arbitrary function classes—even for function classes not produced by Turing machines.* Hence, the theorem is just a justification that we may adopt Υ_k-reducibility as a general notion of bounded Turing reducibility.

5.18 Theorem
For all $k \geq 1$ and all languages L and K we have $L \leq^{\mathbf{P}}_{k\text{-T}} K$, iff $L \leq^{\mathbf{P}}_{\Upsilon_k} K$.

Proof. Assume $L \leq^{\mathbf{P}}_{k\text{-T}} K$ via a polynomial time bounded Turing machine R. We show $L \leq^{\mathbf{P}}_{\Upsilon_k} K$.

First, let's define the generator. Upon input w the first query produced by the generator is the first query produced by the reduction machine R on input w. Next, the generator produces $2^{k-1}-1$ many queries, which are all queries produced by the reduction machine R if the answer to its first query is 'no'. Next, the generator produces another $2^{k-1}-1$ many queries which are the queries produced if the first query is answered by 'yes'. Naturally, each of the sub-blocks of size $2^{k-1}-1$ is in turn is structured in this way.

The evaluator e, upon input w and the answers to the queries generated by the generator, must now decide the word w with respect to L. It does so by simulating R and consulting the answer vector whenever the reduction machine consults the oracle. We must only argue that for each input word w we have $e_w \in \Upsilon_k$. By definition, we must show that $\mathrm{nav}_k(b)=\mathrm{nav}_k(c)$ implies $e_w(b)=e_w(c)$. However, the lookups done by the evaluator for answers b are, in order, *exactly the bits of the navigation path* $\mathrm{nav}_k(b)$ *by construction.* Hence, $\mathrm{nav}_k(b)=\mathrm{nav}_k(c)$ implies $e_w(b)=e_w(c)$.

For the other direction, assume $L \leq^{\mathbf{P}}_{\Upsilon_k} K$ via a generator g and an evaluator e. We must show $L \leq^{\mathbf{P}}_{k\text{-T}} K$.

We construct the reduction machine as follows: It first asks the first query produced by the generator, yielding an answer b_1. The second answer b_2 is determined as follows: If $b_1 = 0$, the reduction continues recursively by asking the first of the queries $q_2, \dots, q_{2^k/2-1}$. If $b_1 = 1$, the reduction continues recursively by asking the first query of $q_{2^k/2}, \dots, q_{2^k-1}$. The recursive descend yields a sequence $b_1, \dots, b_k$ of answers.

The reduction machine has now obtained some k answers out of the needed $2^k - 1$ answers. Nevertheless, whatever the correct answers to the other queries might be, the navigation path of the the full answer vector is exactly $b_1 \dots b_k$. Hence, e_w applied to the correct answers yields the same value as e_w applied to *any bitstring with whose navigation path is* $b_1 \dots b_k$. But then, we can complete the reduction by returning exactly this value. □

Bibliographical Notes

For a general introduction to structural complexity theory, please see Balcázar *et al.* (1988, 1990). Polynomial time many-one reductions are also called Karp reductions in the literature. The usage of logarithmic space many-one reductions instead of polynomial time many-one reductions for completeness arguments is advocated in Papadimitriou (1994). Polynomial time Turing reductions are also called Cook reductions.

For a comparison of the relative power of the four basic forms of truth-table reductions, namely normal, positive, disjunctive and conjunctive truth-table reductions, see Ladner *et al.* (1975).

The split of truth-table reductions into a generation phase and an evaluation phase in Definition 5.4 is taken from Ladner *et al.* (1975). However, in their paper the reductions are not parameterised over evaluation types; instead the different types of truth-table reductions are defined later on by ad-hoc restrictions of the behaviour of the evaluator.

Reductions like parity and cardinality reductions have been studied in a number of papers which have been initiated by the surprising result of Wechsung and Wagner (1985) that k-truth-table reducibility to the satisfiability problem implies k-parity reducibility to this problem. Han and Thierauf (1995) extended this result by examining how *modulo* reductions can be strengthened. Note, that parity reduction corresponds to modulo 2 reduction.

A different way of modelling truth-table reductions has been introduced by Köbler and Thierauf (1994), who showed how *complexity restricted advice functions* can be used to model parity and modulo truth-table reductions. Unfortunately, their approach is not really suited as a general model of truth-table reductions because for instance positive reductions cannot be modelled.

6

Representation of Truth-Table Closures

Der Beweis hat eben nicht nur den Zweck,
die Wahrheit eines Satzes über jeden Zweifel zu erheben,
sondern auch den, eine Einsicht in die Abhängigkeit
der Wahrheiten von einander zu gewähren.

— Gottlob Frege, *Die Grundlagen der Arithmetik*, Seite 2, Breslau 1884

General bounded truth-table closures of partial information classes can be represented combinatorially. More precisely, for every notion of bounded truth-table reducibility and for any two families, it can be decided combinatorially whether the truth-table closure of the partial information class of the first family is *included* in the partial information class of the second family.

The first section introduces in Definition 6.1 the notion of *cones*, which form the combinatorial domain where the truth-table closures of partial information classes are represented. Given a evaluation type Ψ, a pool is called a Ψ-cone over a family $\mathcal{F}$, if the image of the pool under all products of functions from Ψ is an element of $\mathcal{F}$.

The central theorem of this chapter is the *Cone Theorem* for recursively presentable function classes, Corollary 6.8. It states that for any evaluation type Ψ we have $\mathrm{R}_{\Psi}^{\boldsymbol{C}}\big(\boldsymbol{C}[\mathcal{C}]\big) \subseteq \boldsymbol{C}[\mathcal{F}]$, iff all pools in $\mathcal{C}$ are Ψ-cones over the family $\mathcal{F}$. The Cone Theorem provides both a sufficient and necessary condition for inclusion of language classes *in terms of the combinatorial property of cones*.

Before the Cone Theorem is proved in the third section, the second section treats languages which are many-one complete for Ψ-reduction closures. We show in Lemma 6.6 that such many-one complete languages always exist and we also construct a *canonically many-one complete language*.

This chapter concludes with some immediate applications of the Cone Theorem in the fourth section. Specifically, all partial information class are closed under many-one reductions; exactly those partial information classes are closed under 1-tt-reductions, whose families are closed under bit-flip; and the selective languages are closed under bounded positive truth-table reductions. More sophisticated applications of the Cone Theorem are given in the next chapters, where the structure of cones is studied in detail.

6.1 Definition of Cones

How much partial information can be computed for the closure of a partial information class? To tackle this problem, we make some simple observations. For a language K, assume $K \in \boldsymbol{C}[\mathcal{C}]$ witnessed by some function $f \in \boldsymbol{FC}$. Next, assume that a language L is Ψ-reducible to K for some k-ary evaluation type Ψ, i. e., assume $L \leq_{\Psi}^{\boldsymbol{C}} K$ via a generator g and an evaluator e.

We now try to compute partial information *for the language* L for given words $w_1, \ldots, w_n$, knowing only $L \leq_{\Psi}^{\boldsymbol{C}} K$. Using the generator g, for the first word we can produce some k *queries* $q_1^1, \ldots, q_1^k$ with $q_1^i := \mathrm{p}_i\big(g(w_1)\big)$. Then by definition of a Ψ-reduction we have $\chi_L(w_1) = e_{w_1}\big(\chi_K(q_1^1), \ldots, \chi_K(q_1^k)\big)$. Likewise, we can compute queries for the other input words. This results in a total of $n{\cdot}k$ queries, see the left part of Figure 6-1 on the next page.

Observe, that *we can compute a pool from* $\mathcal{C}$ *with respect to the language* K *for these* nk *words*—provided that $\mathcal{C}$ has index nk, which we will tacitly assume in the following. Let Q be such a pool for the bitstring $\chi_K(q_1^1, \ldots, q_n^k)$. If we knew which bitstring in $b \in Q$ is the correct one, we could easily produce the correct characteristic bitstring for the original input words by applying e_{w_1} to the first n bits of b, applying e_{w_2} to the next n bits and so forth, yielding the bitstring $(e_{w_1} \times \cdots \times e_{w_n})(b)$. Although we do not know this correct bitstring, we know that it is in Q somewhere. Hence, *each bitstring in* Q *induces one possible characteristic bitstring for the original words* and the totality of all these possible characteristic bitstrings is a pool $P = (e_{w_1} \times \cdots \times e_{w_k})(Q)$ for the original words.

Naturally, all of this helps only, if $P \in \mathcal{F}$ for some family $\mathcal{F}$ for which we would like to have $L \in \boldsymbol{C}[\mathcal{F}]$. Now, if *for all pools* Q *in* $\mathcal{C}$ *the pool* $(\psi_1 \times \cdots \times \psi_n)(Q)$ *lies in* $\mathcal{F}$ *for all possible functions* $\psi_1, \ldots, \psi_n \in \Psi$, then we necessarily have $P \in \mathcal{F}$. This motivates the below definition of a *cone* and proves the *en suite* lemma.

6.1 Definition (Cone)
Let Ψ be a k-ary evaluation type. An nk-pool Q is *a* Ψ*-cone over an* n*-family* $\mathcal{F}$, if for all functions $\psi_1, \ldots, \psi_n \in \Psi$ we have $(\psi_1 \times \cdots \times \psi_n)(Q) \in \mathcal{F}$.

6.2 Lemma
Let $\boldsymbol{FC}$ *be c.c.c. and* Ψ *a* k*-ary evaluation type. Let* $\mathcal{F}$ *be an* n*-family and let all pools in an* nk*-family* $\mathcal{C}$ *be* Ψ*-cones over* $\mathcal{F}$. *Then* $\mathrm{R}_{\Psi}^{\boldsymbol{C}}\,(\boldsymbol{C}[\mathcal{C}]) \subseteq \boldsymbol{C}[\mathcal{F}]$.

For the situation in the lemma, we say $\mathcal{C}$ *is a family of* Ψ*-cones over* $\mathcal{F}$.

6.3 Example (Cones over size families)
Let Q be any nk-pool of size m, i. e., let Q contain exactly m bitstrings. Then the image of Q under *the product of any* n *Boolean functions trivially has a maximum of* m *bitstrings*, that is $|(\psi_1 \times \cdots \times \psi_n)(Q)| \leq |Q|$. Thus, Q is a cone over the family $\mathrm{SIZE}_n(m)$. ▯

6.4 Corollary
Let $\boldsymbol{FC}$ *be c.c.c. and* $\mathcal{C}$ *an* nk*-family and* Ψ *a* k*-ary evaluation type. Then* $\mathrm{R}_{\Psi}^{\boldsymbol{C}}\,(\boldsymbol{C}[\mathcal{C}]) \subseteq \boldsymbol{C}[\mathrm{SIZE}_n(\tau_{\mathcal{C}})]$.

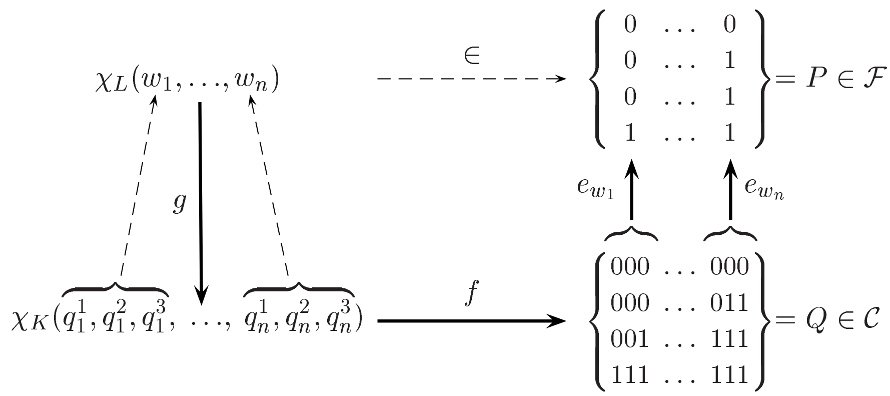


Figure 6-1
Situation of Lemma 6.2 for the case $k = 3$. For the input words w_i, the generator g produces the queries q_j^i. A function f can then produce a pool P for the queries. Each bitstring in P induces one possible $\chi_L(w_1,\ldots,w_n)$. In the figure, the function e_{w_1} is the logical and, whereas the function e_{w_n} is the logical or.

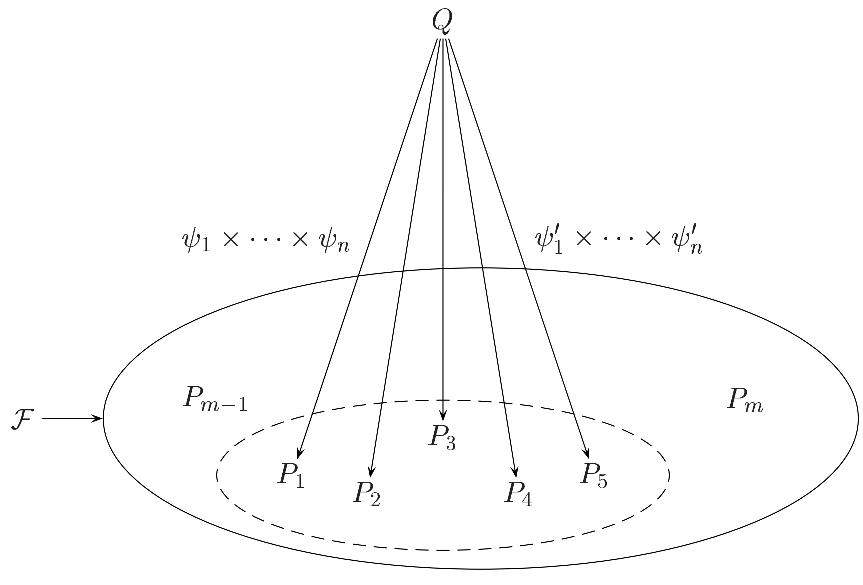


Figure 6-2
Motivation for the name 'cone'. The pool Q on top is called a Ψ-*cone over the family* $\mathcal{F} = \{P_1,\ldots,P_m\}$, iff for all functions $\psi_i \in \Psi$ the image of Q under their product lies within $\mathcal{F}$.

6.2 Definition of Canonically Complete Languages

One of the first things to ask about a complexity class is, does it have complete problems? This section tackles the problem of finding a many-one complete language K_Ψ for the reduction closure $\mathrm{R}^{\boldsymbol{C}}_{\Psi}(K)$ of a non-trivial language K. Definition 6.5 explicitly constructs such a language and Lemma 6.6 shows that it is, indeed, a many-one complete language and that the construction works for all c.c.c. function classes.

For every language $L \in \mathrm{R}^{\boldsymbol{C}}_{\Psi}(K)$ we would like to have $L \leq^{\boldsymbol{C}}_{\mathrm{m}} K_\Psi$. The language L being Ψ-reducible to K means, that for a word w we can generate queries q_1 to q_k by the generator and have a function $e_w \in \Psi$, such that the word w is in L, iff $e_w\big(\chi_K(q_1), \dots, \chi_K(q_k)\big) = 1$. This motivates the following definition:

6.5 Definition (Canonically Many-One Complete Language)
For a language K and a k-ary evaluation type Ψ the *canonically many-one complete language* is

$$K_\Psi := \big\{ \langle q_1, \dots, q_k, \psi\rangle \mid \psi\big(\chi_K(q_1), \dots, \chi_K(q_k)\big) = 1,\ \psi \in \Psi \big\}.$$

6.6 Lemma
Let $\boldsymbol{FC}$ be c.c.c. and Ψ a k-ary evaluation type. For all non-trivial languages K we have $\mathrm{R}^{\boldsymbol{C}}_{\Psi}(K) = \mathrm{R}^{\boldsymbol{C}}_{\mathrm{m}}(K_\Psi)$, i. e., the language K_Ψ is complete for $\mathrm{R}^{\boldsymbol{C}}_{\Psi}(K)$ with respect to many-one reductions.

Proof. As K is non-trivial, by Lemma 5.5 we may assume $\{\bot, \top\} \subseteq \Psi$.

First, we show that $L \leq^{\boldsymbol{C}}_{\Psi} K$ implies $L \leq^{\boldsymbol{C}}_{\mathrm{m}} K_\Psi$. Let L be Ψ-reducible to K via a generator $g \in \boldsymbol{FC}$ and an evaluator $e \in \boldsymbol{FC}$. We must show that there exists a function $f \in \boldsymbol{FC}$ such that $w \in L$, iff $f(w) \in K_\Psi$. By construction of the language K_Ψ, the function f must simply map an input word w to $\langle q_1, \dots, q_k, e_w\rangle$ where $q_i := \mathrm{p}_i\big(g(w)\big)$ are the queries produced by the generator.

Second, we must Ψ-reduce K_Ψ to K. The generator for this reduction, which simply extracts the queries, is even in $\mathbf{FL}$. The evaluator extracts the coding of the function stored in the input word. If the function is in Ψ, it applies this function to the answer vector; otherwise the word is rejected outright. Note, that this actually proves $K_\Psi \leq^{\mathbf{L}}_{\Psi} K$. □

6.3 Proof of the Cone Theorem

In this section two theorems are proved, namely Theorem 6.7 below and the Cone Theorem, Corollary 6.8. While the first theorem holds for all c.c.c. function classes, the Cone Theorem holds only for c.c.c. function classes which are recursively presentable.

6.7 Theorem
Let $\boldsymbol{FC}$ be c.c.c. and Ψ a k-ary evaluation type. Let $\boldsymbol{K}$ be a class of languages and $\mathcal{F}$ a normal n-family. Let $\mathcal{C}$ denote the nk-family of all Ψ-cones over $\mathcal{F}$. Then $\mathrm{R}^{\boldsymbol{C}}_{\Psi}\big(\boldsymbol{K}\big) \subseteq \boldsymbol{C}[\mathcal{F}]$ implies $\boldsymbol{K} \subseteq \boldsymbol{C}[\mathcal{C}]$.

Proof. Let $\mathrm{R}_{\Psi}^{\boldsymbol{C}}(\boldsymbol{K}) \subseteq \boldsymbol{C}[\mathcal{F}]$ and let $K \in \boldsymbol{K}$. We must show that for any nk words $w_1^1, \dots, w_n^k$ we can compute partial information from $\mathcal{C}$ with respect to the language K. We may assume that K is non-trivial.

Definition 6.5 introduced the *canonically many-one complete language K_Ψ*. For this language Lemma 6.6 states $K_\Psi \in \mathrm{R}_{\Psi}^{\boldsymbol{C}}(K)$ and by assumption $\mathrm{R}_{\Psi}^{\boldsymbol{C}}(K) \subseteq \boldsymbol{C}[\mathcal{F}]$. Hence, we have $K_\Psi \in \boldsymbol{C}[\mathcal{F}]$.

Consider arbitrary functions $\psi_1, \dots, \psi_n \in \Psi$. We construct n words $u_1, \dots, u_n$ by setting $u_i := \langle w_i^1, \dots, w_i^k, \psi_i \rangle$ for $i = 1, \dots, n$. As $K_\Psi \in \boldsymbol{C}[\mathcal{F}]$, for the n words $u_1, \dots, u_n$ we can compute a pool $P \in \mathcal{F}$ with respect to the language K_Ψ:

$$\chi_{K_\Psi}(u_1, \dots, u_n) = \chi_{K_\Psi}\big(\langle w_1^1, \dots, w_1^k, \psi_1\rangle, \dots, \langle w_n^1, \dots, w_n^k, \psi_n\rangle\big) \in P.$$

We define the pool $Q_{\psi_1,\dots,\psi_n} := \big\{ b \in \mathbb{B}^{nk} \mid (\psi_1 \times \dots \times \psi_n)(b) \in P \big\}$. The important point is, that *this is a pool for the original words and the language K*.

To see this, consider the pool P. Somewhere in the pool there is the bitstring $\chi_{K_\Psi}(u_1, \dots, u_n)$. By Definition 6.5, for each of the words u_i we have $\chi_{K_\Psi}(u_i) = \psi_i\big(\chi_L(w_i^1), \dots, \chi_L(w_i^k)\big)$. Hence, if we put all the characteristic values of the words u_i alongside, we get

$$\begin{aligned} \chi_{K_\Psi}(u_1, \dots, u_n) &= \psi_1\big(\chi_L(w_1^1), \dots, \chi_L(w_1^k)\big) \cdots \psi_n\big(\chi_L(w_n^1), \dots, \chi_L(w_n^k)\big) \\ &= (\psi_1 \times \dots \times \psi_n)\big(\chi_L(w_1^1, \dots, w_n^k)\big) \end{aligned}$$

Hence, the correct characteristic value b of the original input words has indeed the property $(\psi_1 \times \dots \times \psi_n)(b) \in P$.

The pool $Q_{\psi_1,\dots,\psi_n}$ is a pool for the input words, but typically it is not a Ψ-cone over $\mathcal{F}$. Although it *does* have the property $(\psi_1 \times \dots \times \psi_n)(Q_{\psi_1,\dots,\psi_n}) = P \in \mathcal{F}$, we need this property for *all* functions from Ψ—not just for the functions $\psi_1, \dots, \psi_n$ we happened to choose. Fortunately, this is easy to fix by employing once more the pool intersection trick. For each tuple of functions $\psi_1, \dots, \psi_n$ we compute a pool $Q_{\psi_1,\dots,\psi_n}$. Then, the *intersection Q* of these pools is still a pool for the input words and a Ψ-cone over $\mathcal{F}$. Indeed, for any functions $\psi_1, \dots, \psi_n$ we have $(\psi_1 \times \dots \times \psi_n)(Q) \subseteq (\psi_1 \times \dots \times \psi_n)(Q_{\psi_1,\dots,\psi_n}) \in \mathcal{F}$.

To see that the pool Q can produced by a function in $\boldsymbol{FC}$, simply note that there are only finitely many combinations of Boolean functions to be considered. □

6.8 Corollary (Cone Theorem)
Let $\boldsymbol{FC}$ be c.c.c. and recursively presentable, let $\mathcal{C}$ and $\mathcal{F}$ be respectively normal nk- and n-families. Then $\mathrm{R}_{\Psi}^{\boldsymbol{C}}\big(\boldsymbol{C}[\mathcal{C}]\big) \subseteq \boldsymbol{C}[\mathcal{F}]$, iff $\mathcal{C}$ is a family of Ψ-cones over $\mathcal{F}$.

Proof. If $\mathcal{C}$ is a family of Ψ-cones over $\mathcal{F}$, then Lemma 6.2 asserts $\mathrm{R}_{\Psi}^{\boldsymbol{C}}(\boldsymbol{C}[\mathcal{C}]) \subseteq \boldsymbol{C}[\mathcal{F}]$. Vice versa, if $\mathrm{R}_{\Psi}^{\boldsymbol{C}}(\boldsymbol{C}[\mathcal{C}]) \subseteq \boldsymbol{C}[\mathcal{F}]$, then Theorem 6.7 asserts that there exists a subset closed nk-family $\mathcal{C}'$ of cones over $\mathcal{F}$ with $\boldsymbol{C}[\mathcal{C}] \subseteq \boldsymbol{C}[\mathcal{C}']$. By the Unique Normal Form Theorem, this implies $\mathcal{C} \subseteq \mathcal{C}'$. But then, $\mathcal{C}$ contains only cones over $\mathcal{F}$. □

Corollary 6.8 allows us to check $\mathrm{R}_{\Psi}^{\boldsymbol{C}}(\boldsymbol{C}[\mathcal{C}]) \subseteq \boldsymbol{C}[\mathcal{F}]$ for given families purely combinatorially. Most importantly, we can check whether a partial information class is *closed* under some given truth-table reduction, simply by checking if the upward translated version of the family contains only cones over itself.

6.4 Checking Basic Closure Properties using Cones

Concluding this chapter, this section demonstrates how the Cone Theorem can be used to establish basic closure properties of partial information classes. We study in order many-one reductions, 1-tt-reductions and bounded positive truth-table reductions.

The standard proofs to be found in the literature for the results of this section argue directly in the domain of languages and reductions. Opposed to this, the below proofs are *of purely combinatorial nature*.

Nickelsen (1997) showed that all polynomial time partial information classes are closed under polynomial time many-one reductions. Recall that many-one reductions can be represented by the unary evaluation type $\Psi = \{\mathrm{id}\}$. When instantiated with this evaluation type, Lemma 6.2 states that $\boldsymbol{C}[\mathcal{F}]$ is closed under many-one reductions provided that all pools in $\mathcal{F}$ are Ψ-cones over $\mathcal{F}$. But because Ψ contains only the identity function, every pool in $\mathcal{F}$ is trivially a Ψ-cone over $\mathcal{F}$.

The question, whether partial information classes are closed under many-one reductions, boils down to the trivial question whether the identical image of all pools in a family lie within that family.

For the case where we may only ask distinct words, these claims are too strong. Here, the fitting reduction is a *one to one reduction*, which is a many-one reduction with the special property that no two different words are mapped to the same word. Putting it all together, we get the following theorem.

6.9 Theorem
Let $\boldsymbol{FC}$ be c.c.c. and let $\mathcal{F}$ be a family. Then $\boldsymbol{C}_{\mathrm{dist}}[\mathcal{F}]$ is closed under one to one reductions, and closed under many-one reductions, iff $\boldsymbol{C}_{\mathrm{dist}}[\mathcal{F}] = \boldsymbol{C}[\mathcal{F}]$.

Proof. As we just argued, $\boldsymbol{C}[\mathcal{F}]$ is closed under many-one reductions and an easy modification of Lemma 6.2 yields that $\boldsymbol{C}_{\mathrm{dist}}[\mathcal{F}]$ is closed under one to one reductions. So, we only need to show that if $\boldsymbol{C}_{\mathrm{dist}}[\mathcal{F}]$ is closed under many-one reductions, then $\boldsymbol{C}_{\mathrm{dist}}[\mathcal{F}] = \boldsymbol{C}[\mathcal{F}]$.

But if $\boldsymbol{C}_{\mathrm{dist}}[\mathcal{F}]$ is closed under many to one reductions, for a language $L \in \boldsymbol{C}_{\mathrm{dist}}[\mathcal{F}]$ the *tagged* language $L_{\mathrm{tag}} := \{w \mid \mathrm{p}_1(w) \in L\}$ is also an element of $\boldsymbol{C}_{\mathrm{dist}}[\mathcal{F}]$, because it is trivially many-one reducible to L via the projection p_1 which projects word tuples to their first component. But then, any n *non*-distinct words as input for L can be turned into n *distinct* words as input for L_{tag} by simply adding a unique tag to each input word. Hence, in this case we have $L \in \boldsymbol{C}[\mathcal{F}]$ and hence $\boldsymbol{C}[\mathcal{F}] = \boldsymbol{C}_{\mathrm{dist}}[\mathcal{F}]$. □

Nickelsen (1997) gave a characterisation of the classes $\mathbf{P}[\mathcal{F}]$ which are closed under truth-table reductions with a single query: A polynomial time partial information class $\mathbf{P}[\mathcal{F}]$ is closed under 1-tt-reductions, iff the normal family $\mathcal{F}$ is closed under 'bit-flip' of columns in pools. Using cones, we can easily prove Nickelsen's result combinatorially:

6.10 Theorem
Let $\boldsymbol{FC}$ be c.c.c. and recursively presentable, and let $\mathcal{F}$ be a normal family. Then $\boldsymbol{C}[\mathcal{F}]$ is closed under 1-tt-reductions, iff $\mathcal{F}$ is closed under bit-flip.

Proof. A truth-table reduction with a single query can be represented by the unary evaluation type $\Psi = \{\mathrm{id}, \neg, \bot, \top\}$. A pool $Q \in \mathcal{F}$ is a Ψ-cone over the normal family $\mathcal{F}$, iff every bit-flip of Q is an element of $\mathcal{F}$. Hence, the Cone Theorem specialises to the claim. □

To see the difference between many-one reductions and 1-tt-reductions, consider the selective family SEL_2 and $\boldsymbol{C} = \mathbf{P}$. All partial information classes are closed under many-one reductions, and hence also $\mathbf{P}[\mathrm{SEL}]$. However, the class $\mathbf{P}[\mathrm{SEL}]$ is *not closed* under 1-tt-reductions, simply because the bit-flip of the first position of the chain $\{00, 01, 11\}$ yields the top pool $\mathrm{B}_1(11)$, which is not a selective pool. However, the selective classes are closed under bounded positive truth-table reductions:

6.11 Theorem
Let $\boldsymbol{FC}$ be c.c.c. Then the class $\boldsymbol{C}[\mathrm{SEL}]$ is closed under bounded positive truth-table $\boldsymbol{FC}$-reductions.

Proof. We show that for each k the pools in $\lceil \mathrm{SEL}_2 \rceil_{2k}$ are Δ^k-cones over SEL_2. As $\lceil \mathrm{SEL}_2 \rceil_{2k} = \mathrm{SEL}_{2k}$, we only need to show *that the images of chains in $\mathbb{B}^{2k}$ under products of monotone functions are chains in $\mathbb{B}^2$*. But as a product of monotone functions is monotone itself, these products map chains to chains. This proves the claim. □

Actually, the above theorem does not tell the full story. Selman (1982a) proved that $\mathbf{P}[\mathrm{SEL}]$ is even closed under *unbounded* positive truth-table reductions. This result was improved by Buhrman, Torenvliet and van Emde Boas (1993) in their paper *Twenty questions to a p-selector*, where they proved that $\mathbf{P}[\mathrm{SEL}]$ is closed under positive Turing reductions.

Note however, that Theorem 6.11 holds for *all* c.c.c. function classes. Specifically, $\mathbf{L}[\mathrm{SEL}]$ is closed under logarithmic space bptt-reductions. It is not clear whether the stronger results of Selman and Buhrman *et al.* also hold for logarithmic space.

Bibliographical Notes

We first introduced the notion of *cones* in a research report, see Tantau *et al.* (1998), where we also proved the Cone Theorem for the polynomial case. Nickelsen uses a slightly restricted and simplified version of cones in his PhD thesis. He introduces

k-cones, which correspond to Φ^k-cones, and *positive k-cones*, which correspond to Δ^k-cones.

The notion of canonically complete languages is also introduced in the research report. Apart from its application in the proof of the Cone Theorem, these languages are also useful for the study of *cylinders*, see page 213 of Myhill (1959) for a motivation of the name. In Beigel *et al.* (1994) cylinders are introduced as follows:

> A set A is a *bd-cylinder* iff there is $f \in \mathbf{FP}$ such that $(\forall x, y)[x \in A \vee y \in A \Leftrightarrow f(x,y) \in A]$. A *bc-cylinder* is defined similarly: $\vee$ is replaced by $\wedge$. A set is a *bptt-cylinder* iff it is a bd-cylinder and a bc-cylinder. Finally, A is a *btt-cylinder* iff A is a pbtt-cylinder [sic] and $A \leq^p_m \bar{A}$.

Using the canonically complete languages and evaluation types, these definitions can also be expressed as follows: A language is a *Ψ-cylinder*, if $A_\Psi \leq^{\boldsymbol{C}}_{\mathrm{m}} A$. Then bd-, bc-, bptt- and btt-cylinders correspond to Σ^k-, Π^k-, Δ^k- and Φ^k-cylinders for all $k \geq 2$.

7

Structure of Selective, Bottom and Top Cones

Definitionen bewähren sich durch ihre Fruchtbarkeit.
Solche, die ebensogut wegbleiben könnten,
ohne eine Lücke in der Beweisführung zu öffnen,
sind als völlig wertlos zu verwerfen.

— Gottlob Frege, *Die Grundlagen der Arithmetik*, Seite 81, Breslau 1884

Cones are not only useful for *proving* closure properties of partial information classes, but also for establishing *strict hierarchies of reduction closures* of partial information classes. This chapter *gives a combinatorial proof that all bounded truth-table closures of the selective classes differ*. More precisely, we show that in the resource bounded case for all k there exists a language Φ^k-reducible to a language in $\boldsymbol{C}[\text{SEL}]$, but not Φ^{k-1}-reducible to any language in $\boldsymbol{C}[\text{SEL}]$. For the polynomial case, this was first proved in Theorem 1.2 of Hemaspaandra *et al.* (1996). In this chapter we also show how this result can be modified to establish strict hierarchies of reduction closures of bottom and top partial information.

The proofs of this chapter are elaborations of the following, surprisingly simple argument, which shows that the 1-tt- and the 2-tt-closures of $\mathbf{P}[\text{SEL}]$ differ: The 1-tt-reduction closure of $\mathbf{P}[\text{SEL}]$ is 2-approximable, as Corollary 6.4 tells us this is the case provided $\tau_{\text{SEL}_2} \leq 3$ which is indeed true. However, *the* 2*-tt-closure is not* 2*-approximable*. To see this, note that by the Cone Theorem we only need to show that *some selective* 4*-pools are no cones over* APPROX_2, and indeed, the image of the selective pool $\{0000, 0001, 0101, 0111, 1111\}$ under $\oplus_2 \times \oplus_2$ is exactly $\{00, 01, 11, 10, 00\} = \mathbb{B}^2 \notin \text{APPROX}_2$.

The basic idea of the general proof is to construct exact representations of the *amount of partial information computable for the k-tt-closure of the selective languages* and then to *prove that these representations differ*.

The first section introduces these representations, namely *walks with bounded change numbers*. The second section proves that this notion does, indeed, represent the reduction closures of the selective languages.

Based on these representations, the third section proves that the k-tt-reduction closures of the selective languages differ *by showing that their representations differ*.

The fourth section is a small excursion which shows that we can often *shrink the evaluation types* used in reductions to selective languages. Specifically, we show that k-tt-reductions to a selective language can always be replaced by k-parity reductions to that language. Furthermore, we show that $(2^k - 1)$-tt-reductions to a selective language can be replaced by k-Turing reductions to that language. These results show that *also the bounded parity and bounded Turing closures of the selective languages all differ*.

In the fifth and sixth section, we translate the established results to the bottom classes. For them, we can even establish strict hierarchies of the *normal*, *positive* and *disjunctive* truth-table closures.

7.1 Definition of Walks and Change Numbers

A walk on the hypercube $\mathbb{B}^n$ is a sequence of bitstrings of length n such that any two consecutive bitstrings differ at exactly one position. As shall be shown in the next section, the images of selective pools under tuples of Boolean functions are exactly walks with bounded change numbers.

7.1 Definition (Walk, Transition Sequence)
An *n-walk* is a sequence $b_1, \dots, b_m \in \mathbb{B}^n$ such that the Hamming distance between any two consecutive bitstrings is exactly one.

The *transition sequence* of a walk is the list of bit positions where consecutive bitstrings of the walk differ.

7.2 Definition (Change Number)
The *change number* of a position in a walk is the number of appearances of the position in the corresponding transition sequence.

Note that a walk may 'visit' a bitstring more than once. The names *transition sequence* and *change number* are taken from Reingold *et al.* (1977) and Gilbert (1958), respectively.

7.3 Example (Maximal selective pool)
Consider a maximal pool P in SEL_n, i. e., a maximal chain in $\mathbb{B}^n$. If we sort the bitstrings in P according to their number of 1's, they form a walk. The transition sequence is a permutation of the numbers in $\mathbb{N}_n^*$. The change number of each position is exactly 1, as each position gets bit-flipped exactly once. ▯

7.4 Example (Binary reflected Gray code)
A Gray *n-code* is a *self-avoiding* walk of length 2^n. The most important example is the *binary reflected Gray code*, called thus because if we depict the code in a table, the lower part is a reflection of the upper part disregarding the first column. Table 7-1 on the following page shows the binary reflected Gray code for the case $n = 4$. Note, that the Gray code has the property that it touches each vertex of the hypercube exactly once. ▯

7.5 Example (Symmetric Gray code)
A Gray code is *symmetric*, if the change numbers of all positions differ by no more than 1. Gilbert (1958) discovered a symmetric Gray 4-code, shown in Table 7-1. According to Faloutsos (1988) no systematic way is known to obtain symmetric Gray codes for $n > 4$. □

Table 7-1
Binary reflected Gray code for $n = 4$ and a symmetric Gray code discovered by Gilbert.

Gray code	Transition sequence	Symmetric Gray code	Transition sequence
0000	4	0000	4
0001	3	0001	3
0011	4	0011	4
0010	2	0010	2
0110	4	0110	1
0111	3	1110	4
0101	4	1111	1
0100	1	0111	3
1100	4	0101	1
1101	3	1101	2
1111	4	1001	3
1110	2	1011	4
1010	4	1010	3
1011	3	1000	2
1001	4	1100	1
1000	(1)	0100	(2)

7.2 Characterising the Cone Property for Selective Families

This section shows that walks with change numbers at most k *represent* the k-truth-table closures of the selective languages.

7.6 Theorem
The pools in SEL_{nk} *are all* Φ^k*-cones over a normal* n*-family* $\mathcal{F}$*, iff* $\mathcal{F}$ *contains all walks with change numbers at most* k.

Proof. First, assume that $\mathcal{F}$ contains all walks with change numbers at most k. We must show that then all selective nk-pools are all Φ^k-cones over $\mathcal{F}$, i. e., we must show that the image of any chain in $\mathbb{B}^{nk}$ under functions from Φ^k is contained in a walk with change numbers at most k.

Let Q be a chain in $\mathbb{B}^{nk}$ and let $\psi_i \in \Phi^k$ be any Boolean functions. First, we extend Q to a maximal chain Q'. Then $Q' = \{c_1, \dots, c_{nk+1}\}$ with $c_i \leq_{\mathrm{pw}} c_{i+1}$. This means, that starting from $c_1 = 0^{nk}$ each c_i has exactly one 1 more than the previous bitstring, ending with $c_{nk+1} = 1^{nk}$.

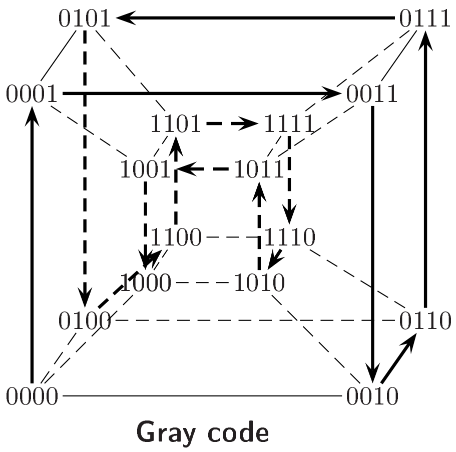


Figure 7-2
Different walks through the Hamming space $\mathbb{B}^4$. The left walk is the binary reflected Gray code. Although it touches all vertices exactly once, its maximal change number 2^{n-1} is rather high. The symmetric Gray code shown on the right also touches all vertices, but its maximal change number is 4 opposed to 8 for the binary reflected Gray code.

Consider the bitstrings $b'_i := (\psi_1 \times \cdots \times \psi_n)(c_i) \in \mathbb{B}^n$. We claim that this is (nearly) an n-walk with change numbers at most k. The Hamming distance between any two consecutive elements of the sequence is at most one, as from c_i to c_{i+1} exactly one position changes, so in the image also at most one position may change. Hence, it makes sense to talk about the change numbers of the sequence. The change numbers of the sequence are bounded by k, because for any position j a bit-flip between bitstrings b'_i and b'_{i+1} at position j can only occur if the bitstrings c_i and c_{i+1} differ at one of the positions $k(j-1)+1$ to kj.

As the c_i form a chain in $\mathbb{B}^{nk}$, so do the selection of the positions $k(j-1)+1$ through to kj form a chain in $\mathbb{B}^k$. But such a chain can have no more than $k+1$ elements. Hence, there are only $k+1$ different indices i such that c_i and c_{i+1} differ at some position between $k(j-1)+1$ and kj. And hence there can be no more than k bit-flips in the image. This shows that the change numbers of the sequence b'_i are at most k.

In order to turn the sequence $b'_1, \dots, b'_{nk+1}$ into a *walk* we simply remove consecutive duplicates, yielding the walk $b_1, \dots, b_m$ with change numbers at most k. By assumption, this walk is an element of $\mathcal{F}$. Hence, the image of the original pool Q under the functions ψ_i is contained in $\mathcal{F}$.

For the other direction, assume that all chains in $\mathbb{B}^{nk}$ are Φ^k-cones over $\mathcal{F}$. Given a walk $b_1, \dots, b_m$ with change numbers at most k, we must show $P = \{b_1, \dots, b_m\} \in \mathcal{F}$. To do so, we show that P is the image of some chain Q in $\mathbb{B}^{nk}$ under some Boolean functions. For now, we will also assume $b_1 = 0^n$.

First, we construct the Boolean functions. These are all identical and given by the parity function $\oplus_k$. Next, we construct a sequence $c_1, \dots, c_m$ with $c_i \in \mathbb{B}^{nk}$, such that we have $b_i = (\oplus_k \times \cdots \times \oplus_k)(c_i)$ and $Q = \{c_1, \dots, c_m\}$. We set $c_1 := 0^n$. Note, that then $b_1 = 0^n$. Having constructed c_i, we construct c_{i+1} as follows: From b_i to b_{i+1} exactly one position j changes its value. Then from c_i to c_{i+1} we change one 0 to a 1 among the 0's between positions $k(j-1)+1$ and kj. This ensures that the image of b_{i+1} will differ exactly at position j from b_i. Naturally, it is only possible to define c_{i+1} in this way, if there still *exists at least one* 0 *at the positions* $k(j-1)+1$ *to* kj. But fortunately this is necessarily the case, as we assumed the sequence b_i to have change numbers at most k and hence we will not run out of 0's.

The construction ensures that the image of Q is P and hence $P \in \mathcal{F}$. It remains to show that the sequence $b_1, \dots, b_m$ may also start with a bitstring different from 0^n. But if this is the case, for all positions j with $b_1[j] = 1$, we simply replace the parity function ψ_j with the inverted parity function. This completes the argument. ☐

7.3 Separating the Closures of Selective Classes

In this section, we finally prove that the k-tt-closure and the $(k+1)$-tt-closure of selective partial information differ for all k in the resource bounded case. For polynomial time, this result is due to Hemaspaandra *et al.* (1996) who used a direct diagonalisation argument.

7.7 Theorem
Let $\boldsymbol{FC}$ *be c.c.c. and recursively presentable. Then*

$$\boldsymbol{C}[\text{SEL}] \subsetneq \mathrm{R}^{\boldsymbol{C}}_{1\text{-tt}}\big(\boldsymbol{C}[\text{SEL}]\big) \subsetneq \mathrm{R}^{\boldsymbol{C}}_{2\text{-tt}}\big(\boldsymbol{C}[\text{SEL}]\big) \subsetneq \mathrm{R}^{\boldsymbol{C}}_{3\text{-tt}}\big(\boldsymbol{C}[\text{SEL}]\big) \subsetneq \dots .$$

Proof. We show $\mathrm{R}^{\boldsymbol{C}}_{k\text{-tt}}\big(\boldsymbol{C}[\text{SEL}]\big) \subsetneq \mathrm{R}^{\boldsymbol{C}}_{(k+1)\text{-tt}}\big(\boldsymbol{C}[\text{SEL}]\big)$. Let k be arbitrary.

First, we pick some n such that the lattice of n-families is fine-grained enough to distinguish the different truth-table closures. It turns out, that it suffices to have $nk + 1 < 2^n$.

Next, consider the family SEL_{nk}. All pools in this family have a maximum size of $nk+1$. Corollary 6.4 tells us that SEL_{nk} is then a family of Φ^k-cones over $\text{SIZE}_n(nk+1)$. Then, if m is the smallest number such that SEL_{nk} is a family of cones over $\text{SIZE}_n(m)$, we have $m \le nk+1 < 2^n$. By Corollary 6.8, we have $\mathrm{R}^{\boldsymbol{C}}_{k\text{-tt}}\big(\boldsymbol{C}[\text{SEL}]\big) \subseteq \boldsymbol{C}[\text{SIZE}_n(m)]$.

To finish the proof, we only need to show that the truth-table closure of $\boldsymbol{C}[\text{SEL}]$ with $k+1$ queries is *not contained in* $\boldsymbol{C}[\text{SIZE}_n(m)]$. But for this in turn, by Theorem 7.6 we only need to find an n-walk with change numbers at most $k+1$ which visits more than m bitstrings.

As m is minimal such that SEL_{nk} is a family of cones over $\text{SIZE}_n(m)$, by Theorem 7.6 there exists a walk $b_1, \dots, b_{m'}$ with change numbers at most k such that $|\{b_1, \dots, b_{m'}\}| = m$. Note, that we might have $m' > m$ as some bitstrings might be visited more than once.

As $m < 2^n$ there exists at least one bitstring b which does not occur in the sequence $b_1, \dots, b_{m'}$. We *extend* the sequence $b_1, \dots, b_{m'}$ by bitstrings $b_{m'+1}, b_{m'+2}, \dots$ in the following way: Starting from $b_{m'}$ we flip the first position where $b_{m'}$ and b differ, obtaining $b_{m'+1}$. Then in $b_{m'+2}$ we flip the next position where they differ and so forth. After exactly $\mathrm{d}(b_{m'}, b)$ steps, we will have reached b. The important point is that we still have a walk by construction, *and its change numbers are at most* $k+1$ as every position gets flipped at most once more. Hence, we have found the desired walk. □

One can rephrase the main idea of the proof just given a little less technical as follows: Let $b_1, \dots, b_m$ be a walk with change numbers at most k touching as many vertices in $\mathbb{B}^n$ as possible. Then, as long as there still exists at least one vertex not visited, *we can find a walk with change numbers at most* $k+1$ *going through more vertices that* $b_1, \dots, b_m$, simply by extending the path's end b_m to the remaining vertex b directly. This extension raises the maximal change number by at most one.

7.4 Shrinking the Evaluation Type of Reductions to Selective Languages

This section proves two theorems, which state that if a language is Φ^k-reducible to a selective language, it is also reducible to this languages via *two smaller evaluation types*. As an application of these results, we show that not only the truth-table closures of the selective languages all differ, but also the Turing closures as well as the parity closures.

The following two theorems are formulated for polynomial time, because they then hold even for non-constant numbers of queries used in the reduction. If the number of queries is constant, they hold for arbitrary c.c.c. function classes.

For the following theorem, recall from Definition 5.7 that a function $s\colon \mathbb{N} \to \mathbb{N}$ is *smooth*, if $w \mapsto 1^{s(|w|)}$ is in $\mathbf{FP}$. The theorem is also stated in Hemaspaandra *et al.* (1996) for fixed $k = k(n)$, although the proof seems a little hard to spot in their paper.

7.8 Theorem
Let $2^{k(n)}$ be a smooth function and K a p-selective language. Then a language L is $\big(2^{k(n)} - 1\big)$-truth-table reducible K, iff L is $k(n)$-Turing reducible to K.

Proof. As the reverse implication has given by Lemma 5.14 and the fact that $2^{k(n)}$ is smooth, assume that L is $\big(2^{k(n)} - 1\big)$-truth-table reducible to K. We wish to construct a Turing reduction from L to K.

Upon input w compute the queries $q_1, \dots, q_{2^{k(n)}-1}$ produced by the generator of the truth-table reduction. Using the selectivity of K, compute a pool for the queries and then compute a permutation σ such that $\chi_L(q_{\sigma_i}) \leq \chi_L(q_{\sigma_{i+1}})$. For the sorted words, we use a binary search to find the first query which is an element of K. Knowing this word, we know the characteristic string for *all* queries and can hence use the evaluator to compute $\chi_L(w)$. □

7.9 Theorem

Let s be a smooth function and K a p-selective language. Then a language L is $s(n)$-truth-table reducible to K, iff L is $s(n)$-parity reducible to K.

Proof. We only need to prove the first direction. Assume that L is $s(n)$-truth-table reducible to K via a generator g and an evaluator e. We must show that L is also $s(n)$-truth-table reducible to K via a generator g' and an evaluator e' which may only use the parity or the negated parity function.

Upon input w the new generator g' first computes the queries $q_1, \dots, q_{s(n)}$ produced by g upon input w. Using the selectivity of K, it computes a maximal selective pool for these words. Let $\{c_1, \dots, c_{s(n)+1}\}$ be this pool with $c_i \leq_{\mathrm{pw}} c_{i+1}$ and let $t_1, \dots, t_{s(n)}$ be its transition sequence.

The new generator, starting with c_1, iterates until it finds an index i such that $e_w(c_i) \neq e_w(c_{i+1})$. The first word generated by the new generator is then q_{t_i}. Next, the search is continued until a new index i' is found with $e_w(c_{i'}) \neq e_w(c_{i'+1})$. The second word produced is $q_{t_{i'}}$. In this fashion, the search continues and queries are produced until no more bitstrings are left. If less than $s(n)$ many queries are produced, the remaining queries are filled up with a dummy word known to be not an element of K.

We claim, that the new generator must simply use the parity function if $e_w(c_1) = 0$ and the negated parity function if $e_w(c_1) = 1$. To see this, note that if none of the queried words is in K, then the answer of the new evaluator is, indeed, correct. If exactly one of the queries in in K, it must be the first query. But then, the answer is also correct by construction. If exactly two queries are in K, it must be the first two queries and again the answer is correct; and so forth. This shows that the new evaluator correctly computes $\chi_L(w)$. $\square$

7.10 Corollary

Let $\boldsymbol{FC}$ be c.c.c. and recursively presentable. Then all k-Turing and all k-parity closures of $\boldsymbol{C}[\mathrm{SEL}]$ differ, i. e.,

$$\begin{aligned}
&\boldsymbol{C}[\mathrm{SEL}] \subsetneq \mathrm{R}^{\boldsymbol{C}}_{\Upsilon_1}\bigl(\boldsymbol{C}[\mathrm{SEL}]\bigr) \subsetneq \mathrm{R}^{\boldsymbol{C}}_{\Upsilon_2}\bigl(\boldsymbol{C}[\mathrm{SEL}]\bigr) \subsetneq \mathrm{R}^{\boldsymbol{C}}_{\Upsilon_3}\bigl(\boldsymbol{C}[\mathrm{SEL}]\bigr) \subsetneq \dots\ , \\
&\boldsymbol{C}[\mathrm{SEL}] \subsetneq \mathrm{R}^{\boldsymbol{C}}_{\Xi^1}\bigl(\boldsymbol{C}[\mathrm{SEL}]\bigr) \subsetneq \mathrm{R}^{\boldsymbol{C}}_{\Xi^2}\bigl(\boldsymbol{C}[\mathrm{SEL}]\bigr) \subsetneq \mathrm{R}^{\boldsymbol{C}}_{\Xi^3}\bigl(\boldsymbol{C}[\mathrm{SEL}]\bigr) \subsetneq \dots\ .
\end{aligned}$$

7.5 Separating the Closures of Bottom and Top Classes

This section shows that the combinatorial arguments used to establish that all k-tt-closures of the selective languages differ can be modified to establish a new result, namely that all k-tt-closures of the top and bottom classes differ. As the top classes are just the complements of the bottom classes, the arguments will be restricted to the bottom case. Recall, that $\mathrm{BOTTOM}_n = \langle \mathrm{B}_1(0^n) \rangle$ and that $\lceil \mathrm{BOTTOM}_2 \rceil_n = \mathrm{BOTTOM}_n$.

7.11 Theorem

Let $\boldsymbol{FC}$ be c.c.c. and recursively presentable. Then

$$\boldsymbol{C}[\text{BOTTOM}] \subsetneq \mathrm{R}^{\boldsymbol{C}}_{1\text{-tt}}(\boldsymbol{C}[\text{BOTTOM}]) \subsetneq \mathrm{R}^{\boldsymbol{C}}_{2\text{-tt}}(\boldsymbol{C}[\text{BOTTOM}]) \subsetneq \dots .$$

Proof. We prove $\mathrm{R}^{\boldsymbol{C}}_{k\text{-tt}}(\boldsymbol{C}[\text{BOTTOM}]) \subsetneq \mathrm{R}^{\boldsymbol{C}}_{(k+1)\text{-tt}}(\boldsymbol{C}[\text{BOTTOM}])$.

First, we must pick an n large enough such that the lattice of n-families is fine-grained enough to distinguish the k-tt- and the $(k+1)$-tt-closures of $\boldsymbol{C}[\text{BOTTOM}]$. Again, as in Theorem 7.7 where we proved the like claim for selective partial information, it suffices to have $nk + 1 < 2^n$.

Consider the family BOTTOM_{nk}. We have $\tau_{\text{BOTTOM}_{nk}} = nk + 1$. Hence, this is a family of Φ_k-cones over the family $\text{SIZE}_n(nk + 1)$, as Corollary 6.4 states exactly that. Let m be minimal such that the pools in BOTTOM_{nk} are cones over $\text{SIZE}_n(m)$. Then we have $m \le nk + 1 < 2^n$.

Once more, we now have the situation that the k-tt-closure of $\boldsymbol{C}[\text{BOTTOM}]$ is contained in $\boldsymbol{C}[\text{SIZE}_n(m)]$ and it suffices to prove that the $(k+1)$-tt-closure is not. By the Cone Theorem, it suffices to find a pool $Q' \in \text{BOTTOM}_{n(k+1)}$ and $(k+1)$-ary Boolean functions $\psi'_1, \dots, \psi'_n$ such that the image of Q' under their product has size $m + 1$.

As m is minimal such that all pools in BOTTOM_{nk} are Φ_k-cones over $\text{SIZE}_n(m)$, there must exist an n-pool P of size m and k-ary Boolean functions $\psi_1, \dots, \psi_n$ and finally a maximal pool $Q \in \text{BOTTOM}_{nk}$ such that $P = (\psi_1 \times \dots \times \psi_n)(Q)$.

First, we 'argue away' any constant columns of 1's in Q. If Q has such a column, then the pool where we replace this column with a column of 0's has the same size. Furthermore, by redefining the functions ψ_i appropriately it has the same image as Q.

The basic idea of how to define the desired $(nk + n)$-pool Q' is shown in Figure 7-3 on the following page. First, after every block of k positions we add a column of 0's. Then, we add one further bitstring which is 0 for all old positions and 1 for the positions where we inserted the columns of 0's.

We claim that the pool Q' obtained in this way from Q is indeed an element of BOTTOM_{nk+n}. Consider the eigenpool of Q'. Recall that the eigenpool was defined as the pool where we remove all constant columns and for duplicate columns all but the first occurrence. As $Q \in \langle \mathrm{B}_1(0^n) \rangle$ its eigenpool is contained in a closed ball of radius 1 around 0^p for some appropriate p.

The eigenpool of Q' is very similar to the eigenpool of Q. We simply add a new column which is 0 but for the last position where it is 1. This last position is part of a new bitstring which is also constantly 0 but for the new column. But then, the eigenpool is contained in a closed ball of radius 1 around 0^{p+1}. This shows that Q' is in BOTTOM_{nk+n}.

Now that we know that Q' is element of BOTTOM_{nk+n} it is quite easy to define functions $\psi'_1, \dots, \psi'_n$ such that the image of Q' under their product is larger than the image of Q under the product of $\psi_1, \dots, \psi_n$.

$$Q = \underbrace{\left\{\begin{array}{c} {*}{*}{*}{*}{*} \;\ldots\; {*}{*}{*}{*}{*} \\ {*}{*}{*}{*}{*} \;\ldots\; {*}{*}{*}{*}{*} \\ {*}{*}{*}{*}{*} \;\ldots\; {*}{*}{*}{*}{*} \\ {*}{*}{*}{*}{*} \;\ldots\; {*}{*}{*}{*}{*} \\ {*}{*}{*}{*}{*} \;\ldots\; {*}{*}{*}{*}{*} \\ {*}{*}{*}{*}{*} \;\ldots\; {*}{*}{*}{*}{*} \\ \underbrace{{*}{*}{*}{*}{*}}_{k} \;\ldots\; \underbrace{{*}{*}{*}{*}{*}}_{k} \end{array}\right\}}_{nk} \qquad Q' = \underbrace{\left\{\begin{array}{c} {*}{*}{*}{*}{*}\mathbf{0} \;\ldots\; {*}{*}{*}{*}{*}\mathbf{0} \\ {*}{*}{*}{*}{*}\mathbf{0} \;\ldots\; {*}{*}{*}{*}{*}\mathbf{0} \\ {*}{*}{*}{*}{*}\mathbf{0} \;\ldots\; {*}{*}{*}{*}{*}\mathbf{0} \\ {*}{*}{*}{*}{*}\mathbf{0} \;\ldots\; {*}{*}{*}{*}{*}\mathbf{0} \\ {*}{*}{*}{*}{*}\mathbf{0} \;\ldots\; {*}{*}{*}{*}{*}\mathbf{0} \\ {*}{*}{*}{*}{*}\mathbf{0} \;\ldots\; {*}{*}{*}{*}{*}\mathbf{0} \\ {*}{*}{*}{*}{*}\mathbf{0} \;\ldots\; {*}{*}{*}{*}{*}\mathbf{0} \\ \underbrace{00000\mathbf{1}}_{k+1} \;\ldots\; \underbrace{00000\mathbf{1}}_{k+1} \end{array}\right\}}_{nk+n}$$

Figure 7-3
Extension of a pool Q of the family $\textsc{bottom}_{nk}$ to a pool Q' of the family $\textsc{bottom}_{nk+n}$. Every block of k positions is extended by adding a column, shown in bold, which is always 0 but for a new last bitstring where it is 1. Furthermore, in the new last bitstring all other columns are extended by 0.

Let $b \notin P$. We define $\psi'_i(c\,0) := \psi_i(c)$ and $\psi'_i(c\,1) := b\texttt{[}i\texttt{]}$ for all $c \in \mathbb{B}^k$. Then we have $(\psi'_1 \times \cdots \times \psi'_n)(Q') = P \cup \{b\}$. But this proves the claim. $\square$

7.6 Separating the Positive Closures of Top and Bottom Classes

In this last section, we establish a strict hierarchy of the *positive* truth-table closures of bottom and top partial information. Note, that for the selective languages such a strict hierarchy does not exist as these are *closed* under positive reductions.

7.12 Theorem
Let $\boldsymbol{FC}$ be c.c.c. and recursively presentable. Then

$$\mathrm{R}^{\boldsymbol{C}}_{1\text{-ptt}}\big(\boldsymbol{C}[\textsc{bottom}]\big) \subsetneq \mathrm{R}^{\boldsymbol{C}}_{2\text{-ptt}}\big(\boldsymbol{C}[\textsc{bottom}]\big) \subsetneq \dots .$$

Proof. We argue as in Theorem 7.11 where the k-tt-closures were treated.

Recall that we had n functions ψ_i, this time elements of Δ^k, such that $P = (\psi_1 \times \cdots \times \psi_n)(Q)$ where P has a maximum size and Q is maximal in $\textsc{bottom}_{nk}$.

First, we argued that if the pool Q had a column of 1's we could replace this column of 1's with a column of 0's, and we could then modify the functions ψ_i appropriately so that the image of this new pool under the new functions was still P. Note, that this 'appropriate' modification does not destroy the functions' being monotone. Hence, we can also apply this modification in the monotone case.

Next, we may assume $\psi_i(0^k) = 0$; for otherwise $\psi_i(c) = 1$ for all $c \in \mathbb{B}^k$ and we can restart the whole argument with the function ψ_i replaced by $c \mapsto 0$, and with the pool P replaced by the pool where the column of 1's at position i is replaced by a column of 0's.

As the next step, for all i with $b[i] = 0$, we project the i-th newly added column in Q' to 0, see once more Figure 6-1. After the modification, the pool Q' is still an element of $\mathrm{BOTTOM}_{n(k+1)}$.

To conclude, set $\psi'_i(c\,0) := \psi_i(c)$ and $\psi'_i(c\,1) := 1$ for all $c \in \mathbb{B}^k$. This ensures $(\psi'_1 \times \cdots \times \psi'_n)(Q') = P \cup \{b\}$ and we are done. $\square$

Using the Cone Theorem, one easily checks that the class $\boldsymbol{C}[\mathrm{BOTTOM}]$ is closed under 2-ctt-reductions and that it is not closed under 2-dtt-reductions. The last theorem of this chapter shows that all disjunctive truth-table closures differ.

7.13 Theorem

Let $\boldsymbol{FC}$ be c.c.c. and recursively presentable. Then

$$\mathrm{R}^{\boldsymbol{C}}_{1\text{-dtt}}\big(\boldsymbol{C}[\mathrm{BOTTOM}]\big) \subsetneq \mathrm{R}^{\boldsymbol{C}}_{2\text{-dtt}}\big(\boldsymbol{C}[\mathrm{BOTTOM}]\big) \subsetneq \dots .$$

Proof. We argue as in the above Theorem 7.12, only this time all functions are always fixed to be the logical or. The only difference occurs if we have a column of 1's in the pool Q. But such a column of 1's in Q implies a column of 1's in P. Hence, replacing the column of 1's in Q by a column of 0's will only make the image of Q larger rather than smaller. Thus, we may assume that Q has no column of 1's. The rest of the argument does not need to be modified. Note especially, that the constructed functions ψ'_i are all the logical or. $\square$

Bibliographical Notes

Gray codes, which are even covered by a patent, see Gray (1953), are a widely studied subject. For an introduction see Reingold *et al.* (1977) page 173. Gilbert discovered the symmetric Gray code for $n = 4$, see the last line of the main table in Gilbert (1958).

The main hierarchy of the truth-table closures of the selective languages is due to Hemaspaandra *et al.* (1996). In the paper, the authors also state the strict hierarchy of the Turing closures, Corollary 7.10 in this text. Furthermore, they prove $\mathrm{R}^{\mathbf{P}}_{\mathrm{btt}}\big(\mathbf{P}[\mathrm{SEL}]\big) \subsetneq \mathrm{R}^{\mathbf{P}}_{\mathrm{tt}}\big(\mathbf{P}[\mathrm{SEL}]\big)$ which will also follow from Theorem 8.4 on page 90. Finally, Watanabe showed, referenced in Toda (1991), that $\mathrm{R}^{\mathbf{P}}_{\mathrm{tt}}\big(\mathbf{P}[\mathrm{SEL}]\big) \subsetneq \mathrm{R}^{\mathbf{P}}_{\mathrm{T}}\big(\mathbf{P}[\mathrm{SEL}]\big)$.

Sheu and Long (1994) introduce the notion of Θ-machines, which are polynomial time oracle Turing machines, which query their oracles at most $O(\log n)$ times. They write $\Theta(K)$ for the class of languages accepted by Θ-machines with oracle K. With this notation, Theorem 7.8 asserts $\mathrm{R}^{\mathbf{P}}_{\mathrm{tt}}(K) = \Theta(K)$ for all p-selective languages K.

Theorem 7.9, which states that truth-table reducibility to a p-selective set implies parity reducibility to this set, is most interesting *because p-selective languages share this property with the satisfiability problem*, which follows from the results of Wechsung (1985).

8

Structure of Cones over Families

So gerät jene formale Theorie in Gefahr,
auf das Aposteriorische oder doch Synthetische zurückzufallen,
wie sehr sie sich auch den Anschein gibt,
in der Höhe der Abstraktionen zu schweben.

— Gottlob Frege, *Die Grundlagen der Arithmetik*, Seite 119, Breslau 1884

While the previous chapter studied in detail for what families the selective, bottom and top pools are cones, this chapter studies under which circumstances pools are cones over given families.

In the first section, we show that the pools in $\lceil \mathcal{F} \rceil_{nk}$ are Φ^k-cones over $\mathcal{F}$ for all k, iff $\mathcal{F}$ is of the form $\text{SIZE}_n(m)$ with $m \leq n$ or $m = 2^n$. By the Cone Theorem, this proves that a non-trivial $\boldsymbol{C}[\mathcal{F}]$ is closed under bounded truth-table reductions in the resource bounded case, iff $\boldsymbol{C}[\mathcal{F}] = \boldsymbol{C}[\text{CHEAT}_n]$ for some n. We first proved this in Tantau *et al.* (1998) for polynomial time.

The second section shows that $\boldsymbol{C}[\text{APPROX}]$ is closed under bounded truth-table reductions. This was first proved by Richard Beigel in his PhD thesis for polynomial time.

The third and last section shows that the result of the second section is *optimal for polynomial time*. This improves on a result of Beigel *et al.* (1994). First, the truth-table closure of any partial information class containing non-cheatable languages is not approximable. Second, the approximable languages are not closed under any *unbounded* truth-table reduction. Note, that for once these results are not obtained combinatorially, but rather by a direct diagonalisation argument.

8.1 Characterising the Cone Property for Upward Translated Families

Given a normal n-family $\mathcal{F}$, when are the pools of the upward translation $\lceil \mathcal{F} \rceil_{nk}$ cones over $\mathcal{F}$? This question is important, because if we can answer it in some satisfactory way, *we can also effectively decide which classes* $\boldsymbol{C}[\mathcal{F}]$ *are closed* *under bounded truth-table reductions.*

Lemma 8.1 below shows that the upward translation of a *non-size family* never yields a family of Φ^k-cones over the family for sufficiently large k. Together with the results of the previous chapter, this yields the main characterisation theorem, Theorem 8.2 below. For a stronger result for polynomial time, please see the bibliographical notes.

8.1 Lemma
Let $\mathcal{F}$ be a normal n-family and not a size family. Then $\lceil \mathcal{F} \rceil_{nk}$ is no family of Φ^k-cones over $\mathcal{F}$ for $k = n$.

Proof. By assumption, $\mathcal{F}$ misses some pool P with $|P| = \tau_{\mathcal{F}}$. Then we can find functions ψ_i such that the image of some pool in $\lceil \mathcal{F} \rceil_{nk}$ is exactly P as follows:

Let Q be some pool *in* $\mathcal{F}$ with $|Q| = \tau_{\mathcal{F}}$ and let $Q = \{c_1, \dots, c_{\tau_{\mathcal{F}}}\}$. Then any nk-pool Q' whose eigenpool is Q is an element of $\lceil \mathcal{F} \rceil_{nk}$. One such nk-pool is given by Q duplicated $n-1$ times in the new positions, that is $Q' = \{\, b^n \mid b \in Q \,\}$. Now, let $P = \{b_1, \dots, b_{\tau_{\mathcal{F}}}\}$. We setup the functions ψ_i as follows: let $\psi_i(c_j) := b_j\,[i]$. This ensures $(\psi_1 \times \cdots \times \psi_n)(Q') = P$. ☐

8.2 Theorem
Let $\boldsymbol{FC}$ be c.c.c. and recursively presentable. Then for every normal n-family $\mathcal{F}$ the following propositions are equivalent:

1 *We have $\mathcal{F} = \text{SIZE}_n(m)$ for some $m \leq n$ or $m = 2^n$.*

2 *The class $\boldsymbol{C}[\mathcal{F}]$ is closed under bounded truth-table $\boldsymbol{FC}$-reductions.*

Proof. The first proposition implies the second. As the case $m = 2^n$ is trivial, consider $m \leq n$. We wish to show that $\boldsymbol{C}[\text{SIZE}_n(m)]$ is closed under k-tt-reductions for all k. In Example 3.19 we showed that $\lceil \text{SIZE}_n(m) \rceil_{nk} = \text{SIZE}_{nk}(m)$. But then Corollary 6.4 implies the claim.

For the other direction, assume that $\boldsymbol{C}[\mathcal{F}]$ is closed under bounded truth-table reductions. Assume furthermore that $\mathcal{F} \neq \text{SIZE}_n(2^n)$. We must now argue $\mathcal{F} = \text{SIZE}_n(m)$ for some $m \leq n$.

By the Cone Theorem, $\lceil \mathcal{F} \rceil_{nk}$ is a family of Φ^k-cones over $\mathcal{F}$ for all k. But Lemma 8.1 states exactly that we then have $\mathcal{F} = \text{SIZE}_n(m)$ for some m. Next, if we had $m > n$, then $\text{SEL}_n \subseteq \text{SIZE}_n(m) = \mathcal{F}$. In this situation, Theorem 7.6 states that $\mathcal{F}$ must contain all walks with change numbers at most k. But for $k = 2^{n-1}$ the binary reflected Gray code is such a walk and hence $\mathbb{B}^n \in \mathcal{F}$ contrary to the assumption $\mathcal{F} \neq \text{SIZE}_n(2^n)$. ☐

8.2 Closure Property of Approximable Classes

For any c.c.c. function class $\boldsymbol{FC}$, the class $\boldsymbol{C}[\text{APPROX}]$ is closed under bounded truth-table reductions. For polynomial time, this was first proved by Richard Beigel in his PhD thesis.

8.3 Theorem
Let $\boldsymbol{FC}$ be c.c.c. Then $\boldsymbol{C}[\mathrm{APPROX}]$ is closed under bounded truth-table $\boldsymbol{FC}$-reductions.

Proof. Given a non-trivial m-family $\mathcal{F}$ and a number k, we wish to find a number n large enough such that $\mathrm{R}^{\boldsymbol{C}}_{\Phi^k}\big(\boldsymbol{C}[\mathcal{F}]\big) \subseteq \boldsymbol{C}[\mathrm{APPROX}_n]$. For this, we have to show that $\lceil\mathcal{F}\rceil_{nk}$ is a family of Φ^k-cones over APPROX_n for some n.

Let's analyse the effect of upward translation of the family $\mathcal{F}$ on the size of its largest pool. In Example 3.21 we saw that $\tau_{\lceil\mathcal{F}\rceil_{nk}} \leq S(nk,m) = \sum_{i=0}^{m-1}\binom{nk}{i}$. *This is a polynomial in* n. Hence, it is dominated by $2^n - 1$ for sufficiently large n.

For the sufficiently large n we have $\lceil\mathcal{F}\rceil_{nk} \subseteq \mathrm{SIZE}_{nk}(2^n-1)$. But then Corollary 6.4 states $\mathrm{R}^{\boldsymbol{C}}_{\Phi^k}\big(\boldsymbol{C}[\mathcal{F}]\big) \subseteq \boldsymbol{C}[\mathrm{SIZE}_n(2^n-1)]$. □

8.3 Closure Property of Approximable Polynomial Time is Optimal

This last section establishes that Theorem 8.3 of the previous section is optimal for polynomial time. This improves on a result of Beigel *et al.* (1994) who showed that the set of approximable languages is not closed under truth-table reductions.

As we deal with *unbounded* truth-table reductions, we will not be able to apply the general theory of cones in this case, but use a direct diagonalisation instead.

8.4 Theorem
Let $\mathbf{P}[\mathcal{F}] \nsubseteq \mathbf{P}[\mathrm{CHEAT}]$. *Then* $\mathrm{R}^{\mathbf{P}}_{s(n)\text{-tt}}\big(\mathbf{P}[\mathcal{F}]\big) \nsubseteq \mathbf{P}[\mathrm{APPROX}]$ *for all smooth unbounded function* s.

Proof. Let m be the index of the family $\mathcal{F}$ which we may assume to be normal. Then $\mathbf{P}[\mathcal{F}] \nsubseteq \mathbf{P}[\mathrm{CHEAT}]$ implies then for all $n \geq m$ we have $\lceil\mathcal{F}\rceil_n \nsubseteq \mathrm{CHEAT}_n$. Hence, for all $n \geq m$ there exists a pool $P_n \in \lceil\mathcal{F}\rceil_n$ such that $|P_n| > n$. We will now construct languages L and K such that $L \leq^{\mathbf{P}}_{s(n)\text{-tt}} K$ and $K \in \mathbf{P}[\mathcal{F}]$ but $L \notin \mathbf{P}[\mathrm{APPROX}]$, thus proving the claim.

Let $\beta\colon \mathbb{N} \to \mathbb{N}\times\mathbb{N}$ be a surjection computable in polynomial time such that for each pair (j,k) the set $\{\, i \mid \beta(i) = (j,k)\,\}$ is infinite and its minimum is larger than both j and k. Let M^j be a standard enumeration of Turing machines which stop after $l^j + j$ steps where l is the input length. In the following, j and k are just a shorthands for the first and second component of $\beta(i)$.

First, we construct a very sparse language L using a diagonalisation argument. For each i we will put some subset of the words $W^i = \{w^i_1, \dots, w^i_{n_k}\}$ into L. Here, n_k is the function $n_k := \log s(k)$. The words $w^i_1, \dots, w^i_{n_k}$ are the lexicographically first words of length $\mathrm{tow}(i)$, defined by $\mathrm{tow}(i+1) := 2^{\mathrm{tow}(i)}$ and $\mathrm{tow}(0) := m$.

Let $Q \in \text{APPROX}_{n_k}$ denote the pool output by the machine M^j upon input $w_1^i, \dots, w_{n_k}^i$ or the empty set if no pool from APPROX_{n_k} is output. Then, we setup the intersection $L \cap W^i$ in such a way that the pool Q is no pool for $\chi_L(w_1^i, \dots, w_{n_k}^i)$. As n_k is unbounded in k, the construction ensures $L \notin \mathbf{P}[\text{APPROX}]$.

We also construct K as a very sparse language which contains only subsets of sets V^i, which are the lexicographical first 2^{n_k} words also of length $\mathsf{tow}(i)$. For each i, we code the information $\chi_L\big(w_1^i, \dots, w_{n_k}^i\big)$ into the language K. If c denotes the number coded by $\chi_L\big(w_1^i, \dots, w_{n_k}^i\big)$, we obviously have $c \leq 2^{n_k}$. Now, recall that the pool $P_{2^{n_k}}$ has the property $|P_{2^{n_k}}| > 2^{n_k}$. We put words from V^i into K such that the *characteristic string of the words in V^i is the c-th bitstring in $P_{2^{n_k}}$.*

We claim $L \leq^{\mathbf{P}}_{s(n)\text{-tt}} K$. Upon input w the reduction machine computes an i such that $w \in W^i$—if no such i exists we have $w \notin L$ and we are done. Otherwise, we query the words in V^i. Then, we compute the position of the bitstring in $P_{2^{n_k}}$ and use this information to output $\chi_L(w)$. The definition of n_k ensures $2^{n_k} \leq s(n_k) \leq s(i) \leq s(|w|)$, i. e., that we do not ask too many questions. Furthermore, as $i = \log^*(|w|) + O(1)$ and n_k is logarithmic in i, the pool $P_{2^{n_k}}$ can easily be computed in time polynomial in $|w|$.

We claim $K \in \mathbf{P}[\mathcal{F}]$. Upon input words $w_1, \dots, w_m$ we first compute the largest i such that $V^i \cap \{w_1, \dots, w_m\}$ is non-empty. For words outside V^i we can compute their characteristic value by simulation, because $\mathsf{tow}(i)$ grows fast enough such that this simulation is easily possible in polynomial time. By construction, for the words in V^i the pool $P_{2^{n_k}}$ is the correct pool. As $P_{2^{n_k}} \in \lceil \mathcal{F} \rceil_{2^{n_k}}$, any selection of m columns from $P_{2^{n_k}}$ yields a pool in $\mathcal{F}$. As $\mathcal{F}$ is normal, we can hence output a pool for the words $w_1, \dots, w_m$. □

8.5 Corollary

For all non-trivial families $\mathcal{F}$ and all smooth unbounded functions s we have

$$\mathrm{R}^{\mathbf{P}}_{\text{btt}}\big(\mathbf{P}[\mathcal{F}]\big) \subseteq \mathbf{P}[\text{APPROX}] \textit{ and}$$
$$\mathrm{R}^{\mathbf{P}}_{s(n)\text{-tt}}\big(\mathbf{P}[\mathcal{F}]\big) \subseteq \mathbf{P}[\text{APPROX}], \textit{ iff } \mathbf{P}[\mathcal{F}] \subseteq \mathbf{P}[\text{CHEAT}].$$

Proof. The first claim holds as $\mathbf{P}[\text{APPROX}]$ is closed under bounded truth-table reductions.

For the second claim, the first direction is the contraposition of Theorem 8.4. For the other direction, it suffices to show that each $\mathbf{P}[\text{CHEAT}_n]$ is closed under arbitrary unbounded truth-table reductions. But if $L \leq^{\mathbf{P}}_{\text{tt}} K \in P[\text{CHEAT}_n]$, then for any n input words for L we can compute a polynomial number of queries to K. Using $K \in \mathbf{P}[\text{CHEAT}_n]$, we get a total of n possibilities for the answers to all queries. But this induces a pool of size n for the input words. □

Bibliographical Notes

Theorem 8.2, which characterises the partial information classes closed under bounded truth-table closures, was first proved in Tantau *et al.* (1998) and appears also in

Nickelsen (1999). In both texts, the following strengthened version for the polynomial time case is given:

> **Theorem 3.23 from Nickelsen (1999).** For $\mathcal{D} \in \mathcal{N}_n$ with $\mathcal{D} \neq (2^n, n)$-SIZE the following are equivalent:

1 $\mathcal{D} = (m, n)$-SIZE for some $m \leq n$.

2 $\mathbf{P}[\mathcal{D}]$ is closed under polynomial time 2-tt reductions.

3 $\mathbf{P}[\mathcal{D}]$ is closed under polynomial time Turing reductions.

Here, $\mathcal{N}_n$ is the lattice of normal n-families. This strengthened version can be obtained from Theorem 8.2 in the polynomial case. First, note that closure under 2-query truth-table reductions implies closure under bounded truth-table reductions in general, see Nickelsen (1999). Second, Lemma 19 of Amir *et al.* (1990) states that for each n the class $\mathbf{P}[\text{CHEAT}_n]$ is even closed under Turing reductions. This was discovered independently by Goldsmith *et al.* (1993).

Theorem 8.4 is a strengthened version of the following theorem:

> **Theorem 2.9 of Beigel *et al.* (1994).** There are sets A, B such that A is p-superterse, B is $(2,3)_p$-recursive, and $A \leq_{tt}^{p} B$.

Translated into the notation of this text, they construct languages $A \notin \mathbf{P}[\text{APPROX}]$ and $B \in \mathbf{P}[\text{FREQ}_3(1)]$ such that $A \leq_{\text{tt}}^{\mathbf{P}} B$. Nickelsen (1999) pointed out that their proof actually even asserts $B \in \mathbf{P}[\text{BOTTOM}_3]$.

Theorem 8.4 settles another small open question. Theorem 4.6 of Beigel *et al.* (1994) states that if SAT is truth-table reducible to an approximable language *with a sublinear number of queries*, then $\mathbf{P} = \mathbf{NP}$. However, it was not clear whether this result was actually stronger than Theorem 1.23 on page 14, which states that if SAT is approximable, then $\mathbf{P} = \mathbf{NP}$. The reason is that it was unclear whether the approximable languages are not perhaps *closed* under unbounded but sublinear truth-table reductions. Theorem 8.4 shows that this is not the case.

Conclusion

»Mondenkind«, flüsterte er, »ist das nun das Ende?«
»Nein«, antwortete sie, »es ist der Anfang.«
— Michael Ende, *Die unendliche Geschichte*

This thesis tried to demonstrate that many problems concerning partial information can be turned into finitary combinatorial problems. These combinatorial problems often have simple, sometimes even elegant solutions.

Representing Classes by Families

The first part of the thesis treated combinatorial representations of partial information classes. Three basic questions were addressed.

How exactly do families and partial information classes relate?

Chapter 2 gave a partial answer to this question by extending Nickelsen's Unique Normal Form Theorem to all function classes which are c.c.c. and recursively presentable. Chapter 4 addressed the problem for the class of recursive functions, but could only identify the stable families for index 2.

As normal families represent resource bounded partial information classes, a result on the structure of normal families is also a result on partial information classes. Hence, combinatorial properties of normal families reflect properties of partial information classes.

What are the combinatorial properties of normal families?

Chapter 3 addressed this question. Normal families correspond to antichains in the unit posets. We saw how families can be translated upward. As an application, I proved that inclusion is well-founded on cheatable partial information classes.

What are the combinatorial properties of stable families?

Stable families are the counterpart of normal families for the class of recursive functions. While Chapter 4 treats stable families, I could give no general theory of the combinatorics of stable families there, *because no decision procedure for stability of families is known*. Hence, I had to address the question in a roundabout manner and often had to argue in the class domain, rather than in the family domain.

Nevertheless, at least one combinatorial property of stable families can be obtained as a consequence of Theorem 4.12: Every stable n-family $\mathcal{F}$ is either contained in $\lceil \mathcal{E} \rceil_n$ or in $\lceil \mathcal{F} : \mathcal{E} \rceil_n$ for any normal family $\mathcal{E}$. As an application, I could prove a generalisation of the Generalised Non-Speedup Theorem.

Representing Closures by Cones

In the second part, this thesis tried to demonstrate that not only the inclusion problem of partial information classes, but also the inclusion problem of their bounded truth-table closures can be turned into a combinatorial problem. Again, three basic questions were addressed:

How can reductions themselves be represented combinatorially?

Chapter 5 answered this question for bounded reductions by introducing the notion of *evaluation types*. While the definition appeals to the notion of truth-table reductions, I could show that all bounded reductions considered in the literature *including bounded Turing reductions* can be modelled by evaluation types.

How can bounded reductions of partial information classes be represented?

The Cone Theorem answered this question in part. It states that in the resource bounded case, the Ψ-reduction closure of a class $\boldsymbol{C}[\mathcal{F}]$ is a subset of a class $\boldsymbol{C}[\mathcal{G}]$, iff the pools of $\mathcal{F}$ are Ψ-cones over $\mathcal{G}$. An immediate consequence of the theorem was that it is combinatorially decidable which partial information classes are closed under a specific bounded reduction. The Cone Theorem does not answer the question which closures *coincide*—which is an intriguing question on which only little is known.

What are the combinatorial properties of cones?

Results on the structure of cones induce theorems on the reduction closures of partial information classes. Two such theorem were obtained:

The selective pools are all Φ^k-cones over a family, iff it contains all walks with change numbers at most k. This purely combinatorial result implied that all bounded truth-table closures of the p-selective languages differ, simply because walks with different maximum change numbers differ.

A non-trivial family can only be a family of Φ_k-cones over itself—after upward translation—if it is a $\text{SIZE}_n(m)$ family with $m \leq n$. As a consequence I obtained a characterisation of the partial information classes closed under bounded truth-table reductions. These classes are exactly the cheatability classes.

Outlook

An important open problem is the stability problem of families. A decision procedure for stability of families would complete the translation process from the class domain to the family domain, which is currently available only for resource bounded function classes.

The analysis of stable families yielded the Non-Speedup Theorem due to Beigel. I could also prove a theorem which states that allowing a cheatable pool to be output does not change the power of selectivity. Perhaps, this is no coincidence and we generally have $\mathbf{REC}[\mathcal{F}] = \mathbf{REC}[\mathcal{F} \cup \text{CHEAT}_n]$ for all n-families $\mathcal{F}$. At least for $n \leq 2$ this conjecture is correct.

The upward translation of families still deserves further research. While the combinatorics of upward translation is solved *in principle* by Theorem 3.12, it remains unclear how upward translation can be done quickly and easily. Apart from the Cone Theorem, also a construction of Ronneburger (1998) would benefit from a quick algorithm for upward translations. Ronneburger introduced the concept of *dedicated pools* which are maximal pools of the upward translated families and showed that if dedicated pools can easily be computed for a given family, their size is a lower bound on the advice necessary to decide the corresponding partial information class. Currently, no algorithm is known for computing dedicated pools in polynomial time in general.

Nickelsen's conjecture that inclusion is well-founded on partial information classes remains unproven despite the progress made in Section 3.6. The proof in Section 3.6 of the somewhat weaker result that inclusion is well-founded on cheatable partial information relied heavily on the correspondence between normal families and antichains in the unit posets. To me, it seems promising to attempt a proof of the general result also using antichains. Nevertheless, antichains might also be helpful in refuting the conjecture. In any case, a better understanding of the effects of upward translation would also be most helpful in this context.

Cones are a powerful tool for representing the inclusion of *bounded* truth-table closures of partial information classes in other classes. It would be most satisfactory if the combinatorial theory of cones could be extended to truth-table reductions with unbounded numbers of queries.

Cones cannot be used to solve the equality problem of bounded truth-table closures. Can a theorem be stated, preferably of combinatorial nature, which gives exact conditions in terms of evaluation types Ψ and Ω and families $\mathcal{F}$ and $\mathcal{G}$ under which we have $\mathrm{R}^{\mathcal{C}}_{\Psi}(\mathcal{F}) \subseteq \mathrm{R}^{\mathcal{C}}_{\Omega}(\mathcal{G})$? It might be that the conditions depend on the specific resources available in the *function class* under consideration. This might even shed some new light on these function classes themselves.

Bibliography

Manindra Agrawal, Richard Beigel and Thomas Thierauf.
Modulo information from nonadaptive queries to **NP**.
Technical Report TR96-001, Electronic Colloquium on Computational Complexity, available at `http://www.eccc.uni-trier.de/eccc`, 1996.

Amihood Amir, Richard Beigel and William I. Gasarch.
Some connections between bounded query classes and non-uniform complexity.
In *Proceedings of the Fifth Annual Structure in Complexity Theory Conference*, pages 232–243, Barcelona, Spain, 1990. IEEE Computer Society Press.

José Luis Balcázar, Josep Díaz and Joaquim Gabarró.
Structural Complexity I, volume 11 of *EATCS Monographs on Theoretical Computer Science*.
Springer-Verlag, Berlin–Heidelberg–New York, 1988.

José Luis Balcázar, Josep Díaz and Joaquim Gabarró.
Structural Complexity II, volume 22 of *EATCS Monographs on Theoretical Computer Science*.
Springer-Verlag, Berlin–Heidelberg–New York, 1990.

Richard Beigel.
Query-limited reducibilities.
PhD thesis, Stanford University, Stanford, USA, 1987.

Richard Beigel.
Bounded queries to SAT and the Boolean hierarchy.
Theoretical Computer Science, 84(2):199–223, 1991.

Richard Beigel, William I. Gasarch and Efim Kinber.
Frequency computation and bounded queries.
In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference*, pages 125–132, Minneapolis, Minnesota, 1995a. IEEE Computer Society Press.

Richard Beigel, Martin Kummer and Frank Stephan.
Quantifying the amount of verboseness.
In Anil Nerode and M. A. Taitslin, editors, *Proceedings of the Second International Symposium on Logical Foundations of Computer Science—Tver 92*, volume 620 of *Lecture Notes in Computer Science*, pages 21–32, Tver, Russia, 1992. Springer-Verlag.
Final version published in Beigel *et al.* (1995c).

Richard Beigel, Martin Kummer and Frank Stephan.
Approximable sets (preliminary version).
In *Proceedings of the Ninth Annual Structure in Complexity Theory Conference*, pages 12–23, Amsterdam, The Netherlands, 1994. IEEE Computer Society Press.
Final version published in Beigel *et al.* (1995b).

Richard Beigel, Martin Kummer and Frank Stephan.
Approximable sets.
Information and Computation, 120(2):304–314, 1995b.

Richard Beigel, Martin Kummer and Frank Stephan.
Quantifying the amount of verboseness.
Information and Computation, 118(1):73–90, 1995c.

Manuel Blum.
A machine-independent theory of the complexity of recursive functions.
Journal of the Association for Computing Machinery, 14(2):322–336, 1967.

Harry Buhrman, Leen Torenvliet and Peter van Emde Boas.
Twenty questions to a p-selector.
Information Processing Letters, 48(4):201–204, 1993.

Stephen A. Cook.
The complexity of theorem-proving procedures.
In *Conference Record of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 1971.

Stephen A. Cook and Pierre McKenzie.
Problems complete for deterministic logarithmic space.
Journal of Algorithms, 8(3):385–394, 1987.

Christos Faloutsos.
Gray codes for partial match and range queries.
IEEE Transactions on Software Engineering, 14(10):1381–1393, 1988.

Otto Forster.
Analysis I.
Vieweg-Verlag, 1976.

Edgard N. Gilbert.
Gray codes and paths on the n-cube.
Bell Systems Technical Journal, 37(3):815–826, 1958.

Judy Goldsmith, Deborah Joseph and Paul Young.
Using self-reducibilities to characterize polynomial time.
Information and Computation, 104(2):288–308, 1993.

Frank Gray.
Pulse code communications.
United States Patent 2 632 058, 1953.

Yenjo Han and Thomas Thierauf.
Restricted information from nonadaptive queries to **NP**.
In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference*, pages 206–213, Minneapolis, Minnesota, 1995. IEEE Computer Society Press.

Lane A. Hemaspaandra, Albrecht Hoene and Mitsunori Ogihara.
Reducibility classes of **P**-selective sets.
Theoretical Computer Science, 155(2):447–457, 1996.

Lane A. Hemaspaandra and Leen Torenvliet.
Optimal advice.
Theoretical Computer Science, 154(2):367–377, 1996.

Maren Hinrichs and Gerd Wechsung.
Time bounded frequency computations.
In *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity*, pages 185–192, Ulm, Germany, 1997. IEEE Computer Society Press.

Albrecht Hoene and Arfst Nickelsen.
Counting, selecting, and sorting by query-bounded machines.
In Patrice Enjalbert, Alain Finkel and Klaus W. Wagner, editors, *Proceedings of the Tenth Annual Symposium on Theoretical Aspects of Computer Science—STACS 93*, volume 665 of *Lecture Notes in Computer Science*, pages 196–205, Würzburg, Germany, 1993. Springer-Verlag.

Valerie Illingworth, editor.
Dictionary of Computing.
Oxford University Press, New York, USA, 1983.

Neil Immerman.
Nondeterministic space is closed under complementation.
SIAM Journal on Computing, 17(5):935–938, 1988.

Thomas Jech.
Set Theory, volume 97 of *Perspectives in Mathematical Logic*.
Springer-Verlag, Berlin–Heidelberg–New York, second edition, 1997.

Carl G. Jockusch, Jr.
Semirecursive sets and positive reducibility.
Transactions of the American Mathematical Society, 131:420–436, 1968.

Neil D. Jones, Y. Edmund Lien and William T. Laaser.
New problems complete for nondeterministic log space.
Mathematical Systems Theory, 10:1–17, 1976.

Richard M. Karp and Richard J. Lipton.
Some connections between nonuniform and uniform complexity classes.
In *Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 302–309, Los Angeles, California, 1980.

Ker-I Ko.
On self-reducibility and weak **P**-selectivity.
Journal of Computer and System Sciences, 26(2):209–221, 1983.

Johannes Köbler.
On the structure of low sets.
In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference*, pages 246–261, Minneapolis, Minnesota, 1995. IEEE Computer Society Press.

Johannes Köbler and Thomas Thierauf.
Complexity-restricted advice functions.
SIAM Journal on Computing, 23(2):261–275, 1994.

Mark W. Krentel.
The complexity of optimization problems.
Journal of Computer and System Sciences, 36(3):490–509, 1988.

Martin Kummer.
A proof of Beigel's cardinality conjecture.
Journal of Symbolic Logic, 57(2):677–681, 1992.

Martin Kummer and Frank Stephan.
Some aspects of frequency computation.
Technical Report 21/91, Universität Karlsruhe, Fakultät für Informatik, Germany, 1991.

Martin Kummer and Frank Stephan.
Recursion theoretic properties of frequency computation and bounded queries.
Information and Computation, 120(1):59–77, 1995.

Richard E. Ladner.
The circuit value problem is log space complete for $\mathbf{P}$.
SIGACT News, 7(1):18–20, 1975.

Richard E. Ladner, Nancy A. Lynch and Alan L. Selman.
A comparison of polynomial time reducibilities.
Theoretical Computer Science, 1(2):103–123, 1975.

Lawrence H. Landweber and Edward L. Robertson.
Recursive properties of abstract complexity classes.
Journal of the Association for Computing Machinery, 19(2):296–308, 1972.

Robert McNaughton.
The theory of automata, a survey.
Advances in Computers, 2:379–421, 1961.

John Myhill.
Recursive digraphs, splinters and cylinders.
Mathematische Annalen, 138:211–218, 1959.

Arfst Nickelsen.
On $\mathcal{D}$-verbose languages.
In Rudiger Reischuk and Michel Morvan, editors, *Proceedings of the Fourteenth Annual Symposium on Theoretical Aspects of Computer Science—STACS 97*, volume 1200 of *Lecture Notes in Computer Science*, pages 307–318, Lübeck, Germany, 1997. Springer-Verlag.

Arfst Nickelsen.
Polynomial Time Partial Information Classes.
PhD thesis, Technische Universität Berlin, Germany, 1999.

Piergiorgio Odifreddi.
Classic Recursion Theory, volume 125 of *Studies in Logic and the Foundations of Mathematics*.
North-Holland, Amsterdam–London–New York–Tokyo, 1989.

James C. Owings, Jr.
A cardinality version of Beigel's nonspeedup theorem.
Journal of Symbolic Logic, 54(3):761–767, 1989.

Christos H. Papadimitriou.
Computational Complexity.
Addison–Wesley, 1994.

Edward M. Reingold, Jurg Nievergelt and Narsingh Deo.
Combinatorial Algorithms: Theory and Practice.
Prentice–Hall, Englewood Cliffs, New Jersey, 1977.

Detlef Ronneburger.
Upper and lower bounds for token advice for partial information classes.
Master's thesis, Technische Universität Berlin, Germany, 1998.

Gene F. Rose.
An extended notion of computability.
In *Proceedings of the 1960 International Congress for Logic, Methodology, and Philosophy of Science*, Stanford, California, 1960.

Uwe Schöning.
A uniform approach to obtain diagonal sets in complexity classes.
Theoretical Computer Science, 18(1):95–103, 1982.

Alan L. Selman.
P-selective sets, tally languages, and the behavior of polynomial time reducibilities on **NP**.
Mathematical Systems Theory, 13:55–65, 1979.

Alan L. Selman.
Some observations on **NP** real numbers and **P**-selective sets.
Journal of Computer and System Sciences, 23(3):326–332, 1981.

Alan L. Selman.
Analogues of semicursive sets and effective reducibilities to the study of **NP** complexity.
Information and Control, 52(1):36–51, 1982a.

Alan L. Selman.
Reductions on **NP** and **P**-selective sets.
Theoretical Computer Science, 19(3):287–304, 1982b.

Ming-Jye Sheu and Timothy J. Long.
The extended low hierarchy is an infinite hierarchy.
SIAM Journal on Computing, 23(3):488–509, 1994.

Róbert Szelepcsényi.
The method of forcing for nondeterministic automata.
Bulletin of the European Association for Theoretical Computer Science, 33:96–100, 1987.

Till Tantau, Detlef Ronneburger, Ronald Grönke and Michael Bahr.
Deterministische Polynomialzeitklassen.
Report of a research class supervised by Arfst Nickelsen, Technische Unversität Berlin, 1998.

Seinosuke Toda.
On polynomial-time truth-table reducibility of intractable sets to **P**-selective sets.
Mathematical Systems Theory, 24(2):68–82, 1991.

Boris A. Trakhtenbrot.
On frequency computations of functions.
Algebra i Logika, 2:25–32, 1963.
In Russian.

Gerd Wechsung.
On the Boolean closure of **NP**.
In Lothar Budach, editor, *Proceedings of the Fifth Conference on Fundamentals of Computation Theory*, volume 199 of *Lecture Notes in Computer Science*, pages 485–493, Cottbus, Germany, 1985. Springer-Verlag.
An unpublished precursor of this paper was coauthored by Klaus W. Wagner.

List of Notations

Set Theory and Calculus

$\lvert S \rvert$	Cardinality of the set S.
$\mathcal{P}(S)$	Powerset of the set S.
$\mathbb{N}$	The natural numbers, i. e., $\mathbb{N} = \{0, 1, 2, \dots\}$.
$\mathbb{N}^*$	The positive integers, i. e., $\mathbb{N}^* = \{1, 2, \dots\}$.
$\mathbb{N}^*_n$	Set of the first n positive integers, i. e., $\mathbb{N}^*_n = \{1, \dots, n\}$.
$\chi_S(x)$	Characteristic value of x, i. e., $\chi_S(x) = 1$ for $x \in S$ and $\chi_S(x) = 0$ otherwise.
$\chi_S(x_1, \dots, x_n)$	Extension to tuples, i. e., $\chi_S(x_1, \dots, x_n) = \chi_S(x_1) \cdots \chi_S(x_n)$.
$f\colon S \to T$	Assertion that f is a total function from S to T.
$S/{\sim}$	Partition of the set S, obtained by factorising S with respect to the equivalence relation $\sim$.
$[x]_\sim$	Equivalence class of the element x with respect to $\sim$.
$S \mathbin{\triangle} T$	Symmetric difference of S and T, i. e., $S \mathbin{\triangle} T = S \setminus T \cup T \setminus S$.
$\log^*\colon \mathbb{N} \to \mathbb{N}$	Iterated logarithm, i. e., $\log^*(n+1) = 1 + \log^*\big(\lfloor \log(n) \rfloor\big)$.
$\mathrm{tow}\colon \mathbb{N} \to \mathbb{N}$	Iterated exponentiation, i. e., $\mathrm{tow}(n+1) = 2^{\mathrm{tow}(n)}$.

Words

$\mathtt{0}$, $\mathtt{1}$	The two values of a bit. Denoted in a smaller font than normal digits.
Σ	An alphabet. In this text $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.
Σ^n	Set of all words over Σ of length n.
$\Sigma^{\leq n}$	Set of all words over Σ of maximum length n.
Σ^*	Set of all words over Σ.
λ	The empty word.
w, u, v, …	Variables for words. Words are elements of Σ^*.
$\langle w_1, \dots, w_n \rangle$	Tupling of the words w_1 to w_n, obtained by writing the words alongside, doubling the bits and inserting a $\mathtt{01}$ stop sequence after each word.
$\mathrm{p}_i\colon \Sigma^* \to \Sigma^*$	Projection to the i-th component, i. e., $\mathrm{p}_i\big(\langle w_1, \dots, w_n \rangle\big) = w_i$ for $i \leq n$, and $\mathrm{p}_i(w) = \lambda$, if w is no n-tuple with $1 \leq i \leq n$.
$f \circ g$	Concatenation of f and g, i. e., $(f \circ g)(w) = f(g(w))$.
$f \times g$	Parallel application of f and g, i. e., $(f \times g)\big(\langle u, v \rangle\big) = \langle f(u), g(v) \rangle$.
$\langle f, g \rangle$	Tupling of f and g, i. e., $\langle f, g \rangle (w) = \langle f(w), g(w) \rangle$.
$\sqsubseteq$	Prefix relation on words.
$\leq_{\mathrm{lex}}$	Lexicographic or dictionary ordering.

Languages

L, K, N, ...	Variables for languages. Languages are subsets of Σ^*.
L_{tag}	Tagged language, i. e., $L_{\text{tag}} = \{ w \mid \mathrm{p}_1(w) \in L \}$.
$\mathrm{L}(M)$	Language accepted by the Turing machine M.
$\mathrm{L}(M, X)$	Language accepted by the oracle Turing machine M with oracle X.

Bits and Bitstrings

$\mathbb{B}$	The set containing the two values of a bit, i. e., $\mathbb{B} = \{0, 1\}$.
b, c, d, ...	Variables for bitstrings. Bitstrings are elements of $\mathbb{B}^*$. Bits are bitstrings of length one.
$\pi_i^b \colon \mathbb{B}^k \to \mathbb{B}^k$	Projection of the i-th coordinate to b.
$\#_1 : \mathbb{B}^k \to \mathbb{N}$	Function that counts the 1's in its input, i. e., $\#_1(b) = \sum_{i=1}^{k} b[i]$.
$\mathrm{d}(b, c)$	Hamming distance between b and c, i. e., the number bits where they differ.
$b[i_1, \dots, i_n]$	Bitstring obtained by writing the bits at positions i_1 to i_n of the bitstring b alongside. The leftmost bit is at position 1.
$P[i_1, \dots, i_n]$	Extension to sets of bitstrings, i. e., the set $\{ b[i_1, \dots, i_n] \mid b \in P \}$.
$\leq_{\text{pw}}$	Pointwise ordering of $\mathbb{B}^n$, i. e., the ordering induced by the Boolean algebra $\mathbb{B}^n$.

Boolean Functions

$\phi, \psi, \dots$	Variables for Boolean functions, i. e., $\phi \colon \mathbb{B}^k \to \mathbb{B}$ for some k.
$\phi(P)$	Image of the set P of bitstrings under ϕ.
$\psi_1 \times \cdots \times \psi_n$	Product of the functions $\psi_1, \dots, \psi_n \colon \mathbb{B}^k \to \mathbb{B}$, going from $\mathbb{B}^{nk}$ to $\mathbb{B}^n$.
$\top, \bot : \mathbb{B}^k \to \mathbb{B}$	*Verum* and *falsum*.
$\oplus_k : \mathbb{B}^k \to \mathbb{B}$	The k-ary parity function, defined as the number of 1's in its argument modulo 2.
nav_k	Navigation functions. *See Definition 5.16.*
$\Psi, \Omega, \dots$	Variables for sets of Boolean functions.
Φ^k	The set of k-ary Boolean functions.
Δ^k	The set of monotone k-ary Boolean functions.
Σ^k	The set containing the k-ary logical or.
Π^k	The set containing the k-ary logical and.
Ξ^k	The set containing $\oplus_k$, its negation, verum and falsum.
Υ_k	The set containing all $(2^k - 1)$-ary Boolean functions whose kernel includes the kernel of nav_k. *See Definition 5.17.*

Pools

P, Q, R, ...	Variables for pools. Pools are subsets of $\mathbb{B}^n$.
$\left\{ \begin{smallmatrix} b_1 \\ \vdots \\ b_n \end{smallmatrix} \right\}$	Compact notation for the set $\{b_1, \dots, b_n\}$ for bitstrings b_i. *See Notation 3.2.*

Special Families

Please see Table 1-3 on page 6.

Families

$\mathcal{F}, \mathcal{G}, \mathcal{H}, \ldots$	Variables for families. Families are coverings of $\mathbb{B}^n$. Usually, $\mathcal{F}$ denotes an n-family, $\mathcal{E}$ an m-family, and $\mathcal{C}$ an nk-family.
$\overline{\mathcal{F}}$	Subset closure of $\mathcal{F}$. *See Definition 2.1.*
$\tau_{\mathcal{F}}$	Size of largest pool in $\mathcal{F}$. *See Definition 1.4.*
$\langle \mathcal{G} \rangle$	Least normal family containing $\mathcal{G}$. *See Definition 3.1.*
$\langle \mathcal{G} \rangle_{\mathrm{dist}}$	Least weakly normal family containing $\mathcal{G}$. *See Definition 3.1.*
$\lceil \mathcal{F} \rceil_m$	Upward translation of $\mathcal{F}$ to index m. *See Notation 3.13.*
$(\mathfrak{U}_n, \preceq)$	The poset of n-units. *See Definition 3.9.*
$(\mathfrak{W}_n, \preceq)$	The poset of weak n-units. *See Definition 3.9.*
$\mathcal{F} : \mathcal{E}$	Family of $\mathcal{E}$-hard remainder pools of pools in $\mathcal{F}$. *See Definition 4.9.*

Complexity Classes

$\mathbf{L}$	Class of languages decidable in deterministic logarithmic space.
$\mathbf{FL}$	Class of functions computable in deterministic logarithmic space.
$\mathbf{NC}$	Nick's class, i. e., class of languages computable in parallel poly-logarithmic time.
$\mathbf{polyL}$	Class of languages decidable in poly-logarithmic space.
$\mathbf{P}$	Class of languages decidable in deterministic polynomial time.
$\mathbf{FP}$	Class of functions computable in deterministic polynomial time.
$\mathbf{NP}$	Class of languages decidable in non-deterministic polynomial time.
$\Delta_2\mathbf{P}$	Second delta level of the polynomial hierarchy, i. e., $\Delta_2\mathbf{P} = \mathbf{P}^{\mathbf{NP}}$.
$\mathbf{PSPACE}$	Class of languages decidable in deterministic polynomial space.
$\mathbf{FPSPACE}$	Class of functions computable in deterministic polynomial space whose output length is polynomial in the input length.
$\mathbf{REC}$	Class of recursive languages.
$\boldsymbol{K}$	Variable for classes of languages.
$\boldsymbol{K}/1$	Class $\boldsymbol{K}$ with one extra bit of advice per length level.
$\boldsymbol{K}/\mathrm{poly}$	Class $\boldsymbol{K}$ with a polynomial amount of advice per length level.
$\boldsymbol{FC}$	Variable for Cartesian and compositionally closed function classes.
$\boldsymbol{C}[\mathcal{F}]$	Partial information class over the family $\mathcal{F}$. *See Definition 1.11.*
$\boldsymbol{C}_{\mathrm{dist}}[\mathcal{F}]$	Partial information class over the family $\mathcal{F}$ where only distinct words may be given. *See Definition 1.11.*

Reductions

$\leq_{\mathrm{m}}$	Many-one reduction.
$\leq_{\Psi}$	General Ψ-truth-table reduction.
$\leq_{k\text{-tt}}, \leq_{\Phi^k}$	Truth-table reduction with k queries.
$\leq_{k\text{-ptt}}, \leq_{\Delta^k}$	Positive truth-table reduction with k queries.
$\leq_{k\text{-ctt}}, \leq_{\Pi^k}$	Conjunctive truth-table reduction with k queries.
$\leq_{k\text{-dtt}}, \leq_{\Sigma^k}$	Disjunctive truth-table reduction with k queries.
$\leq_{\Xi^k}$	Parity truth-table reduction with k queries.
$\leq_{k\text{-T}}, \leq_{\Upsilon_k}$	Turing reduction with k queries.

Index

A

adaptive queries, 59
advice classes, 52
advice functions, 52, 69
Agrawal, Manindra, 65
allowed pools, 3
Amir, Amihood, 92
antichains, 32–33, 36
  figures for index two, 42
  number of, 33
APPROX, *see* approximable families
approximable families
  and reductions, 90
  and satisfiability, 14
  definition of, 4
  optimal reductions, 90, 91
  upward translation, 39

B

Balcázar, José Luis, 23, 69
bases, 31–33
Beigel, Richard, 18, 38, 65, 92
  (1987), 29, 39, 41, 48, 88, 89
  (1992), 18, 46, 50, 56
  (1994), x, 2, 11, 12, 14, 17, 77, 88, 90, 92
  (1995a), vii, 8, 17, 28
binary reflected Gray code, 79, 89
Bischoff, Johann Paul, 1, 58
Blum, Manuel, 28
Boolean algebras, 4
Boolean functions, 63, 71
bottom families, 37
  hierarchy of closures, 85–87
bounded truth-table reductions, 65, 76, 89, 90
bounded Turing reductions, 66
branches, 54–55
Buhrman, Harry, 76
but one polynomial time, 10, 20

C

c.c.c., *see* Cartesian and compositionally closed
canonically complete language, 73
cardinality reductions, 66
Cardinality Theorem, 51
Cartesian and compositionally closed
  definition of, 7
chains, 5, 10, 54
change numbers of walks, 79
changing the index, *see* upward translation
characteristic string, 2
CHEAT, *see* cheatable families
cheatable families, 40, 47, 53
  and reductions, 89
  definition of, 5
  Non-Speedup Theorem, 48
  optimal reductions, 91
  upward translation, 38
circuit value problem, 16
closed ball, 6
closed chains, 10, 54
closed languages, 10
closed sets, 10
closed under projections, 22
closed under selections, 22
closures of reductions, *see* reductions
coding pools, 7
coding word tuples, 7
columns of pools, 30, 47–50
complete problems, 60, 61, 73
Cone Theorem, 74
cones, 72, 88
  and non-size families, 89
  Cone Theorem, 74
  definition of, 71
  selective pools, 80
conjunctive truth-table reductions, 65
Cook, Stephen A., 11, 60, 69

COSMC, *see* strongly membership comparable
coverings, 3

D

decision problems, 2
Dedekind cuts, 9
dense subsets of families, 20
Deo, Narsingh, 79, 87
diagonalisation, 26, 90
diameter of a pool, 6, 10, 20, 57
different tuple length, *see* upward translation
disjunctive truth-table reductions, 65, 87
disjunctively self-reducible, 13
distinct indices, 35, 38–40
distinct words, 9, 11, 75
downward translation, 36
Díaz, Josep, 23, 69

E

effective enumeration, 23
eigenpools, 30, 32, 37, 85
Ende, Michael, 93
enumeration of tuples, 47
enumeration of Turing machines, 23, 90
equality of classes, 28
evaluation types, 63, 66, 68
evaluator, 63, 65, 71

F

Faloutsos, Christos, 80
falsum, 63
families, 3
  closed under projections, 22
  closed under selections, 22
  definition of, 3
  normal, *see* normal families
  of cones, *see* cones
  subset closed, 20
  table of, 6
  weakly normal, *see* normal families
families with index, 3
families without index, 11
Forster, Otto, 9
Frege, Gottlob, 59, 70, 78, 88
frequency computations, vi

G

Gabarró, Joaquim, 23, 69
Gasarch, William I., 92
  (1995a), vii, 8, 17, 28
Generalised Non-Speedup Theorem, 50–51
generating systems, 30–31
generator, 63, 65, 71
Gilbert, Edgard N., 79, 80, 87
Goethe, Johann Wolfgang von, vi
Goldsmith, Judy, 92
Gray code, 79
Gray, Frank, 87

H

Hamming distance, 4, 79
Hamming metric, 6
Hamming spaces, 5, 81
Han, Yenjo, 69
hard remainder pools, 49, 50
hard tuples, 47–50
  definition of, 47
  for minimal families, 52
  for trivial families, 47
Hemaspaandra, Lane A., x, 57, 78, 82, 83, 87
hierarchy of closures, 78, 82, 84
  bottom classes, 85–87
Hinrichs, Maren, 40, 41
Hoene, Albrecht, x, 13, 17, 78, 82, 83, 87
hypercubes, 5, 79

I

Immerman, Neil, 16
inclusion of classes, 26
  well-foundedness, 40–41
index notation
  families with, 3
  families without, 11
injective mapping between pools, 39
intersection trick, 21, 35, 74

J

Jech, Thomas, 41
Jockusch, Carl G. Jr., 10, 16
Jones, Neil D., 60
Joseph, Deborah, 92

K

Karp, Richard M., 52, 57, 69
Kinber, Efim
  (1995a), vii, 8, 17, 28
known words, 21, 49
Ko, Ker-I, 57

Krentel, Mark W., 64
Kummer, Martin, 17, 51, 56, 57
(1992), 18, 46, 50, 56
(1994), x, 2, 11, 12, 14, 17, 77, 88, 90, 92
Köbler, Johannes, 13, 69

L

Laaser, William T., 60
Ladner, Richard E., 15, 63, 64, 69
Landweber, Lawrence H., 23, 28
lattice of n-families, 82, 85
left cuts, 10
lexicographical ordering of words, 9, 90
Lien, Y. Edmund, 60
linear ordering of words, 9
Lipton, Richard J., 52, 57
logarithmic space, 8, 16, 60
Long, Timothy J., 87
Lynch, Nancy A., 63, 64, 69

M

many-one reductions, 60, 65, 73
closure under, 75
Martin, Donald A., 10
maximal chains, 5, 79
maximal pools, 20, 22
maximum pool size of families, 5
McKenzie, Pierre, 60
McLaughlin, Thomas G., 10
McNaughton, Robert, 57
metric spaces, 4, 5
MIN, *see* minimal families
minimal generating system, 31
minimum families, 30, 37, 53
model of computation, 7
Myhill, John, 77

N

navigation functions, 68
Nick's class, 16
Nickelsen, Arfst, 13, 17
(1997), ix, 3, 4, 8, 10, 17, 19, 22, 26, 28, 29, 34, 41, 51, 56, 75, 76
(1999), ix, 3, 6, 17, 20, 29, 30, 34, 37, 40, 41, 76, 92
Nievergelt, Jurg, 79, 87
non-adaptive queries, 59
non-cheatable languages, 88
non-distinct words, 75
non-recursive languages, 50
Non-Speedup Theorem, 48, 50
non-trivial languages, 63, 73
normal families, 36
bases for, 31
definition of, 22
figures for index two, 43
generating systems for, 30–31
upward translation, 34
number of antichains, 33

O

Odifreddi, Piergiorgio, 24
Ogihara, Mitsunori, x, 78, 82, 83, 87
one to one reduction, 75
optimal closure property, 90, 91
optimal pools, *see* intersection trick
oracle Turing machine, 50, 61
orderings
lexicographical, 9, 90
linear, 9
partial of bitstrings, 4
partial of units, 32
partial of words, 10
pointwise, 4
prefix, 10
Owings's Separation Lemma, 38
Owings, James C. Jr., 38, 56

P

Papadimitriou, Christos H., 8, 15, 61, 69
parallel logarithmic time, 60
parity reductions, 65, 84
partial information classes, 2, 8
recursive, 48, 51
partial ordering of bitstrings, 4
partial ordering of units, 32
partial ordering of words, 10
partition, 32
permutations, 21, 27, 31
pointwise ordering, 4
polynomial space, 8, 53
polynomial time, 3, 6, 19, 52, 61, 62, 64, 90
is recursively presentable, 23
pools, 2
coding, 7
columns, 30
cones, *see* cones
definition of, 3
eigenpool, 30, 37
useless, 4
positive reductions, 61

positive truth-table reductions, 65, 86
positive Turing machine, 61
prefix ordering of words, 10
preorder traversal, 67
product of Boolean functions, 71, 74
projections, 22, 30
Ψ-cones, *see* cones
Ψ-reductions, 73
  definition of, 63

R

reachability problem, 16
real numbers, 10
recursion theory, vi, 16, 24, 46
recursively computable, 46, 48
recursively presentable, 19, 23, 28, 36, 70, 74, 76, 82, 84–87, 89
  and universal functions, 25
  definition of, 23
  polynomial time, 23
reduction closures, *see* reductions
reduction trees, 67
reductions
  bounded Turing, 66
  cardinality, 66
  definition of closure, 64
  many-one, 60, 65
  one to one, 75
  parity, 65
  positive, 61
  Ψ-reduction, *see* Ψ-reductions
  truth-table, *see* truth-table reductions
Reingold, Edward M., 79, 87
remainder pools, 49
representative systems, 32–33, 36
resource help, 24
Robertson, Edward L., 23, 28
Ronneburger, Detlef, 5, 57, 95
root, 67
Rose, Gene F., vi, vii, 16

S

SAT, *see* satisfiability problem
satisfiability problem, 11–14, 59, 66
Schöning, Uwe, 24, 28
SEL, *see* selective families
selections, 22, 30, 35
selective families, 51
  and circuit value, 16
  and Dedekind cuts, 9
  and reachability, 16
  and satisfiability, 12, 66
  as cones, 80
  definition of, 5
  hierarchy of closures, 82, 84
  parity reductions, 84
  Turing reductions, 83
  upward translation, 37
self-avoiding walk, 79
self-reducible languages, 13
Selman, Alan L., 9, 10, 17, 63, 64, 69, 76
  (1979), 5, 12, 17, 29, 37, 41
  (1982b), 17, 60, 61
semirecursive, 10, 51
separating closures, 82
separating partial information, 26–28, 50–51
Separation Lemma, 38
set theory, 41
Sheu, Ming-Jye, 87
single query reductions, 76
SIZE, *see* size families
size families
  and cones, 71, 89
  and reductions, 89
  definition of, 4
  hard remainders, 49
SMC, *see* strongly membership comparable
smooth functions, 64, 83, 84, 90
sparse language, 91
stable classes, 61
stable families, 51
  definition of, 46
stacking notation, 30
standard left cuts, 10
standard tree, 54, 55
Stephan, Frank, 17, 57
  (1992), 18, 46, 50, 56
  (1994), x, 2, 11, 12, 14, 17, 77, 88, 90, 92
strongly membership comparable, 12
structural complexity analysis, 59
subbasis, 10
subset closed families, 20, 51
subset complete, *see* subset closed
superfluous pools, 21
symmetric difference, 53
symmetric Gray code, 80
Szelepcsényi, Róbert, 16

T

Thierauf, Thomas, 65, 69

Toda, Seinosuke, 87
TOP, *see* top families
top families, 12, 37
topological closure, 20
topology on Hamming spaces, 4
topology on partial orderings, 10
Torenvliet, Leen, 57, 76
Trakhtenbrot, Boris A., vi, 57
transition sequence, 79
trees, 54, 67
truth-table reductions, 62, 66, 85
  bounded, 65
  cardinality, 66
  conjunctive, 65
  disjunctive, 65
  parity, 65
  positive, 65
Turing evaluation types, 68
Turing reductions, 60, 66, 83

U

uncountable cardinality, 10
Uniform Diagonalisation Theorem, 24, 28
Unique Normal Form Theorem, 28
unit posets, 32
  figure for index three, 44
  figure for index two, 32
units, 32, 42
universal functions, 24
unsatisfiability problem, 59
upward translation, 34–37, 88, 90
  approximable families, 39
  bottom and top families, 37
  cheatable families, 38
  minimum families, 37
  selective families, 37
useless pools, 3, 4, 11

V

van Emde Boas, Peter, 76
variables for families, 3
verum, 63

W

Wagner, Klaus W., 69
walks, 79, 80, 82, 83, 89
Watanabe, Osamu, 87
weak bases, *see* bases
weak minimum families, 6
weak unit posets, *see* unit posets
weak units, *see* units
weakly normal, *see* normal families
WEAKMIN, *see* weak minium families
Wechsung, Gerd, 40, 41, 66, 69, 87
well-founded relations, 40

Y

Young, Paul, 92