Structural Properties of NP-Hard Sets and Uniform Characterisations of Complexity Classes

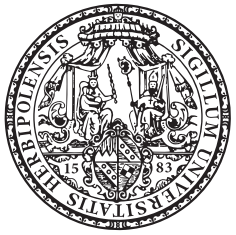
Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Julius-Maximilians-Universität Würzburg

vorgelegt von

Stephen Travers

aus Frankfurt am Main

Würzburg, 2007



Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik

Eingereicht am: 6. Dezember 2007 bei der
Fakultät für Mathematik und Informatik,
Julius-Maximilians Universität Würzburg

1. Gutachter: Prof. Dr. Klaus W. Wagner
2. Gutachter: Prof. Dr. Edith Hemaspaandra
Tag der mündlichen Prüfung: 11. April 2008

Meinen Eltern.

Ich bedanke mich bei allen, die mich während der Entstehung dieser Arbeit unterstützt haben. Besonderer Dank gilt meinem Doktorvater Klaus W. Wagner für die sehr gute Betreuung, seine hilfreichen Anregungen sowie die hervorragende Arbeitsatmosphäre an seinem Lehrstuhl. Besonders hervorheben möchte ich auch Christian Glaßer, der mich gleich zu Beginn meiner Zeit am Lehrstuhl in die aktuelle Forschung eingebunden hat. Durch die Zusammenarbeit mit ihm und durch seinen schier unerschöpflichen Erfahrungsschatz habe ich sehr viel gelernt, und seine ständige Motivation und Unterstützung haben viel zum Gelingen dieser Arbeit beigetragen.

Meinen aktuellen und ehemaligen Kollegen Elmar Böhler, Daniel Meister und Christian Reitwießner danke ich dafür, dass sie dazu beigetragen haben, die Zeit am Lehrstuhl zu einer angenehmen und interessanten zu machen.

Schließlich möchte ich meinen Eltern und auch meinem Bruder danken, die mich stets in meinem Tun unterstützt haben und auf deren Vertrauen ich mich immer verlassen konnte. Meiner Frau Nora danke ich für alles. Im Zusammenhang mit der Entstehung dieser Arbeit danke ich ihr insbesondere für die Geduld und ihre permanente Unterstützung.

Der Konrad-Adenauer-Stiftung danke ich für die Gewährung eines Promotionsstipendiums.

Würzburg, im Dezember 2007 Stephen Travers

Contents

Introduction

Computational complexity theory is a branch of theoretical computer science. The central question posed in this scientific discipline is the question of how difficult concrete algorithmic problems are.

What is a concrete algorithmic problem? Generally, there are few limitations, as long as the problem's definition is mathematically sound and unambiguous. The following are classical examples.

1. Given a map, what is the optimal route from one city A to another city B?
2. Given a natural number, what is its greatest prime factor?
3. Given a formula in propositional calculus with n variables, is there an assignment of truth-values to the variables such that the whole expression evaluates to true?
4. Given a map, a delivery truck stacked with parcels, and a list of n locations on the map where parcels are to be delivered. What is the minimum distance of this round trip?
5. Given a list of n natural numbers, and a natural number m. Is there a subset of these numbers whose sum is precisely m?

When we talk about the difficulty of algorithmic problems, the term *difficult* does not refer to the difficulty of developing an algorithm to solve the problem. Instead, it refers to the amount of resources needed to algorithmically compute the solution for a given problem instance.

Problems 3 and 5 are *decision problems*. Given a problem instance x, the answer is either "yes" or "no". On the other hand, algorithmic problems are often optimisation problems, like the problems 1 and 4 in the list above. Given a problem instance x, the answer to x is a number y. The same holds true for problem 2. These problems are *function problems*.

The major part of computational complexity theory (or complexity theory, for short) deals with decision problems, and so does this thesis. Interestingly, this is not a real restriction, because function problems can in fact be simulated by decision problems.

Let us assume we have an algorithm $\mathcal{A}$ which solves the following decision problem with a certain amount of resources:

"Given natural numbers n and k, does n have a prime factor greater than k?"

By doing a binary search on k and running the algorithm $\mathcal{A}$ multiple times with different k's, we can solve problem 2 without much more effort than $\mathcal{A}$ requires.

From a mathematical point of view, a decision problem is a set, and solving the yes-no question for a given instance x is nothing more than deciding whether x belongs to the set. For example, problem 3 can be represented by the set

$$\{H \mid H \text{ is a formula in propositional calculus and } H \text{ is satisfiable}\}.$$

It remains to clarify how the resources consumed by algorithms are measured. Typically, the scarce resources are time and space, and the complexity is measured with respect to the size of the problem instance. Hence, the central question in complexity theory can be restated as follows:

"As the size of the problem instance x for a set A increases, how do the running time or memory requirements to solve the problem whether x belongs to A change?"

Complexity classes are families of problems that have a similar asymptotic behaviour. One of the most prominent examples is the complexity class P, which comprises all decision problems that can be solved by deterministic Turing machines in polynomial time. For instance, Dijkstra's algorithm [Dij59] solves the shortest path problem in polynomial time, so the decision version of problem 1 is in P. For the other problems in our list, no polynomial algorithms are known. However, it does *not* follow that the problems are not in P, because it could be the case that better algorithms do exist and we just do not know of them yet. It is easy to see that all problems in our list (or their decision version) are in EXP, the class of decision problems solvable in deterministic exponential time.

An algorithm which decides a set A implies an upper bound for the complexity of A. Complexity classes usually consist of problems which share the same upper bound. Proving lower bounds is often much more difficult than proving upper bounds. While one algorithm suffices to prove an upper bound, a lower bound is a statement that no algorithm whatsoever can solve the problem with fewer resources. Up to the present day, no super-polynomial lower bound is known for any of the problems 2-5.

However, there are other means of defining complexity classes which allow a more appropriate classification of the complexity of decision problems. If we take a closer look at the problems 3 and 5, it becomes apparent that both problems share a remarkable property:

Let us analyze problem 5 in more detail. It seems difficult to solve the problem, because there are 2^n possible subsets. Hence, testing each single subset whether it sums up to m

is not possible in polynomial time (however, there could be more sophisticated methods which significantly reduce the search space). In contrast to that, given a potential solution to the problem, i.e., a subset consisting of say $k < n$ natural numbers, then it is easy to verify whether this subset sums up to m, this only requires $k-1$ additions. Both problems 3 and 5 have the property that verifying a potential solution is easy (more precisely, can be done in polynomial time) while it is not clear how to find the correct solution in the exponentially large search space.

Problems with this property form the class NP. By its definition, it is evident that $\mathrm{P} \subseteq \mathrm{NP}$. Whether $\mathrm{P} = \mathrm{NP}$ or $\mathrm{P} \subset \mathrm{NP}$ holds is the most important open question in theoretical computer science, this question is called the P-NP problem. In this connection, the notion of *reduction* plays a crucial role. A problem A reduces to a problem B if any decision algorithm for B can be used to solve a problem A without much additional effort. In other words, A reduces to B if A is at most as difficult as B. In fact, problems 3 and 5 provably belong to the most difficult problems in NP, the so-called NP-complete problems.

In spite of the fact that NP-complete problems are the most difficult problems in NP, the existence of a polynomial-time algorithm for a single NP-complete problem would imply $\mathrm{P} = \mathrm{NP}$. However, most researchers in theoretical computer science strongly believe that such an algorithm does not exist.

Numerous problems of great practical importance are NP-complete. For instance, the Traveling Salesman Problem (which is closely related to problem 4 in our list[1]) is a central question in Operations Research. The same holds true for other optimisation problems like scheduling, and many more.

Complexity theory investigates the scalability of computational problems and algorithms. By doing so, it places practical limits on what computers can accomplish. Furthermore, complexity theory also gives partial or complete answers to philosophical questions like "is finding a correct solution more difficult than verifying whether a given, potential solution is correct?" (the P-NP problem). In the same spirit, the question "can two people communicate securely without ever having the opportunity to meet personally or use a secure channel?" is also closely related to complexity theory (public-key cryptography). There are many other examples of this kind. For this reason, computational complexity theory is particularly exciting.

The first part of this thesis is devoted to the study of NP-complete-, and more generally, NP-hard problems. It aims at improving our understanding of this important class of problems. The second part studies complexity theory in a more general sense. There we focus on the description of complexity classes and analyse uniform frameworks which allow us to characterise a great variety of important complexity classes.

[1] Actually, the decision version of problem 4 is NP-*hard* which means that it is at least as difficult as any NP-complete problem. However, it is unknown whether it is in NP, because it is not known how the *minimality* of a given solution can be checked in polynomial time.

Outline of the Thesis

Part I

The first part of this thesis comprises Chapters 2, 3, and 4. These chapters are devoted to structural properties of NP-hard sets. In Chapters 2 and 3 we investigate various ways of altering NP-hard sets and analyse how this affects NP-hardness, or more generally, the complexity of the resulting set.

Motivated by the recent discovery that all NP-complete sets can in fact be altered in a rather extensive way (they can be split into two equivalent parts [GPSZ05]), Chapter 4 focuses on the question of whether all NP sets have this property.

Part II

The second part of this thesis comprises Chapters 5, 6, and 7. While Part I deals with structural properties of complexity classes and their hard problems, Part II is also dedicated to complexity classes, but from a different perspective:

In some sense, after investigating the interior of complexity classes in Part I, the focus shifts to the description of complexity classes and thereby to the exterior in Part II.

Chapter 5 gives a short introduction to the concept of leaf languages, a uniform way to describe complexity classes. Chapter 6 is devoted to the connections the leaf-language approach establishes between complexity theory and the theory of formal languages, and Chapter 7 introduces a new leaf-language model which allows us to prove further connections between the two research fields.

Synopses of the Individual Chapters

Chapter 1

The first chapter is an introductory chapter where we recapitulate basic notions and notations in theoretical computer science. We start with basic mathematical notations and then proceed to the principles of computational complexity theory.

We briefly introduce Turing machines and explain how different complexity classes can be described by restricting the amount of resources Turing machines may consume while solving problems. Then follows an overview of well and less well known complexity classes which we will address in the subsequent sections.

We then glimpse at the important notions of reducibility and relativisation before we conclude the chapter with a few fundamental notions of recursion theory.

Chapter 2

In this chapter, we systematically study the complexity of unions of disjoint, equivalent sets. This research is motivated by the still open question [Sel88] of whether the union of two disjoint NP-complete sets is always NP-complete. Before we study the situation within NP, we glance at the situation in the recursion theory setting and observe that the union of two disjoint RE-complete sets is always RE-complete.

We then explain that one cannot expect similar absolute results for NP. Hence all one can hope for is results under reasonable hypotheses. We proceed by giving necessary and sufficient conditions for an affirmative answer to the main open question. The former turn out to be very likely to hold true while the latter imply rather unlikely consequences. We resist the temptation to speculate on what the answer to the main question is and approach the problem from another direction.

We prove that under reasonable assumptions, the union of two disjoint NP-complete sets cannot become too easy. More precisely, we show that the union of two disjoint NP-complete sets belongs to the class High_1, the first level of Schöning's high hierarchy [Sch83]. We give further evidence that unions of disjoint NP-complete sets are not far from being NP-complete by showing that under a reasonable assumption, the union of an NP-complete set with a disjoint set in NP is nonuniformly NP-complete.

We then abstract from the main problem and study the more general question of how the complexity of unions of disjoint, equivalent sets can change. As a useful tool, we introduce the notion of m-idempotence. By their definition, m-idempotent sets have the property that their m-degrees are closed under unions of disjoint sets, so Selman's question translates to the question of whether SAT, the satisfiability problem for Boolean formulas, is m-idempotent. In order to show that NP$-$P does contain m-idempotent sets, we prove that every p-selective set is m-idempotent. It follows readily that if NE $\neq$ coNE, then there exists $A \in$ NP$-$coNP such that A is m-idempotent. Although this leaves open whether the sets in the highest degree of NP, i.e., the NP-complete sets are m-idempotent, it does show that NP is likely to contain degrees which have that property.

In contrast to that, we show in the remainder of the chapter that NP$-$P also contains degrees with opposite properties: We show that NP$-$P contains disjoint, equivalent sets whose union can be arbitrary simple unless P $=$ NP $\cap$ coNP. Moreover, if the polynomial hierarchy is strict, then it is possible for the union of two disjoint NP(2)-sets to be harder than either of its components. Furthermore, we explain that under a strong hypothesis, the same can also happen within NP$-$coNP.

Chapter 3

In this chapter, we pursue the track taken in Chapter 2 where we show that when NP-complete data is added to an NP-complete set then the resulting set retains much of its complexity, i.e., it is NP-complete with respect to more general reducibilities.

We now analyse what happens when we alter NP-hard sets in a more general way. By dropping the condition that the data we add must be NP-complete itself, our focus shifts to the more general question of how NP-hard sets can cope with faulty data. This includes adding faulty data (false positives) to NP-hard sets, or removing reasonable data (false negatives).

We investigate how polynomial time reductions can handle combinations of both, false positives and false negatives. This relates our research to the notion of *program self-correction* which was introduced by Blum, Luby, and Rubinfeld [BLR93]. That notion addresses a fundamental question regarding software reliability: Can one increase the reliability of existing software without understanding the way it works?

We show that $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard and $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard sets do not become too easy when false positives are added (as they stay NP-hard with respect to more general reducibilities). Also, $\leq^{\mathrm{p}}_{\text{2-dtt}}$-hard sets stay hard with respect to a more general reducibility when they encounter a sparse amount of false negatives. On the other hand, we show that unless P = NP, there exist sparse sets S_1, S_2 such that $\mathrm{SAT} \cup S_1$ is not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for NP, and $\mathrm{SAT} \cup S_2$ is not $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for NP.

Chapter 4

After investigating questions concerning the complexity of unions of sets in the previous two chapters, this chapter focuses on the inverse question. Given *one* set A, can this set be partitioned into two sets each satisfying certain properties? This question is captured by the notion of *mitoticity*, a notion originally introduced by Lachlan [Lac67] and Ambos-Spies [Amb84]. A set A is *m-mitotic* if there is a set $S \in \mathrm{P}$ such that A, $A \cap S$, and $A \cap \overline{S}$ are all polynomial-time m-equivalent. Informally, if a set is m-mitotic, it can be split into two parts that both have the same complexity as the original set. More generally, r-mitoticity is defined analogously for other polynomial-time reducibilities r.

It is known that all m-complete sets for NP are m-mitotic [GPSZ05], hence it is a natural question to ask whether there exist *non-mitotic* sets in NP, i.e., sets that cannot be split. In this chapter we study the question of the existence of non-mitotic sets in NP. This is a nontrivial question, because there are no natural examples of non-mitotic sets. As all NP-complete sets are m-mitotic and nontrivial sets belonging to the class P are m-mitotic,

any unconditional proof of the existence of non-mitotic sets in NP would prove at the same time that P $\neq$ NP. This implies that we cannot expect unconditional results in this area.

We prove that if EEE $\neq$ NEEE $\cap$ coNEEE, then there exists an $L \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$ that is not m-mitotic but in fact is 1-tt-mitotic. From this, it follows that under the same hypothesis, (NP $\cap$ coNP)$-$P contains a non-mitotic set and that 1-tt-reducibility and m-reducibility differ on sets in NP. This consequence explains the need for a reasonably strong hypothesis. However, we also show that 1-tt-reducibility and m-reducibility separate within NP under the weaker hypothesis that E $\neq$ NE $\cap$ coNE.

The last result in this chapter gives evidence of non-T-mitotic sets in NP. It states that if EEE $\neq$ NEEE, then there exists a set $C \in \mathrm{NP}-\mathrm{P}$ such that C is not T-mitotic.

Chapter 5

This is the first chapter of the second part of this thesis. We give a brief introduction to the leaf-language concept which was introduced by Bovet, Crescenzi, and Silvestri [BCS92] and independently by Vereshchagin [Ver93]. After defining leaf-language complexity classes we show how prominent complexity classes can be characterised by leaf languages. We then give an overview of the known connections which leaf languages establish between complexity theory and formal languages. We conclude this introductory chapter by explaining how the leaf-language approach allows concise oracle separations of leaf-language definable complexity classes.

Chapter 6

The key to the concise oracle constructions via leaf languages mentioned in the last chapter is the so-called BCSV-theorem which states that a language L_1 is polylog-time reducible (plt-reducible) to a language L_2 if and only if $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1)$ is robustly contained in $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_2)$. We explain that for this equivalence it is crucial that balanced leaf-language classes are used, because the theorem does not hold for the unbalanced model. This problem was recently solved by introducing polynomial-time tree reducibility (ptt-reducibility, for short) [Wag04b], an analogue of plt-reducibility for unbalanced leaf languages. Ptt-reducibility admits a BCSV-theorem for unbalanced leaf-language classes.

In this chapter, we analyse this new reducibility and prove that restricted to regular languages, the levels 0, 1/2, 1, and 3/2 of the dot-depth hierarchy are closed under ptt-reducibility.

Moreover, we explain that these results indicate that the connection between dot-depth and polynomial-time hierarchy is closer than formerly known: We show that on the lower

levels, the dot-depth and the polynomial-time hierarchy perfectly correspond, i.e., that formal languages with a certain dot-depth precisely characterise a certain level of the polynomial-time hierarchy.

Chapter 7

In this chapter we offer a useful completion of the known leaf-language concepts. This is the concept of ε-leaf languages. It is inspired by the observation that rejecting paths of nondeterministic computations act as neutral elements. In this sense we allow nondeterministic transducers not only to output single letters but also to output the empty word ε which is the neutral element of Σ^*.

We explain that this approach offers several advantages over the known balanced- and unbalanced concepts. After the definition of ε-leaf languages and a few examples we prove that this new model allows us to establish a tight connection between the polynomial-time hierarchy and the Straubing-Thérien hierarchy, another very important hierarchy of starfree languages.

We prove that the lower levels of the Straubing-Thérien hierarchy and the polynomial-time hierarchy perfectly correspond, and we also provide a precise characterisation of the class 1NP.

In this sense, we argue that the ε-leaf-language approach combines the advantages of the other two leaf-language notions.

Publications

This thesis consists of recently published results as well as unpublished results. The former have appeared in the following refereed conference proceedings or journals.

[GSTW07] C. Glaßer, A. L. Selman, S. Travers, and K. W. Wagner. *The Complexity of Unions of Disjoint Sets.* In *Proceedings 24th Symposium on Theoretical Aspects of Computer Science*, volume 4393 of *Lecture Notes in Computer Science*, pages 248–259. Springer Verlag, 2007

[GSTZ07] C. Glaßer, A. L. Selman, S. Travers, and L. Zhang. *Non-Mitotic Sets.* In *Proceedings 27th Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4855 of *Lecture Notes in Computer Science*, pages 146–157. Springer Verlag, 2007

[GTW06] C. Glaßer, S. Travers, and K. W. Wagner. *Perfect Correspondences Between Dot-Depth and Polynomial-Time* Hierarchy. In *Proceedings 10th Conference Developments in Language Theory*, volume 4036 of *Lecture Notes in Computer Science*, pages 408–419. Springer Verlag, 2006

[GT07] C. Glaßer and S. Travers. *Machines That Can Output Empty Words.* Accepted to appear in *Theory Of Computing Systems*, Springer Verlag

Conference version appeared in: *Proceedings 31st Symposium on Mathematical Foundations of Computer Science*, volume 4162 of *Lecture Notes in Computer Science*, pages 436–446. Springer Verlag, 2006

The author of this thesis is a senior author of all publications listed above. Several results in these papers were obtained in joint work with coauthors. This thesis however consists of results which are predominantly or solely due to the author of this thesis. In general, results due to coauthors were omitted.

Chapter 1

Preliminary Notions, Basic Notations, and Definitions

In this chapter we recapitulate basic notions and notations in theoretical computer science. We assume familiarity with standard mathematical notations.

1.1 Numbers, Words, Sets, and Operators

$\mathbb{N}$ denotes the set of all natural numbers including zero. $\leq$ denotes the standard order on $\mathbb{N}$. A subscript $_{\text{ae}}$ indicates that the statement does not have to hold for finitely many exceptions. The set of all polynomials in one variable is denoted by pol.

Throughout the thesis, Σ denotes a finite alphabet with at least two letters, Σ^* denotes the set of all words over Σ, and $|w|$ denotes the length of a word $w \in \Sigma^*$. We denote the empty word with ε. For $n \geq 0$, Σ^n denotes the set of all words of length n, and $\Sigma^{\leq n}$ denotes the set of all words up to length n. For a set A, $\#A$ denotes the cardinality of A.

We use the natural correspondence between words and numbers; when we consider functions over natural numbers we assume that the input is given in binary coding.

For $L \subseteq \Sigma^*$ and $a \in \Sigma$, $aL =_{\text{def}} \{aw \,|\, w \in L\}$.

A set $A \subseteq \Sigma^*$ is *nontrivial* if $A \neq \emptyset$ and $A \neq \Sigma^*$.

For a set $L \subseteq \Sigma^*$, $\overline{L} = \{x \,|\, x \in \Sigma^*, x \notin L\}$ denotes the *complement* of L.

For any set A, the *power set* of A is denoted by $\mathcal{P}(A) =_{\text{def}} \{B \,|\, B \subseteq A\}$.

The *difference* of sets A and B is defined as $A-B =_{\text{def}} \{x \in A \,|\, x \notin B\}$.

The *symmetric difference* of sets A and B is defined as $A \triangle B =_{\text{def}} (A - B) \cup (B - A)$.

The *census function* of a set S is defined as $\text{census}_S(n) =_{\text{def}} |S \cap \Sigma^n|$.

A set S is *sparse* if there exists a polynomial p such that for all $n \geq 0$, $\text{census}_S(n) \leq p(n)$.

A *tally set* is a subset of 0^*.

Let $A \subseteq \Sigma^*$ be a set. The *characteristic function* $c_A : \Sigma^* \to \{0,1\}$ of A is defined as follows:

$$c_A(x) =_{\text{def}} \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases}$$

The *quasicharacteristic function* $\chi_A : \Sigma^* \to \{0,1\}$ of A is defined as follows:

$$\chi_A(x) =_{\text{def}} \begin{cases} 1, & \text{if } x \in A \\ \text{undefined}, & \text{otherwise} \end{cases}$$

For a finite alphabet Σ, the *initial word relation* $\sqsubseteq$ on Σ^* is defined by

$$u \sqsubseteq v \underset{\text{def}}{\iff} \exists w (w \in \Sigma^* \wedge uw = v).$$

We write $u \sqsubset v$ if and only if $u \sqsubseteq v$ and $u \neq v$.

The *lexicographical order* on $\{0,1\}^*$ is defined by

$$x \leq y \underset{\text{def}}{\iff} x \sqsubseteq y \vee \exists u (u0 \sqsubseteq x \wedge u1 \sqsubseteq y).$$

As we use the same notation for natural numbers, we will ensure that it is always clear from the context which of the two we mean.

The *subword relation* is denoted by $\preceq$. It is defined by

$$v \preceq w \underset{\text{def}}{\iff} v = v_1 \dots v_n,\ v_1, \dots, v_n \in \Sigma,\ w \in \Sigma^* v_1 \Sigma^* v_2 \dots \Sigma^* v_n \Sigma^*.$$

We write $v \prec w$ if $v \preceq w$ and $v \neq w$.

For $k \geq 0$ we write $v \preceq_k w$ if v is a nonempty word that appears precisely k-times as a subword of w. In addition we define $\varepsilon \preceq_1 w$ for every word w.

For $k \geq 0$ we write $v \preceq_{\geq k} w$ if there exists $l \geq k$ such that $v \preceq_l w$.

For $k \geq 0$ and a finite set B of words $v_1, \dots, v_{|B|}$ we write $B \preceq_k w$ if k can be written as $k = k_1 + \dots + k_{|B|}$ such that

$$v_1 \preceq_{k_1} w,\ \ v_2 \preceq_{k_2} w,\ \ \dots,\ \ v_{|B|} \preceq_{k_{|B|}} w.$$

So $v \preceq w$ if and only if there exists $k \geq 1$ such that $v \preceq_k w$. Also, $v \not\preceq w$ if and only if $v \preceq_0 w$.

Regular languages are built up from the empty set $\emptyset$ and singletons $\{a\}$ for $a \in \Sigma$ using Boolean operations, concatenation, and iteration. REG is the class of all regular languages.

We will also consider *starfree languages* which represent a subclass of the regular languages where iteration is not allowed. SF is the class of all starfree languages.

The class of regular languages is precisely the class of languages which can be accepted by finite automata. Moreover, for every regular language L there exists a unique minimal automaton which accepts L. A very nice introduction to this topic is given in [HMU07].

1.2 Principles of Computational Complexity Theory

1.2.1 Turing Machines

Throughout the thesis, we use the concept of deterministic and nondeterministic multi-tape Turing machines. A formal definition of these computation models can be found in any textbook on computational complexity, e.g. [Pap94].

We use two different kinds of Turing machines. 'Normal' Turing machines do not have an output tape but accept/reject an input via accepting and rejecting states.

A deterministic Turing machine M decides a set $A \subseteq \Sigma^*$ if the following holds for all $x \in \Sigma^*$:

$$\begin{aligned} x \in A \quad &\Rightarrow \quad M \text{ on input } x \text{ halts after a finite number of steps.} \\ &\qquad \text{When it halts it is in an accepting state.} \\ x \notin A \quad &\Rightarrow \quad M \text{ on input } x \text{ halts after a finite number of steps.} \\ &\qquad \text{When it halts it is in a rejecting state.} \end{aligned}$$

A nondeterministic Turing machine can split its computation paths in every step into at most two computation paths. Due to this mechanism, nondeterministic Turing machines produce *computation trees*. Each computation path of a machine M on an input x can be described by a word from $\{0,1\}^*$, where 0 stands for "branch left" and 1 stands for "branch right".

Consequently, accepting/rejecting works differently for such machines. A nondeterministic computation path is called accepting if the machine on this path halts after finitely many steps and is in an accepting state when it halts. A nondeterministic computation path

is called rejecting if the machine on this path halts after finitely many steps and is in a rejecting state when it halts.

A nondeterministic Turing machine M decides a set $A \subseteq \Sigma^*$ if the following holds for all $x \in \Sigma^*$:

$$\begin{aligned} x \in A &\Rightarrow \text{on input } x, M \text{ develops at least one accepting path.} \\ x \notin A &\Rightarrow \text{all paths of } M \text{ on input } x \text{ are rejecting paths.} \end{aligned}$$

Given a (deterministic or nondeterministic) Turing machine M, the *language accepted by* M is denoted by $L(M)$.

The second kind of Turing machine we consider is that of the *Turing transducer*. The difference is that a transducer has an output tape by which it can output symbols and hence compute functions.

A Turing transducer computes a function $f : \Sigma^* \to \Sigma^*$ if for all $x \in \Sigma^*$, the following holds:

- If $f(x)$ is defined, then M on input x halts after a finite number of steps. When it stops, all tapes are cleared (i.e., all cells contain the symbol $\Box$) except for the first working tape which contains $f(x)$.
- If $f(x)$ is not defined, then M does not halt on input x.

When talking about Turing machines and transducers, we will not always distinguish between the two terms. It will be clear from the context what kind of Turing machine we mean.

1.2.2 Complexity- and Function Classes

A *complexity class* is a subset of $\mathcal{P}(\Sigma^*)$. We will call the elements of a complexity class *languages*, *problems*, or simply *sets*. A complexity class $\mathcal{K}$ is *nontrivial* if $\mathcal{K} \notin \{\emptyset, \Sigma^*\}$.

Complexity classes are often defined by restricting the amount of resources Turing machines may consume while solving problems. In this thesis, the focus lies on time- and space-bounded computations.

Definition 1.1 *Let $f : \mathbb{N} \to \mathbb{N}$ be a monotonic increasing function.*

1. $\mathrm{DTIME}(f) =_{\mathrm{def}} \{A \subseteq \Sigma^* \mid$ there exists a deterministic Turing machine M such that M decides the set A in time $f\}$.

2. $\mathrm{DSPACE}(f) =_{\mathrm{def}} \{A \subseteq \Sigma^* \mid$ there exists a deterministic Turing machine M such that M decides the set A in space $f\}$.

3. $\mathrm{NTIME}(f) =_{\mathrm{def}} \{A \subseteq \Sigma^* \mid$ there exists a nondeterministic Turing machine M such that M decides the set A in time $f\}$.

4. $\mathrm{NSPACE}(f) =_{\mathrm{def}} \{A \subseteq \Sigma^* \mid$ there exists a nondeterministic Turing machine M such that M decides the set A in space $f\}$.

We define several standard complexity classes.

Definition 1.2 *The following complexity classes are all defined by specifying different time- or space bounds.*

1. $\mathrm{P} =_{\mathrm{def}} \mathrm{DTIME}(\mathrm{pol})$
2. $\mathrm{E} =_{\mathrm{def}} \mathrm{DTIME}(2^{\mathcal{O}(n)})$
3. $\mathrm{EXP} =_{\mathrm{def}} \mathrm{DTIME}(2^{\mathrm{pol}})$
4. $\mathrm{NP} =_{\mathrm{def}} \mathrm{NTIME}(\mathrm{pol})$
5. $\mathrm{NE} =_{\mathrm{def}} \mathrm{NTIME}(2^{\mathcal{O}(n)})$
6. $\mathrm{NEXP} =_{\mathrm{def}} \mathrm{NTIME}(2^{\mathrm{pol}})$
7. $\mathrm{L} =_{\mathrm{def}} \mathrm{DSPACE}(\log)$
8. $\mathrm{NL} =_{\mathrm{def}} \mathrm{NSPACE}(\log)$
9. $\mathrm{PSPACE} =_{\mathrm{def}} \mathrm{DSPACE}(\mathrm{pol})$

Definition 1.3 *The following are iterated-exponential-time analogues of the complexity classes* E*,* EXP*,* NE*, and* NEXP*. Let* $k > 1$*.*

1. $\underbrace{\mathrm{E}\ldots\mathrm{E}}_{k} =_{\mathrm{def}} \mathrm{DTIME}(2^{\cdot^{\cdot^{\cdot^{2^{\mathcal{O}(n)}}}}}\Big\}k)$
2. $\underbrace{\mathrm{E}\ldots\mathrm{E}}_{k}\mathrm{XP} =_{\mathrm{def}} \mathrm{DTIME}(2^{\cdot^{\cdot^{\cdot^{2^{\mathrm{pol}}}}}}\Big\}k)$
3. $\mathrm{N}\underbrace{\mathrm{E}\ldots\mathrm{E}}_{k} =_{\mathrm{def}} \mathrm{NTIME}(2^{\cdot^{\cdot^{\cdot^{2^{\mathcal{O}(n)}}}}}\Big\}k)$
4. $\mathrm{N}\underbrace{\mathrm{E}\ldots\mathrm{E}}_{k}\mathrm{XP} =_{\mathrm{def}} \mathrm{NTIME}(2^{\cdot^{\cdot^{\cdot^{2^{\mathrm{pol}}}}}}\Big\}k)$

Observe that some authors alternatively define these classes differently, e.g., EE is defined as being equal to $\mathrm{DTIME}(2^{O(2^n)})$.

Definition 1.4 *The following complexity classes are defined by altering the acceptance behaviour of Turing machines.*

1. *The class* 1NP *[GW86] (also called* US *[BG82]) is the class of languages* L *for which there exists a nondeterministic polynomial-time bounded Turing machine* M *such*

that an input x belongs to L if and only if M on input x has exactly one accepting path.

2. *For $k > 1$, the class* $\mathrm{Mod}_k\mathrm{P}$ *[CH90] is defined as the class of languages L for which there exists a nondeterministic polynomial-time bounded Turing machine M such that an input x belongs to L if and only if the number of accepting paths is not divisible by k.*

3. *The class* PP *[Gil77] is the class of languages L for which there exists a nondeterministic polynomial-time bounded Turing machine M such that an input x belongs to L if and only if M on input x produces more accepting than rejecting paths.*

Definition 1.5 *In the definitions of the following complexity classes we demand Turing machines to satisfy certain promises.*

1. *The class* UP *[Val76] is the class of languages L for which there exists a nondeterministic polynomial-time bounded Turing machine M such that for every input x, if $x \in L$, then the machine M produces precisely one accepting path. If $x \notin L$, then M produces rejecting paths only.*

2. *The class* BPP *[Gil77] is the class of languages L for which there exists a nondeterministic polynomial-time bounded Turing machine M such that on every input x, all paths of M have the same length, and whenever $x \in L$, then at least 2/3 of all paths of M accept. Whenever $x \notin L$, then at most 1/3 of all paths of M accept.*

Observe that the two latter classes are different from all other classes defined so far, because not all nondeterministic Turing machines can be used to accept their languages. Suitable machines for these classes have to keep a *promise*, namely that they never output more than one accepting path (as in the case of UP), or always accept/reject with a clear majority (as in the case of BPP). For this reason, classes of this kind are called *promise classes*.

FP denotes the class of total functions computable in deterministic polynomial time.

FP/poly is the superclass of FP that consists of all functions f for which there exists a total function $a : 0^* \to \Sigma^*$ such that

- there exists a polynomial p such that for all n, $|a(0^n)| \leq p(n)$, and
- there exists a $g \in \mathrm{FP}$ such that for all x, $f(x) = g(x, a(0^{|x|}))$.

The function a is called the *advice* function.

The standard Boolean operations can be extended to complexity classes in a natural way:

Definition 1.6 *Let $\mathcal{C} \subseteq \mathcal{P}(\Sigma^*)$ and $\mathcal{D} \subseteq \mathcal{P}(\Sigma^*)$ be complexity classes. We define*

$$\begin{aligned}
\mathrm{co}\mathcal{C} &=_{\mathrm{def}} \{\overline{A} \,\big|\, A \in \mathcal{C}\} \\
\mathcal{C}\vee\mathcal{D} &=_{\mathrm{def}} \{A \cup B \,\big|\, A \in \mathcal{C}, B \in \mathcal{D}\}, \\
\mathcal{C}\wedge\mathcal{D} &=_{\mathrm{def}} \{A \cap B \,\big|\, A \in \mathcal{C}, B \in \mathcal{D}\}, \\
\mathcal{C} \oplus \mathcal{D} &=_{\mathrm{def}} \{A \triangle B : A \in \mathcal{C}, B \in \mathcal{D}\} \\
\mathcal{C}\dot{\vee}\mathcal{D} &=_{\mathrm{def}} \{A \cup B \,\big|\, A \in \mathcal{C}, B \in \mathcal{D}, A \cap B = \emptyset\} \\
\mathcal{C}\dot{\wedge}\mathcal{D} &=_{\mathrm{def}} \mathrm{co}(\mathrm{co}\mathcal{C}\dot{\vee}\mathrm{co}\mathcal{D}).
\end{aligned}$$

Notice that the union of disjoint sets used here is not the same concept as the marked union which is sometimes denoted by $\dot{\cup}$. The reason is that the latter leads to unions of disjoint p-separable sets, which does not have to be the case with $\dot{\vee}$. For instance, for all sets $A, B \in$ 1NP, it holds that $A\dot{\cup}B = 0A \cup 1B \in$ 1NP, implying that 1NP is closed under $\dot{\cup}$. Contrary to that, there exists an oracle relative to which 1NP$\dot{\vee}$1NP $\neq$ 1NP [GT07].

Definition 1.7 *For a class of languages $\mathcal{C}$ which is closed under union and intersection, the Boolean hierarchy over $\mathcal{C}$ [WW85] is the family of classes $\mathcal{C}(k)$ and $\mathrm{co}\mathcal{C}(k)$ where $k \geq 1$,*

$$\begin{aligned}
\mathcal{C}(k) &=_{\mathrm{def}} \overbrace{\mathcal{C} \oplus \mathcal{C} \oplus \cdots \oplus \mathcal{C}}^{k \text{ times}}, \text{ and} \\
\mathrm{co}\mathcal{C}(k) &=_{\mathrm{def}} \left\{\overline{L} : L \in \mathcal{C}(k)\right\}.
\end{aligned}$$

The properties of Boolean hierarchies were studied by Köbler, Schöning, and Wagner [KSW87], and Cai et al. [CGH$^+$88].

We define several operators which can be applied to complexity classes. $\exists!\cdot$ and $\forall!\cdot$ are the unique variants of $\exists\cdot$ and $\forall\cdot$, respectively. The semantics are as follows: $\exists!\cdot$ stands for "there exists precisely one" while $\forall!\cdot$ stands for "there exists at most one exception".

Definition 1.8 *Let $\mathcal{D}$ be a complexity class. A language L belongs to the class $\exists\cdot\mathcal{D}$ if there exist a polynomial p and $B \in \mathcal{D}$ such that:*

$$\begin{aligned}
x \in L &\Rightarrow \exists y\big(|y| \leq p(|x|) \wedge (x,y) \in B\big) \\
x \notin L &\Rightarrow \forall y\big(|y| \leq p(|x|) \rightarrow (x,y) \notin B\big)
\end{aligned}$$

A language L belongs to the class $\exists!\cdot\mathcal{D}$ if there exist a polynomial p and $B \in \mathcal{D}$ such that:

$$\begin{aligned}
x \in L &\Rightarrow \exists! y\big(|y| \leq p(|x|) \wedge (x,y) \in B\big) \\
x \notin L &\Rightarrow \forall y\big(|y| \leq p(|x|) \rightarrow (x,y) \notin B\big)
\end{aligned}$$

A language L belongs to $\forall\cdot\mathcal{D}$ if there exist a polynomial p and $B \in \mathcal{D}$ such that:

$$\begin{aligned} x \in L &\Rightarrow \forall y\big(|y| \leq p(|x|) \rightarrow (x,y) \in B\big) \\ x \notin L &\Rightarrow \exists y\big(|y| \leq p(|x|) \wedge (x,y) \notin B\big) \end{aligned}$$

A language L belongs to $\forall!\cdot\mathcal{D}$ if there exist a polynomial p and $B \in \mathcal{D}$ such that:

$$\begin{aligned} x \in L &\Rightarrow \forall y\big(|y| \leq p(|x|) \rightarrow (x,y) \in B\big) \\ x \notin L &\Rightarrow \exists! y\big(|y| \leq p(|x|) \wedge (x,y) \notin B\big) \end{aligned}$$

1.2.3 Relativisation

Relativisation is a concept which formalises the idea of subroutine- or procedure calls in programming languages. In computational complexity theory, this concept is modeled by the use of *oracle turing machines*. An oracle Turing machine (OTM) M is a normal Turing machine which differs from a normal Turing machine in that it has a *query tape* and three special states z_{ask}, z_{yes}, and z_{no}. The query tape is a write-only tape.

A computation of M on input x with oracle $O \subseteq \Sigma^*$ then goes as follows: M acts as a normal Turing machine, but when it reaches the state z_{ask} the following happens: Let q be the string written on the query tape. If $q \in O$, then the computation immediately continues in state z_{yes}, if $q \notin O$, then the computation continues in state z_{no}. Simultaneously, the query tape is cleared. All this happens in one single computation step, i.e., the idea is that M is provided with some subroutine which can decide in one step whether any given string belongs to O or not.

With this concept it is possible to define relativised variants of standard complexity classes.

Let $O \subseteq \Sigma^*$. By considering languages accepted by nondeterministic polynomial-time oracle Turing machines (NPOTM) with access to the oracle O, we obtain the class NP^O. Let $\mathcal{K}$ be any complexity class. Then

$$\text{NP}^{\mathcal{K}} =_{\text{def}} \bigcup_{O \in \mathcal{K}} \text{NP}^O.$$

The other Turing machine time- and space-bounded classes can be relativised in the same way. For an oracle Turing machine M and $O \in \Sigma^*$ the language decided by M with oracle O is denoted by $L(M^O)$.

Stockmeyer [Sto77] introduced the *polynomial-time hierarchy* (PH for short). The polynomial-time hierarchy (sometimes also called *polynomial hierachy*) is a hierarchy of complexity classes that generalise the classes P, NP and coNP to oracle machines. The levels of the PH are $\Delta_{\text{k}}^{\text{P}}$, $\Sigma_{\text{k}}^{\text{P}}$, and $\Pi_{\text{k}}^{\text{P}}$ for $k \geq 0$.

It is defined as follows:

$$\begin{aligned} \Sigma_0^{\mathrm{P}} = \Pi_0^{\mathrm{P}} = \Delta_0^{\mathrm{P}} &=_{\mathrm{def}} \mathrm{P}, \\ \Sigma_{\mathrm{n}+1}^{\mathrm{P}} &=_{\mathrm{def}} \mathrm{NP}^{\Sigma_{\mathrm{n}}^{\mathrm{P}}}, \\ \Pi_{\mathrm{n}+1}^{\mathrm{P}} &=_{\mathrm{def}} \mathrm{coNP}^{\Sigma_{\mathrm{n}}^{\mathrm{P}}}, \\ \Delta_{\mathrm{n}+1}^{\mathrm{P}} &=_{\mathrm{def}} \mathrm{P}^{\Sigma_{\mathrm{n}}^{\mathrm{P}}}. \end{aligned}$$

The following is an equivalent definition which is due to Wrathall [Wra77].

$$\begin{aligned} \Sigma_0^{\mathrm{P}} = \Pi_0^{\mathrm{P}} &=_{\mathrm{def}} \mathrm{P}, \\ \Sigma_{\mathrm{n}+1}^{\mathrm{P}} &=_{\mathrm{def}} \exists \cdot \Pi_{\mathrm{n}}^{\mathrm{P}}, \\ \Pi_{\mathrm{n}+1}^{\mathrm{P}} &=_{\mathrm{def}} \forall \cdot \Sigma_{\mathrm{n}}^{\mathrm{P}}. \end{aligned}$$

1.2.4 Reducibilities

A binary relation $\leq$ over $\mathcal{P}(\Sigma^*)$ is called a reducibility if it is a preorder, i.e., it is reflexive and transitive. The notion of reducibility is to compare sets according to their complexity. Assume we want to determine whether a string x belongs to a a set A. If we can decide membership in A by asking whether certain other strings belong to a set B, then every decision procedure for B immediately yields a decision procedure for A. We say that A is at most as hard as B, or A is not harder than B and denote this with $A \leq B$. According to how the set B is queried when we decide the set A, we distinguish between a variety of different reducibilities whose definitions we are about to give.

The focus of the studies in this thesis will lie on different kinds of *polynomial-time reducibilities*. However, we will also address recursive reducibilities which we will define in Section 1.3.

We recall standard polynomial-time reducibilities [LLS75].

Definition 1.9 *A set B* many-one-reduces *to a set C (*m-reduces *for short; in notation $B \leq_{\mathrm{m}}^{\mathrm{p}} C$) if there exists a total, polynomial-time-computable function f such that for all strings x,*

$$x \in B \;\Leftrightarrow\; f(x) \in C.$$

Definition 1.10 *A set B* Turing-reduces *to a set C (*T-reduces *for short; in notation $B \leq_{\mathrm{T}}^{\mathrm{p}} C$) if there exists a deterministic polynomial-time bounded oracle Turing machine M such that for all strings x,*

$$x \in B \;\Leftrightarrow\; M \text{ with } C \text{ as oracle accepts the input } x.$$

Definition 1.11 *A set B* 2-disjunctively truth-table-reduces *to a set C (*2-dtt-reduces *for short; in notation $B \leq^{\mathrm{p}}_{2-\mathrm{dtt}} C$) if there exists a total, polynomial-time-computable function $f : \Sigma^* \to \Sigma^* \times \Sigma^*$ such that for all strings x,*

$$x \in B \;\Leftrightarrow\; \text{at least one word from the pair } f(x) \text{ belongs to } C.$$

Definition 1.12 *A set B* truth-table-reduces *to a set C (*tt-reduces *for short; in notation $B \leq^{\mathrm{p}}_{\mathrm{tt}} C$) if there exists a deterministic polynomial-time bounded oracle Turing machine M that queries nonadaptively such that for all strings x,*

$$x \in B \;\Leftrightarrow\; M \text{ with } C \text{ as oracle accepts the input } x.$$

Definition 1.13 *A set B* disjunctively truth-table-reduces to a set C (*dtt-reduces for short; in notation $B \leq^{\mathrm{p}}_{\mathrm{dtt}} C$) if there exists a total, polynomial-time-computable function $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$ such that for all strings x,*

$$x \in B \;\Leftrightarrow\; f(x) \cap C \neq \emptyset.$$

Definition 1.14 *A set B* conjunctively truth-table-reduces to a set C (*ctt-reduces for short; in notation $B \leq^{\mathrm{p}}_{\mathrm{ctt}} C$) if there exists a total, polynomial-time-computable function $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$ such that for all strings x,*

$$x \in B \;\Leftrightarrow\; f(x) \subseteq C.$$

Definition 1.15 *A set B* bounded truth-table-reduces to a set C (*btt-reduces for short; in notation $B \leq^{\mathrm{p}}_{\mathrm{btt}} C$) if there exists a $k \geq 1$, a k-ary Boolean function α, and $g_1, \ldots, g_k \in \mathrm{FP}$ such that for all x*

$$x \in B \Leftrightarrow \alpha(c_C(g_1(x)), c_C(g_2(x)), \ldots, c_C(g_k(x))) = 1.$$

Definition 1.16 *A set B 1-tt reduces to C (in notation $B \leq^{\mathrm{p}}_{1-\mathrm{tt}} C$) if for some M, $B \leq^{\mathrm{p}}_{\mathrm{tt}} C$ via M and for all x, M queries the oracle C at most once.*

Similarly, we define 2-tt, and so on.

Definition 1.17 *A set B* non-uniformly many-one-reduces *to a set C (*non-uniformly m-reduces *for short; in notation $B \leq^{\mathrm{p/poly}}_{\mathrm{m}} C$) if there exists a total function $f \in \mathrm{FP/poly}$ such that for all strings x,*

$$x \in B \;\Leftrightarrow\; f(x) \in C.$$

We remark that non-uniform reductions are of interest in cryptography. There they are a model of an adversary who is capable of long preprocessing [BV97]. They also have applications in structural complexity theory. Agrawal [Agr02] and Hitchcock and Pavan [HP06] investigate non-uniform reductions and show under reasonable hypotheses that every many-one complete set for NP is also hard for length-increasing, non-uniform reductions.

Definition 1.18 *A set B* strongly nondeterministic Turing-reduces *to a set C [Lon78] (*snT-reduces *for short; in notation $B \leq^{\mathrm{p}}_{\mathrm{snT}} C$) if there exists a nondeterministic polynomial-time bounded oracle Turing machine M that on each computation path outputs exactly one symbol from $\{+, -, ?\}$ such that for all strings x,*

$$\begin{aligned} x \in B &\Rightarrow M^C \text{ on } x \text{ produces at least one } + \text{ and no } -, \\ x \notin B &\Rightarrow M^C \text{ on } x \text{ produces at least one } - \text{ and no } +. \end{aligned}$$

If $B \leq^{\mathrm{p}}_{\mathrm{m}} C$ and $C \leq^{\mathrm{p}}_{\mathrm{m}} B$, then we say that B and C are *many-one-equivalent* (*m-equivalent* for short, in notation $B \equiv^{\mathrm{p}}_{\mathrm{m}} C$). Similarly, we define equivalence for other reducibilities. A set B is *many-one-hard* (*m-hard* for short) for a complexity class $\mathcal{C}$ if every $B \in \mathcal{C}$ m-reduces to B. If additionally $B \in \mathcal{C}$, then we say that B is *many-one-complete* (*m-complete* for short) for $\mathcal{C}$. Similarly, we define hardness and completeness for other reducibilities. We use the term $\mathcal{C}$-complete as an abbreviation for m-complete for $\mathcal{C}$.

Lemma 1.19 *For any nontrivial complexity class $\mathcal{K} \subseteq \mathcal{P}(\Sigma^*)$, it holds that neither $\emptyset$ nor Σ^* are $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard for $\mathcal{K}$.*

Proof. Let $\mathcal{K}$ be a nontrivial complexity class. Hence there exists $L \in \mathcal{K}$ such that $L \neq \emptyset$ and $L \neq \Sigma^*$. However, for all $A \subseteq \Sigma^*$, $A \leq^{\mathrm{p}}_{\mathrm{m}} \Sigma^*$ holds if and only if $A = \Sigma^*$, and $A \leq^{\mathrm{p}}_{\mathrm{m}} \emptyset$ holds if and only if $A = \emptyset$. So neither $L \leq^{\mathrm{p}}_{\mathrm{m}} \Sigma^*$ nor $L \leq^{\mathrm{p}}_{\mathrm{m}} \emptyset$ holds. □

Definition 1.20 *Let A be a set, $\mathcal{C}$ be a complexity class and $r \in \{\mathrm{m}, \mathrm{T}, \mathrm{dtt}, \mathrm{ctt}, \text{1-tt}, \dots\}$ be some polynomial-time reducibility. The* reduction closure under r-reducibility *and the* r-degree *of A (resp., $\mathcal{C}$) are defined as follows.*

$$\begin{aligned} \mathcal{R}^{\mathrm{p}}_{\mathrm{r}}(A) &=_{\mathrm{def}} \{B \mid B \leq^{\mathrm{p}}_{\mathrm{r}} A\}, \\ \mathcal{R}^{\mathrm{p}}_{\mathrm{r}}(\mathcal{C}) &=_{\mathrm{def}} \bigcup_{A \in \mathcal{C}} \mathcal{R}^{\mathrm{p}}_{\mathrm{r}}(A), \\ \deg^{\mathrm{p}}_{\mathrm{r}}(A) &=_{\mathrm{def}} \{B \mid A \equiv^{\mathrm{p}}_{\mathrm{r}} B\}, \\ \deg^{\mathrm{p}}_{\mathrm{r}}(\mathcal{C}) &=_{\mathrm{def}} \bigcup_{A \in \mathcal{C}} \deg^{\mathrm{p}}_{\mathrm{r}}(A). \end{aligned}$$

A class $\mathcal{C}$ is closed under $\leq^{\mathrm{p}}_{\mathrm{r}}$ if $\mathcal{R}^{\mathrm{p}}_{\mathrm{r}}(\mathcal{C}) = \mathcal{C}$. It is easy to see that whenever a class $\mathcal{C}$ is closed under $\leq^{\mathrm{p}}_{\mathrm{r}}$, it also follows that $\deg^{\mathrm{p}}_{\mathrm{r}}(\mathcal{C}) = \mathcal{C}$.

1.2.5 Some Structural Properties

Definition 1.21 *Disjoint sets A and B are called* p-separable *if there exists a set $S \in \mathrm{P}$ (the separator) such that $A \subseteq S$ and $B \subseteq \overline{S}$.*

Definition 1.22 ([Sch83]) *A set $A \in$ NP is* high *for $\Sigma_{\mathrm{k}}^{\mathrm{P}}$ (the k-th level of the polynomial-time hierarchy) if $\Sigma_{\mathrm{k}}^{\mathrm{P}A} = \Sigma_{\mathrm{k+1}}^{\mathrm{P}}$. High_k is the class of languages that are high for $\Sigma_{\mathrm{k}}^{\mathrm{P}}$.*

Definition 1.23 ([BH77]) *A set A is* paddable *if there exists a polynomial-time computable, polynomial-time invertible function $p(\cdot,\cdot)$ such that for all $x \in \Sigma^*$ and $a \in \Sigma$,*

$$x \in A \Leftrightarrow p(x,a) \in A.$$

Definition 1.24 ([Sel79]) *A set B is* p-selective *if there exists a total function $f \in$ FP (the selector function) such that for all x and y, $f(x,y) \in \{x,y\}$ and if either of x and y belongs to B, then $f(x,y) \in B$.*

Definition 1.25 ([Amb84]) *A set A is* polynomial-time T-autoreducible *(T-autoreducible, for short) if there exists a polynomial-time bounded oracle Turing machine M such that $A = L(M^A)$ and for all x, M on input x never queries x. A set A is* polynomial-time m-autoreducible *(m-autoreducible, for short) if $A \leq_{\mathrm{m}}^{\mathrm{P}} A$ via a reduction function f such that for all x, $f(x) \neq x$.*

Let $\leq_r^p$ be a polynomial time reducibility.

Definition 1.26 ([Amb84]) *A set A is* polynomial-time r-mitotic *(r-mitotic, for short) if there exists a set $B \in$ P such that:*

$$A \equiv_r^p A \cap B \equiv_r^p A \cap \overline{B}.$$

A set A is polynomial-time weakly r-mitotic *(weakly r-mitotic, for short) if there exist disjoint sets A_0 and A_1 such that $A_0 \cup A_1 = A$, and*

$$A \equiv_r^p A_0 \equiv_r^p A_1.$$

1.3 Recursion Theory

Definition 1.27 *A partial function $f : \Sigma^* \to \Sigma^*$ is* computable *if there exists a Turing machine M such that the following holds for all $x \in \Sigma^*$:*

- *If $f(x)$ is defined, then M on input x halts after finitely many computation steps and outputs $f(x)$.*
- *If $f(x)$ is not defined, then M does not halt on input x.*

Definition 1.28 *The class of* recursive sets *(or decidable sets) is defined as*

$$\mathrm{REC} =_{\mathrm{def}} \{A \subseteq \Sigma^* \,\big|\, \text{there exists a Turing machine } M \text{ that computes } c_A\}.$$

Definition 1.29 *The class of* recursively enumerable sets *is defined as*

$$\mathrm{RE} =_{\mathrm{def}} \{A \subseteq \Sigma^* \,\big|\, \text{there exists a Turing machine } M \text{ that computes } \chi_A\}.$$

Definition 1.30 *Let $A, B \subseteq \Sigma^*$. The set A is* recursive many-one reducible *to the set B (notation $A \leq_m B$) if there exists a computable function f such that for all x, it holds that*

$$x \in A \Leftrightarrow f(x) \in B.$$

Definition 1.31 *Let $A, B \subseteq \Sigma^*$. The set A is* recursive Turing reducible *to the set B if there exists a deterministic oracle Turing machine M such that for all strings x,*

$$x \in A \;\Leftrightarrow\; M \text{ with } B \text{ as oracle accepts the input } x.$$

Observe that in the literature, these reducibilities are normally referred to as *many-one reducibility* and *Turing reducibility*. However, we add the prefix *recursive* in order to distinguish these reducibilities from from their polynomial-time counterparts.

Definition 1.32 *Let A be a set, $\mathcal{C}$ be a complexity class and let $r \in \{\mathrm{m}, \mathrm{T}\}$ be some recursive reducibility. The* reduction closure under recursive r-reducibility *and the* recursive r-degree *of A (resp., $\mathcal{C}$) are defined as follows.*

$$\begin{aligned}
\mathcal{R}_r(A) &=_{\mathrm{def}} \{B \,\big|\, B \leq_{\mathrm{r}} A\},\\
\mathcal{R}_r(\mathcal{C}) &=_{\mathrm{def}} \bigcup_{A \in \mathcal{C}} \mathcal{R}_r(A),\\
\deg_{\mathrm{r}}(A) &=_{\mathrm{def}} \{B \,\big|\, A \equiv_r B\},\\
\deg_{\mathrm{r}}(\mathcal{C}) &=_{\mathrm{def}} \bigcup_{A \in \mathcal{C}} \deg_{\mathrm{r}}(A).
\end{aligned}$$

Definition 1.33 *For $i \in \mathbb{N}$, we define*

$$D_i =_{\mathrm{def}} \{x \in \Sigma^* \,\big|\, \text{the Turing machine with Gödel number } i \text{ halts on input } x\}.$$

We denote the *halting problem* with K_0. It is defined as

$$K_0 =_{\mathrm{def}} \{i \in \mathbb{N} \,\big|\, \text{the Turing machine with Gödel number } i \text{ halts on input } i\}.$$

Turing's famous result states that the halting problem K_0 is undecidable [Tur36]. More precisely, $K_0 \in \mathrm{RE} - \mathrm{REC}$.

Part I

Structural Properties of NP-Hard Sets

Chapter 2

Unions of Disjoint, Equivalent NP-Sets

Whenever a new complexity class is discovered, one of the first questions usually posed is what kind of closure properties it has (on the other hand, new complexity classes are often discovered by considering the closure of a known class under a certain operation). In particular, the closure properties under Boolean operations, i.e., under union, intersection, and complementation are of great interest.

Almost all of the better-known complexity classes like $\mathrm{L}, \mathrm{P}, \mathrm{NP}, \mathrm{coNP}, \mathrm{PSPACE}$, and the levels of the polynomial-time hierarchy are closed under union and intersection. Moreover, the closure under union and intersection is usually very easy to obtain as in the cases of the classes mentioned above. Here the classes PP and 1NP represent exceptions: The former class is closed under union and intersection but the only known proof [BRS91] of this is rather involved. The latter class is closed under intersection but is not known (and not expected) to be closed under union.

Due to the asymmetry of nondeterministic time bounded complexity classes, these are not (believed to be) closed under complementation. Interestingly, the situation is different for nondeterministic space which is closed under complementation by the famous result of Immerman and Szelepcsényi [Imm88, Sze87].

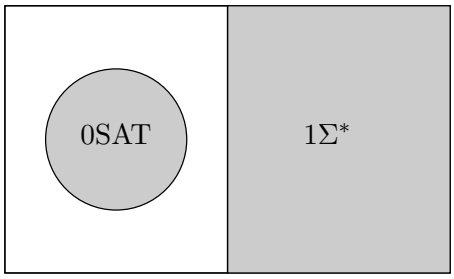
Polynomial-time complexity classes are closed downwards with respect to many-one polynomial-time reducibility. Therefore, many complexity classes can be characterised by their complete problems. For instance, NP can be characterised by SAT, the problem of testing satisfiability for Boolean formulas:

$$\mathrm{NP} = \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(\mathrm{SAT}) = \{A \subseteq \Sigma^* \mid A \leq_{\mathrm{m}}^{\mathrm{p}} \mathrm{SAT}\}$$

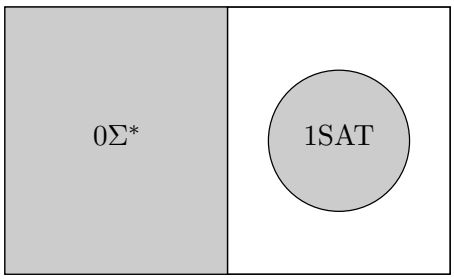
From a structural point of view, it is interesting to study closure properties of objects less coarse than complexity classes, namely of degrees. By their definition, degrees of sets differ from complexity classes in that they are generally not closed under reducibilities since they only contain *equivalent* sets. The most prominent degree in computational complexity theory is definitely the degree of NP-complete sets, which is defined as follows:

$$\mathrm{NPC} =_{\mathrm{def}} \deg_{\mathrm{m}}^{\mathrm{p}}(\mathrm{SAT}) = \{A \subseteq \Sigma^* \mid A \equiv_{\mathrm{m}}^{\mathrm{p}} \mathrm{SAT}\}$$

A naive approach yields that NPC is not closed under union. More precisely, it is easy to see that there exist NP-complete sets whose union is not NP-complete. This is depicted in Figure 2.1. First observe that $A =_{\mathrm{def}} 0\mathrm{SAT} \cup 1\Sigma^*$ is NP-complete via the reduction function $f(x) =_{\mathrm{def}} 0x$ and $B =_{\mathrm{def}} 0\Sigma^* \cup 1\mathrm{SAT} \cup \varepsilon$ is NP-complete via the reduction function $g(x) =_{\mathrm{def}} 1x$. Consequently, $A \cup B = \Sigma^*$. Σ^* is not NP-complete by Lemma 1.19.



$A = 0\mathrm{SAT} \cup 1\Sigma^*$



$B = 0\Sigma^* \cup 1\mathrm{SAT} \cup \varepsilon$

Figure 2.1: Non-disjoint NP-complete sets A and B where $A \cup B$ is not NP-complete.

Clearly, this approach does not capture the intrinsic difficulty of the question. The fact that the two sets are not disjoint oversimplifies things. As we will see, the following question is far more subtle:

Is the union of two *disjoint* NP-complete sets always NP-complete?

Selman [Sel88] posed this question in 1988. It turned out to be a very difficult question which has not been solved up to the present day. Therefore, this question gives a nice

motivation for the studies in this chapter. Throughout this chapter, we will refer to it as the *main question*.

If one attempts to prove that unions of disjoint NP-complete sets are always NP-complete, one encounters the following difficulty: Let $A, B, C \subseteq \Sigma^*$ such that A is in NP and B and C are disjoint NP-complete sets. Assume $A \leq_m^p B$ via $f \in \mathrm{FP}$ and $A \leq_m^p C$ via $g \in \mathrm{FP}$. This means $x \in A \leftrightarrow f(x) \in B$ and $x \in A \leftrightarrow g(x) \in C$. Let $x \in A$, so f maps x to B and g maps x to C. However, neither of the two functions qualifies to prove that $A \leq_m^p B \cup C$: Let $x' \notin A$. Then it follows that $f(x') \notin B$ and $g(x') \notin C$. Nevertheless, it could happen that $f(x') \in C$ or $g(x') \in B$. It is not at all clear how we can ensure that f avoids C and that g avoids B for elements outside A. More precisely, a function which reduces A to $B \cup C$ must map elements from $\overline{A}$ to $\overline{B \cup C}$. This problem is depicted in Figure 2.2.

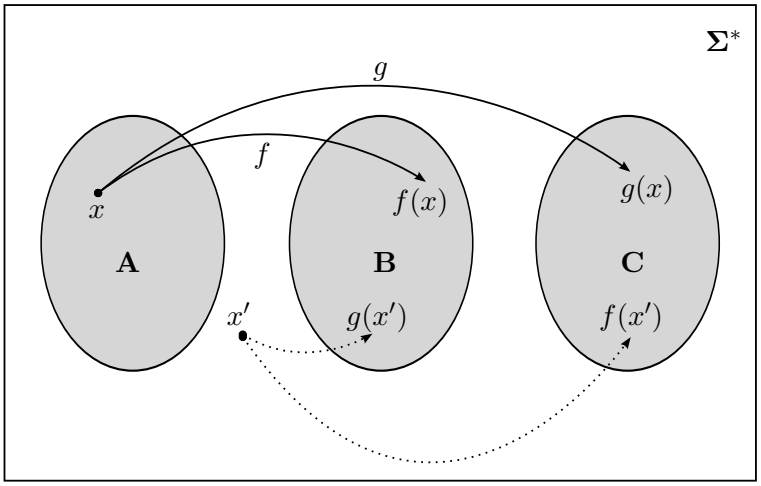


Figure 2.2: A, B, and C are subsets of Σ^*, $A \in \mathrm{NP}$, B and C are disjoint NP-complete sets. $x \in A$, $x' \notin A$. A reduces to B via $f \in \mathrm{FP}$, A reduces to C via $g \in \mathrm{FP}$.

There are many natural generalisations of the main question, some of which we will also address in this chapter. For instance, it is also interesting to consider coarser degrees, i.e., degrees defined by more general reducibilities like Turing reducibility. Moreover, we are not bound to limit our research to the complete sets, i.e., to the sets in the highest degree of NP. A natural generalisation of our main question is whether there exists some degree in NP$-$P which has the property that it is closed under unions of disjoint, equivalent sets.

Before we study the complexity of unions of equivalent sets within NP, we digress to recursion theory in order to catch a glimpse of the situation there.

2.1 A Digression to Recursion Theory

While the complexity classes P and NP are the most prominent complexity classes in computational complexity theory, the classes REC and RE are situated in the very center of interest in recursion theory.

Interestingly, there exists a natural correspondence between computational complexity theory and recursion theory. Both the class P and REC represent classes of *feasible* problems in their particular discipline. The classes NP and RE are obtained by applying polynomially bounded existential quantification to P and unbounded existential quantification to REC, respectively.

In contrast to the polynomial-time setting where the question of whether the NP-complete sets are closed under unions of disjoint sets seems to be very difficult, the similar question for RE-complete sets can be answered affirmatively without much effort. In this section we will prove that unions of disjoint RE-complete sets are always RE-complete. Note that in the recursive setting, with "-complete" we will always refer to *recursive many-one reducibility* or *recursive Turing reducibility*.

In the recursive setting, the notion of *creative sets* enables us to prove the central statement of this section in a straightforward and concise way. This notion is motivated by the idea that RE-complete sets are those whose complements are effectively non-recursively enumerable. The notation *creative sets* was introduced by Post [Pos44]. We first give the formal definition of creative sets.

Definition 2.1 *Let $A \subseteq \Sigma^*$ be a recursively enumerable set. A is* creative *if there exists a computable function $\psi : \mathbb{N} \to \Sigma^*$ such that for all $i \in \mathbb{N}$, it holds that*

$$D_i \subseteq \overline{A} \quad \Rightarrow \quad \psi(i) \text{ is defined and } \psi(i) \in \overline{A \cup D_i}.$$

The notion of creative sets plays an important role in recursion theory as it provides an alternative characterisation of many-one completeness for the class of recursively enumerable sets. Moreover, this characterisation allows a fairly easily demonstration that all many-one RE-complete sets are recursively isomorphic [Rog67].

Theorem 2.2 *[Myh55] A set $A \subseteq \Sigma^*$ is creative if and only if it is recursive m-complete for* RE.

The following is easy to see:

Lemma 2.3 *There exists a Turing machine M which on input $(i, j) \in \mathbb{N} \times \mathbb{N}$ computes $k \in \mathbb{N}$ such that $D_k = D_i \cup D_j$.*

Proof. Let $(i, j) \in \mathbb{N} \times \mathbb{N}$. M works as follows on input (i, j): M first decodes the Gödel numbers i and j to obtain the instruction sets of M_i, the i-th Turing machine and M_j, the j-th Turing machine.

For all x, it holds that $x \in D_i \cup D_j$ if at least one of the machines M_i, M_j started on input x stops after a finite number of steps.

M needs to construct a machine N which on input x stops if and only if $x \in D_i \cup D_j$. This can be accomplished by a simple dovetailing argument:

On input x, N interleaves the computations of M_i and M_j. More precisely, N alternately simulates a step of M_i and then a step of M_j. It repeats doing so until either of the two machines stops. In this case, N stops. Otherwise, N runs infinitely. Clearly, M can compute an instruction set for N out of the instruction sets for M_i and M_j. All N then has to do is to compute k, the Gödel number of N. □

Theorem 2.4 *Let A be recursive m-complete for RE, and let $B \in$ RE such that $A \cap B = \emptyset$. Then $A \cup B$ is m-complete for* RE.

Proof. Let A be m-complete for RE, and let $B \in$ RE such that $A \cap B = \emptyset$. By Theorem 2.2, A is creative. As $B \in$ RE, there exists $b \in \mathbb{N}$ such that $B = D_b$. Let ξ be the function computed by the Turing machine M in Lemma 2.3, and let ψ be the computable function which witnesses A's creativity. We define a computable function $\phi : \mathbb{N} \to \mathbb{N}$.

$$\phi(i) =_{\text{def}} \psi(\xi(i, b))$$

We now prove that this function witnesses that $A \cup B$ is creative: Clearly, ϕ is computable because both ψ and ξ are computable. Let $i \in \mathbb{N}$ and suppose that $D_i \subseteq \overline{A \cup B}$. Hence $D_{\xi(i,b)} = D_i \cup D_b = D_i \cup B \subseteq \overline{A}$. Since ψ witnesses that A is creative, we obtain that $\phi(i) = \psi(\xi(i, b))$ is defined. Moreover, $\psi(\xi(i, b)) \in \overline{A \cup D_{\xi(i,b)}} = \overline{A \cup B \cup D_i}$. So $A \cup B$ is creative via ϕ. By Theorem 2.2, $A \cup B$ is m-complete for RE. □

Corollary 2.5 *Let A and B be disjoint, recursive m-complete sets for RE. Then the set $A \cup B$ is m-complete for* RE.

Corollary 2.5 implies that the $\text{degree}_m(K_0)$ is closed under unions of disjoint sets.

In the recursive setting, the case of T-reducibility can be settled in a straightforward way.

Theorem 2.6 *Let A be recursive T-complete for RE, and let $B \in$ RE such that $A \cap B = \emptyset$. Then $A \cup B$ is T-complete for* RE.

Proof. Let A be T-complete for RE, and let $B \in$ RE such that $A \cap B = \emptyset$. We describe a Turing machine M which has oracle access to $A \cup B$ and decides the set A. On input

x, M first queries the oracle for x. If it gets a negative answer, i.e. $x \notin A \cup B$, it follows that $x \notin A$, so M can reject x. If $x \in A \cup B$, M gets a positive answer from the oracle. M has to find out whether x belongs to A or to B. Let $a, b \in \mathbb{N}$ such that $D_a = A$ and $D_b = B$. Similarly to Lemma 2.3, M can simulate the a-th and the b-th Turing machine simultaneously on input x. As $A \cap B = \emptyset$, precisely one of the two machines will eventually halt on x. If the a-th machine halts, M accepts x, if the b-th machine halts, M rejects x. This proves $A \leq_{\mathrm{T}} A \cup B$. □

Theorem 2.6 was already observed by Shoenfield [Sho76].

Corollary 2.7 *Let A and B be disjoint, recursive T-complete sets for RE. Then the set $A \cup B$ is T-complete for* RE.

Observe that contrary to the m-case, Corollary 2.7 does not imply that $\mathrm{degree}_T(K_0)$ is closed under unions of disjoint sets. Obviously, this is not the case because $\mathrm{degree}_T(K_0)$ contains both K_0 and $\overline{K_0}$ since $K_0 \equiv_{\mathrm{T}} \overline{K_0}$. For the same reason, no full Turing-degree of a nontrivial set can be closed under unions of disjoint sets. Nevertheless, Corollary 2.7 does imply that $\mathrm{degree}_T(K_0) \cap \mathrm{RE}$ is closed under unions of disjoint sets.

2.2 Necessary and Sufficient Conditions for the Polynomial-Time Setting

The unconditional proofs in Section 2.1 are in contrast to the polynomial-time setting, where we do not know whether similar statements hold. It is evident that we cannot expect unconditional results about unions of disjoint NP-complete sets:

In the unlikely case that P = NP, unions of disjoint T-complete sets for NP are always T-complete because the union of two P-sets is always in P and any set in P is T-complete for P. Under the same improbable assumption, the situation for m-complete sets is similar. The only difference is that by the definition of m-reducibility, neither Σ^* nor the empty set can be complete for NP (or for P). However, the situation is almost the same as in the T-case: If A and B are m-complete for NP such that $A \cap B = \emptyset$ and $A \cup B \neq \Sigma^*$, then $A \cup B$ is m-complete for NP.

Consequently, the question of whether unions of disjoint NP-complete sets are always NP-complete is only interesting under the widely believed assumption that P ≠ NP.

It turns out that the situation is more subtle in the polynomial setting. In fact, the assumption P ≠ NP is not strong enough for an affirmative answer to the main question:

Although P $\neq$ NP, it could still be the case that NP = coNP. Then SAT and $\overline{\text{SAT}}$ are both NP-complete, but SAT $\cup$ $\overline{\text{SAT}}$ = Σ^* cannot be NP-complete, so the main question has a negative answer.

Ogiwara and Hemaspaandra [OH93] construct a relativised world where this actually is the case. They construct an oracle O relative to which $\text{UP}^O \neq \text{NP}^O = \text{PSPACE}^O$. Relative to their oracle O, $\text{P}^O \neq \text{NP}^O$ and $\text{NP}^O = \text{coNP}^O$.

In fact, NP $\neq$ coNP is the weakest hypothesis we know of which could enable us to prove that the union of two disjoint NP-complete sets is always NP-complete.

In this section we prove that the situation in fact is as follows:

- If the main question has a positive answer, then NP does not equal coNP.
- If the main question has a negative answer, then NP equals coNP or there exist p-inseparable NP-complete sets in NP.

This reveals that we do not know any unlikely consequences of a yes- or no- answer. So basically, it is not clear what to believe with respect to the main question.

2.2.1 P-Separable Sets and Paddable Sets

Now that we now that NP $\neq$ coNP is a necessary condition for an affirmative answer to the main question, we also want to formulate a sufficient condition.

If one is asked to name two disjoint NP-complete problems, then one will usually come up with two natural sets that are disjoint for syntactic reasons, e.g., 0SAT and 1SAT. Trivially, 0SAT $\cup$ 1SAT is NP-complete.

Interestingly, the same can also happen with more sophisticated examples of disjoint NP-complete sets:

$$\begin{aligned} \textsc{Clique} &= \{(G,k) \mid G \text{ is a graph which has a clique of size } \geq k+1\} \\ \textsc{Colouring} &= \{(G,k) \mid G \text{ is a graph which is k-colourable}\} \end{aligned}$$

It is easy to see that Clique and Colouring are disjoint, because whenever a graph can be coloured with k colours, it cannot have a clique of size $\geq k+1$. Observe that $\overline{\textsc{Clique} \cup \textsc{Colouring}}$ is not empty, because not every graph that does not have a clique of size $\geq k+1$ can be coloured with k colours, as Figure 2.3 depicts.

It turns out that Clique $\cup$ Colouring is in fact NP-complete.

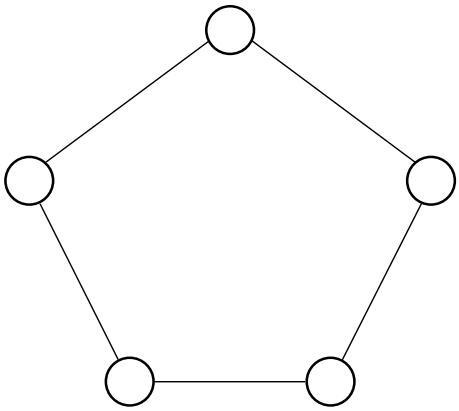


Figure 2.3: A graph G where $(G, 2) \in \overline{\textsc{Clique} \cup \textsc{Colouring}}$. G cannot be coloured with 2 colours although it does not have a clique of size 3.

These two examples have in common that the involved sets are p-separable. For 0SAT and 1SAT this is accomplished by the separator $0\Sigma^*$. For CLIQUE and COLOURING the separator is constructed via the θ function by Lovász [Lov79]. It turns out that it is the p-separability of the sets that causes the unions to be NP-complete. In fact, all unions of p-separable NP-complete sets are NP-complete (with the trivial exception of Σ^*):

Proposition 2.8 *If A and B are p-separable such that $A \neq \overline{B}$, then $A \leq_{\mathrm{m}}^{\mathrm{p}} A \cup B$.*

Proof. Let $w \in \overline{A \cup B}$ and let S be a separator such that $A \subseteq S$ and $B \subseteq \overline{S}$. A reduces to $A \cup B$ via the function that on input x, outputs w if $x \notin S$, and outputs x otherwise. □

Proposition 2.9 *If* NP $\neq$ coNP *and all pairs of disjoint* NP*-complete sets are p-separable, then the union of disjoint* NP*-complete sets is always* NP*-complete.*

Unfortunately, it is questionable whether the approach via p-separable sets will allow us to answer the main question. On the one hand, Homer and Selman [HS92] construct an oracle relative to which NP does not contain p-inseparable pairs. Under this oracle, the main question has an affirmative answer. On the other hand, Grollmann and Selman [GS88] show that it is unlikely that *all* disjoint NP sets are p-separable because this would imply that P = UP.

Consequently, it is unclear whether the approach via p-separability can solve the question of whether *all* unions of disjoint NP-complete sets are NP-complete.

What is the situation for *paddable* NP-complete sets? The notion of paddability is closely related to the famous *Berman-Hartmanis Isomorphism Conjecture* [BH77]. The conjecture states that all NP-complete sets are p-isomorphic. It is easy to see that this implies

P $\neq$ NP, because no finite set can be p-isomorphic to the infinite set SAT. Moreover, the Isomorphism Conjecture is equivalent to assuming that all NP-complete sets are paddable.

The conjecture has neither been proven nor refuted although it has generated a considerable amount of research in computational complexity. Serious objections were raised against it by Joseph and Young [JY85], who constructed NP-complete sets of the form $f(A)$ where A is a paddable NP-complete set and f is a one-way function. They argued that since one-way functions are not invertible in polynomial time, it may be that $f(A)$ is not p-isomorphic to A. Based on this, they conjectured that there is a one-way function f and a paddable NP-complete set A such that $f(A)$ is not p-isomorphic to A. This conjecture has been referred to as the *Encrypted Complete Set Conjecture*.

Furthermore, Kurtz, Mahaney, and Royer [KMR95] show that the Isomorphism Conjecture fails relative to a random oracle. Nevertheless, the Isomorphism Conjecture ist not known to imply any really unlikely consequences.

So if we then assume that all NP-complete sets are paddable, is it then easier to find an answer to our main question concerning unions of disjoint NP-complete sets? Unfortunately, it turns out that a restriction to paddable sets will not make the question easier:

Theorem 2.10 *[GPSS06] The following are equivalent:*

1. *For all disjoint* NP*-complete sets A and B it holds that $A \cup B$ is* NP*-complete.*

2. *For all* NP*-complete sets A and $B \in$ NP where $A \cap B = \emptyset$ it holds that $A \cup B$ is* NP*-complete.*

3. *For all paddable, disjoint* NP*-complete sets A and B it holds that $A \cup B$ is* NP*-complete.*

4. *For all* NP*-complete sets B where* SAT$\cap B = \emptyset$ *it holds that* SAT$\cup B$ *is* NP*-complete.*

In the recursion theory setting, the notion of *creative sets* allowed us to prove that unions of disjoint RE-complete sets are RE-complete (see Section 2.1). There were several attempts to translate the definition of creativeness to the polynomial settings. For instance, the aforementioned NP-complete sets of the form $f(A)$ where A is a paddable NP-complete set and f is a one-way function are called *k-completely-creative sets* [JY85]. Also, Wang [Wan91] defined k-creative sets which are a generalisation of k-completely-creative sets.

However, neither of the two concepts have been able to successfully capture NP-complete sets. Most of the natural complete sets for NP are not known to be even k-creative. Moreover, it is not known whether all k-creative sets are NP-complete [AB96].

2.3 Evidence for the Complexity of Unions of Disjoint NP-Complete Sets

We have argued in the last section that the perspective of answering the main question is not too promising. Nevertheless, we can prove partial results: Although we cannot show that unions of disjoint NP-complete are always NP-complete, we can show that such unions cannot become too easy. More precisely, we prove the following for all disjoint NP-complete sets B and C:

1. $B \cup C$ is high for NP. By [KS97], this is equivalent to saying that $B \cup C$ is strongly nondeterministic-Turing-complete for NP.

2. Under a reasonable hypothesis, $B \cup C$ is non-uniformly many-one-complete for NP.

Our results show that unions of disjoint NP-complete sets remain complete with respect to more general reducibilities. This is evidence that unions of disjoint NP-complete sets retain much of the complexity of their single components.

As a byproduct, we obtain that the levels $1, 2, \ldots$ of the high-hierarchy are closed under unions with arbitrary (but disjoint) NP-sets.

2.3.1 The High-Hierarchy

High sets are generalisations of NP-complete sets. In this section, we show that relaxed to high sets, our main question has an affirmative answer.

Lemma 2.11 *Let $A, B \in \mathrm{NP}$ such that $A \cap B = \emptyset$. Then $\mathrm{NP}^A \subseteq \mathrm{NP}^{A \cup B}$.*

Proof. Let M_A and M_B be nondeterministic polynomial-time Turing machines such that $L(M_A) = A$ and $L(M_B) = B$, and let $C \in \mathrm{NP}^A$ via a nondeterministic polynomial-time oracle Turing machine (NPOTM) M; i.e., $L(M^A) = C$.

We construct a NPTOM N such that $L(N^{A \cup B}) = C$:

N simulates M on input x until M wants to query the oracle A. Assume M wants to query A for the string q. Recall that N on its simulation of M cannot query oracle A but only the oracle $A \cup B$. So N queries $A \cup B$ for q.

Case 1: $q \notin A \cup B$: It then follows that $q \notin A$, so N can continue the simulation of M with a negative answer to the query q.

Case 2: $q \in A \cup B$: N branches nondeterministically into two paths. On the first path, it simulates $M_A(q)$, on the second, it simulates $M_B(q)$. Since A and B are disjoint, only one of these two machines produces an accepting path. Then N continues as follows:

- On all accepting paths of $M_A(q)$ (if any), N continues the simulation of M with a positive answer to the query q.
- On all rejecting paths of $M_A(q)$, N rejects.
- On all accepting paths of $M_B(q)$ (if any), N continues the simulation of M with a negative answer to the query q.
- On all rejecting paths of $M_B(q)$, N also rejects.

During its simulation of M, N proceeds in the same way for all of M's queries to A. Observe that since $M_A(q)$ (or $M_B(q)$, respectively) can produce more than one accepting path, N will in general perform several parallel simulations of M after a simulation of $M_A(q)$ or $M_B(q)$. As $L(M_A) \cap L(M_B) = \emptyset$, all these parallel simulations are identical. Consequently, it is immediately clear that $N^{A\cup B}(x)$ produces an accepting path if and only if $M^A(x)$ produces an accepting path and hence $L(N^{A\cup B}) = L(M^A) = C$. This proves $C \in \mathrm{NP}^{A\cup B}$ and we obtain $\mathrm{NP}^A \subseteq \mathrm{NP}^{A\cup B}$. □

If we were able to prove a similar statement for P, it would follow that Turing-complete sets for NP are closed under unions of disjoint sets. The proof of Theorem 2.11 however crucially depends on nondeterminism.

Theorem 2.12 *Let $k \geq 1$, $A \in \mathrm{High}_k$ and $B \in \mathrm{NP}$ such that $A \cap B = \emptyset$. Then $A \cup B \in \mathrm{High}_k$.*

Proof. Let $k \geq 1$, $A \in \mathrm{High}_k$ and $B \in \mathrm{NP}$ such that $A \cap B = \emptyset$. We will show that

$$\Sigma_{\mathrm{k+1}}^{\mathrm{P}} = \Sigma_{\mathrm{k}}^{\mathrm{P}\,A} \subseteq (\Sigma_{\mathrm{k}}^{\mathrm{P}})^{A\cup B}.$$

Since A is a set from High_k, the first equality follows from the definition. We will argue for $\Sigma_{\mathrm{k}}^{\mathrm{P}\,A} \subseteq (\Sigma_{\mathrm{k}}^{\mathrm{P}})^{A\cup B}$ by induction over k.

(IB) Let k=1. Then $\Sigma_1^{\mathrm{P}\,A} = \mathrm{NP}^A \subseteq (\Sigma_1^{\mathrm{P}})^{A\cup B} = \mathrm{NP}^{A\cup B}$ holds due to Lemma 2.11.

(IH) Let us assume that $\Sigma_{\mathrm{k}}^{\mathrm{P}\,A} \subseteq (\Sigma_{\mathrm{k}}^{\mathrm{P}})^{A\cup B}$ holds for a $k \geq 1$.

(IS) By definition, $(\Sigma_{\mathrm{k+1}}^{\mathrm{P}})^A = (\mathrm{NP}^{\Sigma_{\mathrm{k}}^{\mathrm{P}}})^A$.

Observe that $\left(\mathrm{NP}^{\Sigma_{\mathrm{k}}^{\mathrm{P}}}\right)^A \subseteq \mathrm{NP}^{0A\cup 1O}$ for a suitable set $O \in \Sigma_{\mathrm{k}}^{\mathrm{P}\,A}$.
By the induction hypothesis, $O \in (\Sigma_{\mathrm{k}}^{\mathrm{P}})^{A\cup B}$.

Arguing similarly as in Lemma 2.11, we obtain

$$\mathrm{NP}^{0A\cup 1O} \subseteq (\mathrm{NP}^{\Sigma_{\mathrm{k}}^{\mathrm{P}}})^{A\cup B} = (\Sigma_{\mathrm{k+1}}^{\mathrm{P}})^{A\cup B}.$$

This shows that for all $k \geq 1$, it holds that $\Sigma_{\mathrm{k+1}}^{\mathrm{P}} = \Sigma_{\mathrm{k}}^{\mathrm{P}\,A} \subseteq (\Sigma_{\mathrm{k}}^{\mathrm{P}})^{A\cup B}$. □

The following corollary is an immediate consequence of Theorem 2.12.

Corollary 2.13 *For all $k \geq 1$, High_k is closed under unions of disjoint sets.*

Corollary 2.14 *Let $A, B \in \mathrm{NPC}$ such that $A \cap B = \emptyset$. Then $A \cup B$ is $\leq^{\mathrm{p}}_{\mathrm{snT}}$-complete for* NP*.*

Proof. Since A and B are both NP-complete, they obviously are in High_1. Theorem 2.12 yields that $A \cup B$ is also in High_1. However, a set is in High_1 if and only if it is $\leq^{\mathrm{p}}_{\mathrm{snT}}$-complete for NP [KS97]. □

2.3.2 A Non-Uniform Reducibility

In Section 2.3.1 we showed that the union of a disjoint NP-complete set and an arbitrary NP-set is high for NP. In this section we give further evidence that unions of disjoint NP-complete are not far from being NP-complete. To do so, we assume that NP contains uniformly hard languages, i.e., languages that are uniformly not contained in coNP. Under this hypothesis we show the following:

For every NP-complete A and every $B \in \mathrm{NP}$ that is disjoint from A it holds that $A \cup B$ is nonuniformly NP-complete.

We remark that Downey and Fortnow [DF03] studied languages that are uniformly hard for P. The notion we study in this section is a similar notion describing uniform-hardness for NP.

We start with some notations which are related to the notion of *uniform hard languages*.

Definition 2.15 *Let $\mathcal{C}$ and $\mathcal{D}$ be complexity classes, and let A and B be subsets of Σ^*.*

1. *$A \overset{\text{i.o.}}{=} B \iff_{\text{def}}$ for infinitely many n it holds that $A \cap \Sigma^n = B \cap \Sigma^n$.*
2. *$A \overset{\text{i.o.}}{\in} \mathcal{C} \iff_{\text{def}}$ there exists $C \in \mathcal{C}$ such that $A \overset{\text{i.o.}}{=} C$.*
3. *$\mathcal{C} \overset{\text{i.o.}}{\subseteq} \mathcal{D} \iff_{\text{def}} C \overset{\text{i.o.}}{\in} \mathcal{D}$ for all $C \in \mathcal{C}$.*

The following can easily be seen.

Proposition 2.16 *Let $\mathcal{C}$ and $\mathcal{D}$ be complexity classes, and let A and B be subsets of Σ^*.*

1. *$A \overset{\text{i.o.}}{=} B$ if and only if $\overline{A} \overset{\text{i.o.}}{=} \overline{B}$.*
2. *$A \overset{\text{i.o.}}{\in} \mathcal{C}$ if and only if $\overline{A} \overset{\text{i.o.}}{\in} \mathrm{co}\mathcal{C}$.*
3. *$\mathcal{C} \overset{\text{i.o.}}{\subseteq} \mathcal{D}$ if and only if $\mathrm{co}\mathcal{C} \overset{\text{i.o.}}{\subseteq} \mathrm{co}\mathcal{D}$.*

Proposition 2.17 *The following are equivalent:*

(i) $\mathrm{coNP} \overset{\text{i.o.}}{\not\subseteq} \mathrm{NP}$

(ii) $\mathrm{NP} \overset{\text{i.o.}}{\not\subseteq} \mathrm{coNP}$

(iii) There exists an $A \in \mathrm{NP}$ *such that* $A \overset{\text{i.o.}}{\notin} \mathrm{coNP}$.

(iv) There exists a paddable NP*-complete* A *such that* $A \overset{\text{i.o.}}{\notin} \mathrm{coNP}$.

Proof. The equivalence of (i) and (ii) is by Proposition 2.16. Moreover, from the definition it immediately follows that $\neg(\mathrm{ii}) \Rightarrow \neg(\mathrm{iii})$ and $\neg(\mathrm{iii}) \Rightarrow \neg(\mathrm{iv})$. It remains to show $\neg(\mathrm{iv}) \Rightarrow \neg(\mathrm{ii})$. So we assume that for all paddable NP-complete A it holds that $A \overset{\text{i.o.}}{\in} \mathrm{coNP}$. Choose any $C \in \mathrm{NP}$ and let $B = 0C \cup 1\mathrm{SAT}$. Hence B is paddable and NP-complete. By our assumption $B \overset{\text{i.o.}}{\in} \mathrm{coNP}$. So there exists a $D \in \mathrm{coNP}$ such that $B \overset{\text{i.o.}}{=} D$. Let $D' = \{w \mid 0w \in D\}$ and note that $D' \in \mathrm{coNP}$. Observe that for every n, if $B \cap \Sigma^{n+1} = D \cap \Sigma^{n+1}$, then $C \cap \Sigma^n = D' \cap \Sigma^n$. Hence $C \overset{\text{i.o.}}{=} D'$ which shows $C \overset{\text{i.o.}}{\in} \mathrm{coNP}$. □

For the following results, we assume that $\mathrm{NP} \overset{\text{i.o.}}{\not\subseteq} \mathrm{coNP}$. This is a believable assumption that says that (for sufficiently long formulas) not all tautologies of a given size have short proofs (confer Proposition 2.17).

Theorem 2.18 *If* $\mathrm{NP} \overset{\text{i.o.}}{\not\subseteq} \mathrm{coNP}$*, then for every* NP*-complete* A *and every* $B \in \mathrm{NP}$ *that is disjoint to* A *it holds that* $A \cup B$ *is* $\leq_{\mathrm{m}}^{\mathrm{p/poly}}$*-complete for* NP.

Proof. By assumption, there exists an NP-complete K such that $K \overset{\text{i.o.}}{\notin} \mathrm{coNP}$. Then choose $f \in \mathrm{FP}$ such that $K \leq_{\mathrm{m}}^{\mathrm{p}} A$ via f, and choose $g \in \mathrm{FP}$ such that $\{(u,v) \mid u \in K \vee v \in K\} \leq_{\mathrm{m}}^{\mathrm{p}} K$ via g.

$$\mathrm{EASY} =_{\mathrm{def}} \{u \mid \exists v, |v| = |u|, f(g(u,v)) \in B\}$$

EASY belongs to NP. That $\mathrm{EASY} \subseteq \overline{K}$ can be seen as follows: $f(g(u,v)) \in B$ implies $g(u,v) \notin K$ which shows $u \notin K$. Intuitively, EASY is a set of words u that are outside K and that have short proofs for this. (The proof is v together with an accepting path proving $f(g(u,v)) \in B$.) From our assumption $\overline{K} \overset{\text{i.o.}}{\notin} \mathrm{NP}$ it follows that there exists an $n_0 \geq 0$ such that

$$\forall n \geq n_0, \overline{K}^{=n} \not\subseteq \mathrm{EASY}^{=n}.$$

So for every $n \geq n_0$ we can choose a word $w_n \in \overline{K}^{=n} - \mathrm{EASY}$. For $n < n_0$, let $w_n = \varepsilon$. Choose a fixed $z_1 \in A \cup B$ and a $z_0 \notin A \cup B$ (such a z_0 exists since $\mathrm{NP} \overset{\text{i.o.}}{\not\subseteq} \mathrm{coNP}$ implies $\mathrm{NP} \neq \mathrm{coNP}$). We define the reduction that witnesses $K \leq_{\mathrm{m}}^{\mathrm{p/poly}} A \cup B$.

$$h(v) =_{\mathrm{def}} \begin{cases} f(g(w_{|v|}, v)) & : \text{ if } |v| \geq n_0 \\ z_1 & : \text{ if } |v| < n_0 \text{ and } v \in K \\ z_0 & : \text{ if } |v| < n_0 \text{ and } v \notin K \end{cases}$$

Observe that $h \in \mathrm{FP/poly}$ with the advice $n \mapsto w_n$.

We claim that for all v it holds that

$$v \in K \Leftrightarrow h(v) \in A \cup B. \tag{2.1}$$

This equivalence obviously holds for all v where $|v| < n_0$. So we assume $|v| \geq n_0$. Let $n = |v|$.

If $v \in K$, then $g(w_n, v) \in K$ and hence $f(g(w_n, v)) \in A \subseteq A \cup B$.

If $v \notin K$, then $g(w_n, v) \notin K$ (since $w_n \notin K$). Hence $f(g(w_n, v)) \notin A$. If $f(g(w_n, v)) \in B$, then $w_n \in \mathrm{EASY}$ which contradicts the choice of w_n. Therefore, $f(g(w_n, v)) \notin B$. This proves (2.1) and therefore, $A \cup B$ is $\leq_{\mathrm{m}}^{\mathrm{p/poly}}$-complete for NP. □

2.4 Upper and Lower Bounds

In this section, we abstract from the main question. We investigate how complex the union of two disjoint equivalent NP sets can be, and we state upper and lower bounds.

For any set A, we define the set $\mathcal{U}(A)$ which is the class of all sets which are m-equivalent to the union of two disjoint sets from the m-degree of A.

Definition 2.19 *For a set A, we define the class*

$$\mathcal{U}(A) =_{\mathrm{def}} \deg_{\mathrm{m}}^{\mathrm{p}}\big(\{C \cup D \,\big|\, C \cap D = \emptyset \wedge C \equiv_{\mathrm{m}}^{\mathrm{p}} D \equiv_{\mathrm{m}}^{\mathrm{p}} A\}\big).$$

The next theorem characterises the scope of $\mathcal{U}(A)$. We state a technical lemma first.

Lemma 2.20 *Let $\mathcal{K}$ and $\mathcal{M}$ be complexity classes that are closed under $\leq_{\mathrm{m}}^{\mathrm{p}}$. Then the class $\mathcal{K}\dot{\vee}\mathcal{M}$ is closed under $\leq_{\mathrm{m}}^{\mathrm{p}}$ as well.*

Proof. We have to show that $A \in \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(\mathcal{K}\dot{\vee}\mathcal{M})$ implies $A \in \mathcal{K}\dot{\vee}\mathcal{M}$.

Let $A \in \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(\mathcal{K}\dot{\vee}\mathcal{M})$, hence there exist $f \in \mathrm{FP}, A_1 \in \mathcal{K}, A_2 \in \mathcal{M}$ such that $A_1 \cap A_2 = \emptyset$, and $x \in A \Leftrightarrow f(x) \in A_1 \cup A_2$. For $i \in \{1, 2\}$, let $f^{-1}[A_i] =_{\mathrm{def}} \{x \,\big|\, f(x) \in A_i\}$. Observe that for $i \in \{1, 2\}$, f reduces $f^{-1}[A_i]$ to A_i. As $\mathcal{K}$ and $\mathcal{M}$ are closed under $\leq_{\mathrm{m}}^{\mathrm{p}}$, it follows that $f^{-1}[A_1] \in \mathcal{K}$ and $f^{-1}[A_2] \in \mathcal{M}$. Moreover $f^{-1}[A_1] \cap f^{-1}[A_2] = \emptyset$. We obtain

$$\begin{aligned} x \in A &\Leftrightarrow f(x) \in A_1 \cup A_2 \\ &\Leftrightarrow \big(f(x) \in A_1\big) \vee \big(f(x) \in A_2\big) \\ &\Leftrightarrow \big(x \in f^{-1}[A_1]\big) \vee \big(x \in f^{-1}[A_2]\big) \\ &\Leftrightarrow x \in f^{-1}[A_1] \cup f^{-1}[A_2]. \end{aligned}$$

So x is in A if and only if x is in the union of a $\mathcal{K}$-set and a disjoint $\mathcal{M}$-set, hence $A \in \mathcal{K}\dot{\vee}\mathcal{M}$. □

Theorem 2.21 *For all nontrivial sets A, it holds that*

$$\deg_{\mathrm{m}}^{\mathrm{p}}(A) \subseteq \mathcal{U}(A) \subseteq \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A)\dot{\vee}\mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A).$$

Proof. Let A be a set and $B \in \deg_{\mathrm{m}}^{\mathrm{p}}(A)$. Hence, we have $A \equiv_{\mathrm{m}}^{\mathrm{p}} B \equiv_{\mathrm{m}}^{\mathrm{p}} 0A \cup 1A$. To see that $0A \cup 1A$ is in $\mathcal{U}(A)$, notice that $0A \cap 1A = \emptyset$ and $0A \equiv_{\mathrm{m}}^{\mathrm{p}} 1A \equiv_{\mathrm{m}}^{\mathrm{p}} A$. As B is m-equivalent to A and $0A \cup 1A$, B is also in $\mathcal{U}(A)$.

For the second inclusion, let $E \in \mathcal{U}(A)$. So there exist $C, D \in \deg_{\mathrm{m}}^{\mathrm{p}}(A)$ such that $C \cap D = \emptyset$ and $E \equiv_{\mathrm{m}}^{\mathrm{p}} C \cup D$. So $E \neq \emptyset$. It follows that $E \in \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(C \cup D) \subseteq \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(\deg_{\mathrm{m}}^{\mathrm{p}}(A)\dot{\vee}\deg_{\mathrm{m}}^{\mathrm{p}}(A)) \subseteq \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(\mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A)\dot{\vee}\mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A))$. By Lemma 2.20, this is equal to $\mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A)\dot{\vee}\mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A)$. □

The next proposition implies that for nontrivial sets, at least one of the two inclusions has to be strict.

Proposition 2.22 *For any nontrivial set A, it holds that* $\deg_{\mathrm{m}}^{\mathrm{p}}(A) \subsetneq \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A)\dot{\vee}\mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A)$.

Proof. Let A be a nontrivial set. By definition, $\deg_{\mathrm{m}}^{\mathrm{p}}(A) \subseteq \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A)\dot{\vee}\mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A)$. Since $A \neq \overline{\emptyset}$, it is clear that $\emptyset \in \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A)\dot{\vee}\mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A)$. If $\deg_{\mathrm{m}}^{\mathrm{p}}(A)$ contained the empty set, it would follow that $A = \emptyset$, contradicting our assumption. □

Let A be a set and B and C be disjoint sets that are m-equivalent to A. In the next sections we will study the following phenomena:

- For some A, the union $B \cup C$ is always m-equivalent to A, no matter how B and C are chosen.
- For some A, the union $B \cup C$ can be less complex than A.
- For some A, the union $B \cup C$ can be more complex than A.

2.4.1 M-Idempotent Sets

In the following section, we consider m-equivalent, disjoint sets whose union is at most as complex as the single components. We prove that two extremes can occur:

- Unions of disjoint, m-equivalent NP sets can be equivalent to their single components (Theorem 2.27).
- Unions of disjoint, m-equivalent NP sets can be very easy, e.g. in P (Theorem 2.33).

Definition 2.23 *We say that a nontrivial set A is* m-idempotent *if the following holds for all sets B and C:*

$$(A \equiv_{\mathrm{m}}^{\mathrm{p}} B \equiv_{\mathrm{m}}^{\mathrm{p}} C) \wedge (B \cap C = \emptyset) \Longrightarrow A \equiv_{\mathrm{m}}^{\mathrm{p}} B \cup C.$$

Observe that a set A is m-idempotent if and only if $\deg_{\mathrm{m}}^{\mathrm{p}}(A) = \mathcal{U}(A)$; that is, if the first inclusion in Theorem 2.21 is an equality. Furthermore, it is clear that whenever a set A is m-idempotent, the same holds for all sets $B \in \deg_{\mathrm{m}}^{\mathrm{p}}(A)$.

It turns out that our main question can be formulated equivalently with the notion of m-idempotence.

Proposition 2.24 SAT *is m-idempotent if and only if the union of two disjoint* NP*-complete sets is always* NP*-complete.*

So it is open whether the sets in the highest degree of NP are m-idempotent. A more general question is to ask whether there exists a set $A \in \mathrm{NP}$ such that the sets in the m-degree of A are m-idempotent. In other words, this is the question whether there is a set A in NP that has the least possible scope for $\mathcal{U}(A)$. Observe that such a set A must be in NP$-$P. Otherwise $0\Sigma^* \equiv_{\mathrm{m}}^{\mathrm{p}} (1\Sigma^* \cup \{\varepsilon\}) \equiv_{\mathrm{m}}^{\mathrm{p}} A$, which would imply that $\Sigma^* \equiv_{\mathrm{m}}^{\mathrm{p}} A$. This is a contradiction because A is nontrivial.

The next theorem states that the notion of p-selectivity can help us to find m-idempotent sets. More precisely, p-selectivity implies m-idempotence for any set outside P.

Theorem 2.25 *Let $A \notin \mathrm{P}$. If A is p-selective, then A is m-idempotent.*

Proof. Let A be a p-selective set outside P.

Claim 2.26 *For all disjoint $B, C \in \deg_{\mathrm{m}}^{\mathrm{p}}(A)$ it holds that the pair (B, C) is p-separable.*

Proof of the claim. Let $B, C \in \deg_{\mathrm{m}}^{\mathrm{p}}(A)$ such that $B \cap C = \emptyset$. Let $g, h \in \mathrm{FP}$ such that $B \leq_{\mathrm{m}}^{\mathrm{p}} A$ via g and $C \leq_{\mathrm{m}}^{\mathrm{p}} A$ via h. Furthermore, let $f \in \mathrm{FP}$ be the selector of A. We now define a set $S \in \mathrm{P}$ which separates the pair (B, C). Let

$$S =_{\mathrm{def}} \{x \mid f(g(x), h(x)) = g(x)\}.$$

Since $f, g, h \in \mathrm{FP}$, S clearly is in P. It remains to show that S separates (B, C), this means that for all x, it must hold that

$$\begin{aligned} x \in B &\Rightarrow x \in S \\ x \in C &\Rightarrow x \in \overline{S}. \end{aligned}$$

Let $x \in B$. Then $g(x) \in A$. Moreover, $h(x) \notin A$ since B and C are disjoint. Consequently, $f(g(x), h(x)) = g(x)$ and $x \in S$. If $x \in C$, $h(x) \in A$ and $g(x) \notin A$. We obtain $f(g(x), h(x)) = h(x)$ and $x \notin S$. This proves our claim. $\diamond$

Hence, we have shown that all disjoint $B, C \in \deg_{\mathrm{m}}^{\mathrm{p}}(A)$ are p-separable. We argue that this implies that A is m-idempotent. Let $B, C \in \deg_{\mathrm{m}}^{\mathrm{p}}(A)$ such that $B \cap C = \emptyset$ and $C \leq_{\mathrm{m}}^{\mathrm{p}} B$ via $f \in \mathrm{FP}$. We have to show that $B \cup C \equiv_{\mathrm{m}}^{\mathrm{p}} B$.

Clearly,

$$g(x) =_{\mathrm{def}} \begin{cases} x, & \text{if } x \in S \\ f(x), & \text{if } x \notin S \end{cases}$$

yields $B \cup C \leq_{\mathrm{m}}^{\mathrm{p}} B$.

Let us assume that $C = \overline{B}$. This implies $A \equiv_{\mathrm{m}}^{\mathrm{p}} \overline{A}$, because $A \equiv_{\mathrm{m}}^{\mathrm{p}} B \equiv_{\mathrm{m}}^{\mathrm{p}} \overline{B} \equiv_{\mathrm{m}}^{\mathrm{p}} \overline{A}$. From [Sel79] it then follows that $A \in \mathrm{P}$. This is a contradiction, so $C \neq \overline{B}$. By Proposition 2.8, $B \cup C \equiv_{\mathrm{m}}^{\mathrm{p}} B$. From this, it follows that $B \cup C \equiv_{\mathrm{m}}^{\mathrm{p}} A$. This finishes our proof. □

The proof of Theorem 2.25 does also show that every degree having the property that all pairs of disjoint sets are p-separable is m-idempotent.

Claim 2.26 then states that this holds in particular for degrees of p-selective sets. Recall that if all pairs of disjoint NP sets were p-separable, it would follow that P = UP [GS88] and that all sets in NP are m-idempotent. Moreover, we refer to Fortnow and Rogers [FR02] for an analysis of this hypothesis.

The next theorem gives a positive answer to the more general question whether NP contains m-idempotent sets under the assumption that NE ≠ coNE.

Theorem 2.27 *If* NE ≠ coNE*, there exists* $A \in \mathrm{NP} - \mathrm{coNP}$ *such that* A *is m-idempotent.*

Proof. We assume that NE ≠ coNE. This implies the existence of a tally set $T \in \mathrm{NP} - \mathrm{coNP}$ [BWSD77]. It then follows [Sel79] that there exists $A \equiv_{\mathrm{T}}^{\mathrm{p}} T$ such that $A \in \mathrm{NP}$ and A is p-selective. Suppose that $A \in \mathrm{NP} \cap \mathrm{coNP}$. Since $\mathrm{NP} \cap \mathrm{coNP}$ is closed under $\leq_{\mathrm{T}}^{\mathrm{p}}$-reducibility, this implies that $T \in \mathrm{NP} \cap \mathrm{coNP}$. As $T \in \mathrm{NP} - \mathrm{coNP}$, this is a contradiction. It follows that $A \in \mathrm{NP} - \mathrm{coNP}$. So we have identified a p-selective set in $\mathrm{NP} - \mathrm{coNP}$. In particular, $A \notin \mathrm{P}$. By Theorem 2.25, A is m-idempotent. □

We will now show that the complexity class EXP contains m-idempotent sets unconditionally.

Theorem 2.28 *There exists an m-idempotent set* $A \in \mathrm{EXP}$.

Proof. We first prove that there exists a tally set in $\mathrm{EXP} - \mathrm{P}$. We use a standard diagonalisation argument. Let $M_1, M_2, \ldots$ be an enumeration of all deterministic polynomial-time Turing machines. For all $i \geq 1$, let the running time of machine M_i be bounded by polynomial p_i. For technical reasons, we choose an enumeration of machines and polynomials such that for all $i \geq 1$, $p_i(i) \leq 2^i - 1$.

Define

$$H =_{\text{def}} \{0^i \mid M_i \text{ accepts } 0^i \text{ after at most } 2^i \text{ steps}\}.$$

Obviously, H is a tally set in EXP. We now prove that $H \notin \text{P}$. We suppose, for the sake of contradiction, that there exists an $x \geq 1$ such that M_x accepts H. We construct a Turing machine D as follows: On input 0^i, D simulates M_x on input 0^i. D accepts the input 0^i if and only if M_x rejects 0^i.

Such a machine clearly exists, so there exists $y \geq 1$ such that $D = M_y$. The running time of M_y on an input 0^n can be bounded by $p_x(n) + 1 \leq p_y(n)$.

We now run M_y on input 0^y. M_y then starts a simulation of M_x on input 0^y.

Let us assume that M_x accepts 0^y. By the definition of H, it must hold that M_y accepts 0^y after at most 2^y steps. Nevertheless, we have designed M_y to reject whenever M_x accepts, so this is a contradiction. Hence, M_x rejects 0^y. Similarly, it follows that M_y does not accept 0^y after at most 2^y steps. Since we know that M_y on input y halts within $p_x(y) + 1 \leq 2^y$ steps, it follows that M_y rejects 0^y within $p_y(y)$ steps. Again, this is a contradiction.

Consequently, no such machine M_x can exist, hence $H \notin \text{P}$. Hence, H is in $\text{EXP} - \text{P}$. Since H is a tally set, it follows [Sel79] that there exists $A \equiv_{\text{T}}^{\text{p}} H$ such that A is p-selective. It is easy to see that $A \in \text{EXP} - \text{P}$. Together with Theorem 2.25, this implies that A is m-idempotent. □

We have shown that there are sets in EXP for which the first inclusion in Theorem 2.21 is an equality. Under a reasonable assumption, we have shown the same for NP. We now take a look at the second inclusion.

We will show that there exists a set $A \in \text{NP}$ such that

$$\deg_{\text{m}}^{\text{p}}(A) \subsetneq \mathcal{U}(A) = (\mathcal{R}_{\text{m}}^{\text{p}}(A) \dot{\vee} \mathcal{R}_{\text{m}}^{\text{p}}(A)) - \{\emptyset\}$$

under the assumption that $\text{P} \neq \text{NP} \cap \text{coNP}$. We first prove that a set A cannot be m-idempotent if $\mathcal{R}_{\text{m}}^{\text{p}}(A)$ is closed under Boolean operations.

Theorem 2.29 *Let A be a nontrivial set. If $\mathcal{R}_{\text{m}}^{\text{p}}(A)$ is closed under Boolean operations then $\mathcal{U}(A) = \mathcal{R}_{\text{m}}^{\text{p}}(A) - \{\emptyset\}$.*

Proof. As $\mathcal{R}_{\text{m}}^{\text{p}}(A)$ is closed under Boolean operations, it is easy to see that $\mathcal{R}_{\text{m}}^{\text{p}}(A) = \mathcal{R}_{\text{m}}^{\text{p}}(A) \dot{\vee} \mathcal{R}_{\text{m}}^{\text{p}}(A) = \mathcal{R}_{\text{m}}^{\text{p}}(A) \vee \mathcal{R}_{\text{m}}^{\text{p}}(A)$. Hence it follows from Theorem 2.21 that we only have to show $\mathcal{R}_{\text{m}}^{\text{p}}(A) - \{\emptyset\} \subseteq \mathcal{U}(A)$.

Let $E \in \mathcal{R}_{\text{m}}^{\text{p}}(A) - \{\emptyset\}$ and Σ be an alphabet such that $A \cup E \subseteq \Sigma^*$, let $a \notin \Sigma$ be a new letter, and let $\Delta =_{\text{def}} \Sigma \cup \{a\}$. Assume $E \leq_{\text{m}}^{\text{p}} A$ via function $h \in \text{FP}$. Since $\mathcal{R}_{\text{m}}^{\text{p}}(A)$ is

closed under complementation it follows that $\overline{A} \leq^{\mathrm{p}}_{\mathrm{m}} A$, say via function $h' \in \mathrm{FP}$, and hence $\overline{A} \equiv^{\mathrm{p}}_{\mathrm{m}} A$. So we can assume that $E \neq \Sigma^*$, since otherwise $E \in \mathcal{U}(A)$ holds trivially because $A \cup \overline{A} = \Sigma^*$.

Let $a_0, e_0, e_1 \in \Sigma^*$ such that $a_0 \notin A$ and $e_0 \notin E$ and $e_1 \in E$.

We will define sets $A_0, A_1 \subseteq \Delta^*$ such that

- $A_0 \cap A_1 = \emptyset$,
- $A_0 \cup A_1 \equiv^{\mathrm{p}}_{\mathrm{m}} E$,
- $A_0 \equiv^{\mathrm{p}}_{\mathrm{m}} A_1 \equiv^{\mathrm{p}}_{\mathrm{m}} A$.

Notice that this implies $E \in \mathcal{U}(A)$.

We define $A_1 =_{\mathrm{def}} aA \cup E$ and $A_0 =_{\mathrm{def}} a(\Sigma^* - A)$. Clearly, $A_0 \cap A_1 = \emptyset$.

Claim 2.30 $A_0 \cup A_1 \equiv^{\mathrm{p}}_{\mathrm{m}} E$

Proof of the claim. It holds that $A_0 \cup A_1 = a\Sigma^* \cup E$. Let $f_1 : \Delta^* \to \Sigma^*$ be defined by

$$f_1(x) =_{\mathrm{def}} \begin{cases} x, & \text{if } x \in \Sigma^* \\ e_1, & \text{if } x \in a\Sigma^* \\ e_0, & \text{otherwise.} \end{cases}$$

Observe that $x \in a\Sigma^* \cup E \Leftrightarrow f_1(x) \in E$. As f_1 clearly is in FP, we have shown $A_0 \cup A_1 \leq^{\mathrm{p}}_{\mathrm{m}} E$. For the other direction, let $f_2 : \Sigma^* \to \Delta^*$ be defined by $f_2(x) = x$. Again, it is easy to see that $x \in E \Leftrightarrow f_2(x) \in a\Sigma^* \cup E$ and $f_2 \in \mathrm{FP}$. This proves the claim. $\diamond$

Claim 2.31 $A_0 \equiv^{\mathrm{p}}_{\mathrm{m}} A_1 \equiv^{\mathrm{p}}_{\mathrm{m}} A$

Proof of the claim. We will define functions $f_3, f_4, f_5 \in \mathrm{FP}$ such that $A_0 \leq^{\mathrm{p}}_{\mathrm{m}} A_1$ via f_3, $A_1 \leq^{\mathrm{p}}_{\mathrm{m}} A$ via f_4, and $A \leq^{\mathrm{p}}_{\mathrm{m}} A_0$ via f_5.

Define $f_3 : \Delta^* \to \Delta^*$ by

$$f_3(x) =_{\mathrm{def}} \begin{cases} ah'(z), & \text{if } x = az \text{ where } z \in \Sigma^* \\ e_0, & \text{otherwise.} \end{cases}$$

If $x \in A_0$, there exists $z \in \Sigma^* - A$ such that $x = az$. As h' reduces $\overline{A}$ to A, $ah'(z)$ is in A_1. If $x \notin A_0$, it either is of the form $x = az'$ where $z' \in A$ or $x \in \Delta^* - a\Sigma^*$. In the first case, $h'(z') \in \Sigma^* - A$, so $ah'(z) \notin A_1$. In the second case $f_3(x) = e_0 \notin A_1$. Obviously, $f_3 \in \mathrm{FP}$, hence $A_0 \leq^{\mathrm{p}}_{\mathrm{m}} A_1$.

We define $f_4 : \Delta^* \to \Sigma^*$ by

$$f_4(x) =_{\text{def}} \begin{cases} z, & \text{if } x = az \text{ where } z \in \Sigma^* \\ h(x), & \text{if } x \in \Sigma^* \\ a_0, & \text{otherwise.} \end{cases}$$

If $x \in A_1$, either $x = az$ where $z \in A$ or $x \in E$. In the first case, $f_4(x) = z \in A$. In the second case, $f_4(x) = h(x) \in A$ since h reduces E to A. If $x \notin A_1$, we distinguish three cases:

1. Assume $x \in a(\Sigma^* - A)$, i.e. there exists $z' \in \Sigma^* - A$ such that $x = az'$. Then $f_4(x) = z' \notin A$.
2. Assume $x \in \Sigma^* - E$. Then $f_4(x) = h(x) \notin A$.
3. Assume $x \in (\Delta^* a \Delta^*) - (a\Sigma^*)$. Then $f_4(x) = a_0 \notin A$.

Together with $f_4 \in \text{FP}$, we obtain $A_1 \leq^{\text{p}}_{\text{m}} A$.

Define $f_5 : \Sigma^* \to \Delta^*$ by $f_5(x) = ah'(x)$. If $x \in A$ then $h'(x) \in \Sigma^* - A$ hence $f_5(x) = ah'(x) \in a(\Sigma^* - A) \subseteq A_0$. If $x \notin A$ then $h'(x) \in A$ and hence $f_5(x) = ah'(x) \in A_0$. Obviously, $f_5 \in \text{FP}$. This proves our claim. $\diamond$

As argued above, we have now shown that $E \in \mathcal{U}(A)$. This proves $\mathcal{R}^{\text{p}}_{\text{m}}(A) - \{\emptyset\} \subseteq \mathcal{U}(A)$. Altogether, we obtain $\mathcal{U}(A) = \mathcal{R}^{\text{p}}_{\text{m}}(A) - \{\emptyset\}$. $\square$

Corollary 2.32 *Let A be a set. If $\mathcal{R}^{\text{p}}_{\text{m}}(A)$ is closed under Boolean operations, then A is not m-idempotent.*

Proof. Follows immediately from Proposition 2.22 and Lemma 2.29. $\square$

Consequently, no $\leq^{\text{p}}_{\text{m}}$-complete problem for a deterministic Turing machine time or space complexity class that is closed under $\leq^{\text{p}}_{\text{m}}$-reducibility can be m-idempotent.

The next theorem shows that unions of disjoints sets in NP can be much easier than the single components. In particular, there exists a degree $\deg^{\text{p}}_{\text{m}}(A)$ in NP$-$P such that all intermediate degrees can be reached by unions from disjoint sets from $\deg^{\text{p}}_{\text{m}}(A)$.

Theorem 2.33 *If $\text{P} \neq \text{NP} \cap \text{coNP}$, then there exists a set $A \in (\text{NP} \cap \text{coNP}) - \text{P}$ such that $\mathcal{U}(A) = \mathcal{R}^{\text{p}}_{\text{m}}(A) - \{\emptyset\} = \mathcal{R}^{\text{p}}_{\text{m}}(A) \dot{\vee} \mathcal{R}^{\text{p}}_{\text{m}}(A) - \{\emptyset\}$.*

Proof. By Lemma 2.29, it suffices to show under the assumption $\text{P} \neq \text{NP} \cap \text{coNP}$, that there exists a set $A \in (\text{NP} \cap \text{coNP}) - \text{P}$ such that $\mathcal{R}^{\text{p}}_{\text{m}}(A)$ is closed under Boolean operations.

Let us assume that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$. Then there exists a set $D \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$. Let c_D be the characteristic function of D. We now define a set A which has the desired properties.

We define

$$A =_{\mathrm{def}} \{H(x_1, \ldots, x_n), w_1, \ldots, w_n \mid H \text{ is a Boolean formula with variables } x_1, \ldots, x_n \text{ and } H(c_D(w_1), \ldots, c_D(w_n)) = 1\}.$$

It remains to show that

(1) $A \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$,

(2) $B \in \mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(A)$ implies $\overline{B} \in \mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(A)$,

(3) $B, C \in \mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(A)$ implies $B \cup C \in \mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(A)$.

We first argue for (1). A cannot be in P since it obviously holds that $D \leq^{\mathrm{p}}_{\mathrm{m}} A$. We have to show that $A \in \mathrm{NP} \cap \mathrm{coNP}$. Let M_1 and M_2 be nondeterministic machines such that the following holds for all x:

$$\begin{aligned} x \in D &\Leftrightarrow M_1 \text{ on input } x \text{ has (at least) one accepting path} \\ &\Leftrightarrow M_2 \text{ on input } x \text{ has no accepting paths.} \end{aligned}$$

Clearly, this implies that for all inputs x, precisely one of the machines M_1, M_2 produces an accepting path when running on input x. We informally describe a nondeterministic algorithm which decides A in polynomial time:

On input $\big(H(x_1, \ldots, x_n), w_1, \ldots, w_n\big)$ do the following:

1. $\mathtt{i} := 1$
2. Nondeterministically simulate $\mathtt{M}_1$ and $\mathtt{M}_2$ on input $\mathtt{w}_\mathtt{i}$.
3. On all nondeterministic paths of $\mathtt{M}_1$ and $\mathtt{M}_2$:
   (a) If the current path is rejecting, terminate the computation on this path.
   (b) If the current path accepts, set $\mathtt{c}_\mathtt{i} := 1$ if the path belongs to $\mathtt{M}_1$, set $\mathtt{c}_\mathtt{i} := 0$ if it belongs to $\mathtt{M}_2$.
   (c) If $\mathtt{i} < \mathtt{n}$, set $\mathtt{i} := \mathtt{i} + 1$ and goto 2.
   (d) If $\mathtt{i} = \mathtt{n}$, evaluate $\mathtt{H}(\mathtt{c}_1, \ldots, \mathtt{c}_\mathtt{n})$.
   (e) Accept if and only if $\mathtt{H}(\mathtt{c}_1, \ldots, \mathtt{c}_\mathtt{n}) = 1$.

Algorithm 2.1: A nondeterministic algorithm which decides A in polynomial time

Observe that the algorithm runs in polynomial time and produces an accepting path if and only if the input $\big(H(x_1,\ldots,x_n),w_1,\ldots,w_n\big)$ is in A. So we obtain $A \in \mathrm{NP}$. To see that $A \in \mathrm{coNP}$, note that $\overline{A} \leq^{\mathrm{p}}_{\mathrm{m}} A$ via the function $f(H(x_1,\ldots,x_n),w_1,\ldots,w_n) =_{\mathrm{def}} (\neg H(x_1,\ldots,x_n),w_1,\ldots,w_n)$. Hence, $A \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$.

We now prove (2) and (3). Let $B \leq^{\mathrm{p}}_{\mathrm{m}} A$ and $C \leq^{\mathrm{p}}_{\mathrm{m}} A$ via functions g_1, g_2, that means $x \in B \Leftrightarrow g_1(x) \in A$ and $x \in C \Leftrightarrow g_2(x) \in A$ holds for all x. Clearly, the function f defined above reduces $\overline{B}$ to B. It remains to show that $B \cup C \leq^{\mathrm{p}}_{\mathrm{m}} A$. This is accomplished by the function h.

$$h(x) =_{\mathrm{def}} \big(H_1 \vee H_2(x_1,\ldots,x_n,x'_1,\ldots,x'_m),w_1,\ldots,w_n,w'_1,\ldots,w'_m\big),$$

where

$$\big(H_1(x_1,\ldots,x_n),w_1,\ldots,w_n\big) =_{\mathrm{def}} g_1(x) \text{ and } \big(H_2(x'_1,\ldots,x'_m),w'_1,\ldots,w'_m\big) =_{\mathrm{def}} g_2(x).$$

It now holds that

$$\begin{aligned} x \in B \cup C &\Leftrightarrow (g_1(x) \in A) \vee (g_2(x) \in A) \\ &\Leftrightarrow h(x) \in A. \end{aligned}$$

Function h is computable in polynomial time. We obtain $B \cup C \leq^{\mathrm{p}}_{\mathrm{m}} A$ via function h and hence $B \cup C \in \mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(A)$. This finishes our proof. □

By Proposition 2.22, the set A in Theorem 2.33 cannot be m-idempotent. Informally speaking, the reason is that unions of sets in the degree of A can be too *easy* to be in the degree of A. As stated before, the question whether unions of NP-complete sets can be less than NP-complete is still open.

As a bibliographic note, the next proposition states a connection to secure *public-key cryptosystems* which is known due to Grollmann and Selman [GS88]:

Proposition 2.34 *If* SAT *is not m-idempotent, then there exist secure public-key cryptosystems.*

Proof. In the proof of Theorem 2.25, we show that if all disjoints sets in the degree of a set A are p-separable, then A is m-idempotent. Hence, if the NP-complete sets are not m-idempotent, there exist NP-complete sets B and C such that the pair (B,C) is not p-separable. Grollmann and Selman showed [GS88] that such a pair (B,C) exists if and only if secure public-key cryptosystems do exist. □

In the next section, we will show that the opposite can occur also, i.e. unions of equivalent sets can be harder than the original sets.

2.4.2 Harder Unions

Buhrman, Hoene, and Torenvliet [BHT98] showed unconditionally that there exists an $A \in \mathrm{EXP}-\mathrm{P}$ such that A is not EXP-complete and not m-idempotent. Recall that due to Corollary 2.32, no EXP-complete problem can be m-idempotent.

Theorem 2.35 *[BHT98] Let C be m-complete for* EXP*. Then C can be split into A and B such that*

- $A, B \in \mathrm{EXP}$,
- $A \equiv_{\mathrm{m}}^{\mathrm{p}} B$,
- $A \leq_{\mathrm{m}}^{\mathrm{p}} A \cup B = C$,
- *$A \cup B$ does not m-reduce to A, that means A, B are not m-complete for* EXP*.*

Corollary 2.36 *There exists $A \in \mathrm{EXP}$ such that*

$$\deg_{\mathrm{m}}^{\mathrm{p}}(A) \subsetneq \mathcal{U}(A) \subseteq \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A) \dot{\vee} \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A) = \mathrm{EXP},$$

hence A is not m-idempotent.

Proof. Let C be m-complete for EXP. By Theorem 2.35, C can be split into sets $A, B \in \mathrm{EXP}$ such that $A \equiv_{\mathrm{m}}^{\mathrm{p}} B$, $A \cap B = \emptyset$, $A \leq_{\mathrm{m}}^{\mathrm{p}} A \cup B$, and $A \cup B$ does not m-reduce to A. Hence, $A \cup B \in \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A) \dot{\vee} \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A)$. As $C = A \cup B$ is EXP-complete and $A \equiv_{\mathrm{m}}^{\mathrm{p}} B$, it follows that $\mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A) \dot{\vee} \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(A) = \mathrm{EXP}$. □

In this case, the union of sets in $\deg_{\mathrm{m}}^{\mathrm{p}}(A)$ can be harder than A. We will identify degrees in Θ_2^{P} for which the same holds. After this, we will construct such sets within NP.

The *chromatic number* of a graph G (in notation $\gamma(G)$) is the smallest number k such that G is k-colourable.

Definition 2.37 *Let $\gamma(G)$ be the chromatic number of a graph G, and let $k \geq 1$. Then*

$$\textsc{Colour}_k =_{\mathrm{def}} \{(G, a_1, \ldots, a_k) \,\big|\, G \text{ is a graph, } a_1, \ldots, a_k \geq 0 \text{ are pairwise different, and } \gamma(G) \in \{a_1, \ldots, a_k\}\}.$$

It follows from [CGH+88] that $\textsc{Colour}_k$ is $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete for $\mathrm{NP}(2k)$. Since $\textsc{Colour}_1$ is m-complete for NP(2), it follows that $\deg_{\mathrm{m}}^{\mathrm{p}}(\textsc{Colour}_1) = \{A \,\big|\, A \text{ is m-complete for NP(2)}\}$ and $\mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(\textsc{Colour}_1) \dot{\vee} \mathcal{R}_{\mathrm{m}}^{\mathrm{p}}(\textsc{Colour}_1) = \mathrm{NP}(2) \dot{\vee} \mathrm{NP}(2)$.

Theorem 2.38 *If the Boolean hierarchy over* NP *does not collapse to the second level, then there exist $A, B \in \mathrm{NP}(2)$ such that*

- $A \equiv^{\mathrm{p}}_{\mathrm{m}} B$,
- $A \leq^{\mathrm{p}}_{\mathrm{m}} A \cup B$,
- $A \cup B$ *does not m-reduce to* A.

Proof. We prove a stronger statement: For every $k \geq 1$, there exist disjoint sets A and B such that A and B are NP($2k$)-complete and $A \cup B$ is NP($4k$) complete. We consider variants of the well known graph colouring problem.

Let $k \geq 1$. Then the set $\textsc{Colour}_{2k}$ can be partitioned into L$-\textsc{Colour}_{2k} \subseteq \textsc{Colour}_{2k}$ and R$-\textsc{Colour}_{2k} \subseteq \textsc{Colour}_{2k}$, where

$$\begin{aligned} \text{L}-\textsc{Colour}_{2k} &=_{\text{def}} \{(G, a_i \text{ pairwise different}, \gamma(G) \in \{a_1, \ldots, a_k\}\}, \\ \text{R}-\textsc{Colour}_{2k} &=_{\text{def}} \{(G, a_1, \ldots, a_{2k}) \,\big|\, a_i \text{ pairwise different}, \gamma(G) \in \{a_{k+1}, \ldots, a_{2k}\}\}. \end{aligned}$$

It is easy to see that the following holds for all $k \geq 1$:

- $\textsc{Colour}_k \equiv^{\mathrm{p}}_{\mathrm{m}} \text{L}-\textsc{Colour}_{2k} \equiv^{\mathrm{p}}_{\mathrm{m}} \text{R}-\textsc{Colour}_{2k}$.
- $\text{L}-\textsc{Colour}_{2k} \cap \text{R}-\textsc{Colour}_{2k} = \emptyset$.
- $\text{L}-\textsc{Colour}_{2k} \cup \text{R}-\textsc{Colour}_{2k} = \textsc{Colour}_{2k}$.

In particular, the problem $\textsc{Colour}_2$ (which is m-complete for NP(4)) does neither m-reduce to L$-\textsc{Colour}_2$ nor to $\textsc{Colour}_1$ (both of which are m-complete for NP(2)) unless the Boolean hierarchy collapses to NP(2). □

Corollary 2.39 *There exist* $A, B \in \mathrm{NP}(2)$ *such that*

- $A \equiv^{\mathrm{p}}_{\mathrm{m}} B$,
- $A \leq^{\mathrm{p}}_{\mathrm{m}} A \cup B$,
- $A \cup B$ *does not m-reduce to* A.

unless the polynomial-time hierarchy collapses.

Under the assumption that the Boolean hierarchy over NP does not collapse, it follows that $\deg^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1) \subsetneq \mathcal{U}(\textsc{Colour}_1)$. Hence, the NP(2)-complete sets are not m-idempotent. This indicates that the converse of Corollary 2.32 does not hold.

The next theorem states that $\textsc{Colour}_1$ is an example for which $\mathcal{U}(\textsc{Colour}_1)$ lies strictly between $\deg^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1)$ and $\mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1)\dot{\vee}\mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1) = \mathrm{NP}(2)\dot{\vee}\mathrm{NP}(2)$.

First we prove a lemma.

Lemma 2.40 *For all sets A, the following are equivalent:*

1. $\mathcal{U}(A) \cap \mathrm{P} \neq \emptyset$
2. $\mathcal{U}(A) \supseteq \mathrm{P} - \{\emptyset\}$
3. $A \equiv^{\mathrm{p}}_{\mathrm{m}} \overline{A}$.

Proof. Let A be a set.

For the implication from item 1 to item 2, assume that there exists a set $B \in \mathcal{U}(A) \cap \mathrm{P}$. By definition, $\mathcal{U}(A)$ contains all sets in $\deg^{\mathrm{p}}_{\mathrm{m}}(B) = \mathrm{P} - \{\emptyset\}$, i.e. $\mathcal{U}(A) \supseteq \mathrm{P} - \{\emptyset\}$. For the implication from 2 to 3, assume that $\mathcal{U}(A) \supseteq \mathrm{P} - \{\emptyset\}$. Hence, there exists $E \in \mathrm{P} - \{\emptyset\}$ such that $E \in \mathcal{U}(A)$. So there exist sets C and D such that $C \equiv^{\mathrm{p}}_{\mathrm{m}} D \equiv^{\mathrm{p}}_{\mathrm{m}} A$, $C \cap D = \emptyset$, and $E \equiv^{\mathrm{p}}_{\mathrm{m}} C \cup D$. Observe that $C \cup D \in \mathrm{P}$ and $\overline{C \cup D} \cap D = \emptyset$. Therefore, it is easy to see that $\overline{C} = \overline{C \cup D} \cup D \equiv^{\mathrm{p}}_{\mathrm{m}} D$. We now have $A \equiv^{\mathrm{p}}_{\mathrm{m}} C \equiv^{\mathrm{p}}_{\mathrm{m}} D \equiv^{\mathrm{p}}_{\mathrm{m}} \overline{C}$. We conclude $A \equiv^{\mathrm{p}}_{\mathrm{m}} \overline{A}$. For the implication from 3 to 1, we assume $A \equiv^{\mathrm{p}}_{\mathrm{m}} \overline{A}$. Hence, $A \neq \emptyset$. Let $a \in A$. Trivially, $A \equiv^{\mathrm{p}}_{\mathrm{m}} A - \{a\} \equiv^{\mathrm{p}}_{\mathrm{m}} \overline{A}$. Therefore, $\Sigma^* - \{a\} = A - \{a\} \cup \overline{A} \in \mathcal{U}(A)$. □

Theorem 2.41 *If the Boolean hierarchy over* NP *does not collapse to* NP(2)*, it holds that*

$$\deg^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1) \subsetneq \mathcal{U}(\textsc{Colour}_1) \subsetneq \mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1)\dot{\vee}\mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1).$$

Proof. Due to Theorem 2.21 and Theorem 2.38 it suffices to show that there exists $D \in \mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1)\dot{\vee}\mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1) = \mathrm{NP}(2)\dot{\vee}\mathrm{NP}(2)$ such that $D \notin \mathcal{U}(\textsc{Colour}_1)$. Clearly, $\mathrm{NP}(2)\dot{\vee}\mathrm{NP}(2)$ contains P, so let $D \in \mathrm{P}$. As we assumed that the Boolean hierarchy does not collapse to NP(2), it follows that $\mathrm{NP}(2) \neq \mathrm{coNP}(2)$ and hence $\textsc{Colour}_1 \not\equiv^{\mathrm{p}}_{\mathrm{m}} \overline{\textsc{Colour}_1}$. From Lemma 2.40 we then obtain $\mathcal{U}(\textsc{Colour}_1) \cap \mathrm{P} = \emptyset$. Consequently, $D \notin \mathcal{U}(\textsc{Colour}_1)$. □

Corollary 2.42 *It holds that*

$$\deg^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1) \subsetneq \mathcal{U}(\textsc{Colour}_1) \subsetneq \mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1)\dot{\vee}\mathcal{R}^{\mathrm{p}}_{\mathrm{m}}(\textsc{Colour}_1)$$

unless the polynomial-time hierarchy collapses.

2.5 Summary and Outlook

In this chapter, we systematically studied the complexity of unions of disjoint, equivalent sets. We glanced at the situation in the recursion theory setting and observed that the union of two disjoint RE-complete sets is always RE-complete.

We then explained that one cannot expect similar absolute results for NP. We presented necessary and sufficient conditions for an affirmative answer to the main open question of whether the union of two disjoint NP-complete sets is always NP-complete.

We proved that under reasonable assumptions, the union of two disjoint NP-complete sets cannot become too easy: More precisely, we showed that the union of two disjoint NP-complete sets belongs to the class High_1, the first level of the high-hierarchy. We give further evidence that unions of disjoint NP-complete sets are not far from being NP-complete by showing that under a reasonable assumption, the union of an NP-complete set with a disjoint set in NP is nonuniformly NP-complete.

We then abstracted from the main problem and studied the more general question of how the complexity of unions of disjoint, equivalent sets can change. As a useful tool, we introduced the notion of m-idempotence. In order to show that NP$-$P contains m-idempotent sets, we proved that every p-selective set is m-idempotent. We obtained that if NE $\neq$ coNE, then there exists $A \in \mathrm{NP}-\mathrm{coNP}$ such that A is m-idempotent. Although it remained open whether the sets in the highest degree of NP, i.e., the NP-complete sets are m-idempotent, we were able to show that NP is likely to contain degrees which have that property.

In contrast to that, we also showed that NP$-$P also contains degrees with opposite properties: NP$-$P contains disjoint, equivalent sets whose union can be arbitrary simple unless $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$. Moreover, if the polynomial hierarchy is strict, then it is possible for the union of two disjoint NP(2)-sets to be harder than either of its components.

Furthermore, we remark that under a strong hypothesis, one can also show that the same can happen within NP$-$coNP. More precisely, under a strong assumption, it is possible to show that there exist m-equivalent disjoint sets E and F in NP such that $E \cup F$ is harder than E. This assumption utilises the notion of *immunity*:

Definition 2.43 *A set L is* immune *to a complexity class $\mathcal{C}$, or* $\mathcal{C}$-immune*, if L is infinite and no infinite subset of L belongs to $\mathcal{C}$. A set L is* bi-immune *to a complexity class $\mathcal{C}$, or* $\mathcal{C}$-bi-immune*, if both L and $\overline{L}$ are $\mathcal{C}$-immune.*

Theorem 2.44 *[GSTW07] If* NP *has* $\mathrm{NP} \cap \mathrm{coNP}$*-bi-immune sets and* $\mathrm{NP} \cap \mathrm{coNP}$ *has* P*-bi-immune sets, then there exist disjoint sets $E, F \in \mathrm{NP} - \mathrm{coNP}$ such that $E \equiv_{\mathrm{m}}^{\mathrm{p}} F$, but $E \cup F \not\leq_{\mathrm{m}}^{\mathrm{p}} E$.*

As a bibliographical remark, we note that the marked union can in fact lower low-hierarchy[1] complexity [HJRW98]. Contrary to the results in this section however, the involved sets are not equivalent.

[1] The low-hierarchy was defined in [Sch83]. It is related to the high-hierarchy. These hierarchies provide a yardstick to measure the complexity of sets that are known to be in NP but that are seemingly neither in P nor NP-complete

Chapter 3

NP-Hard Sets and Faulty Data

In Chapter 2, we investigated unions of disjoint NP-complete sets. We showed that when we add NP-complete data to an NP-complete set, then the resulting set retains much of its complexity, i.e., it is NP-complete with respect to more general reducibilities.

In this chapter, we follow this track and analyse what happens when we alter NP-hard sets in a more general way. This includes adding faulty data (*false positives*) to NP-hard sets, or removing reasonable data (*false negatives*). By dropping the condition that the data we add must be NP-complete itself, we shift our focus to the question of how NP-hard sets can cope with faulty data.

Even small amounts of faulty data can obscure reasonable information. For instance, by filling more and more whitespaces of a printed text with arbitrary letters, it can become quite difficult to understand the original meaning of the text.

The same holds true for NP-complete sets. Take for instance SAT, the set of all satisfiable formulas. By adding false positives to SAT, i.e., some unsatisfiable formulas, we can actually lose information: If we overdo it, we end up with $\mathrm{SAT} \cup \overline{\mathrm{SAT}} = \Sigma^*$, and by this definitely lose NP-completeness. But how much false positive data can NP-hard sets handle, i.e., how many false positives can we add such that the resulting set stays NP-hard? Alternatively, how much effort is needed to extract the original information?

In this chapter, we investigate how polynomial time reductions can cope with false positives. More precisely, we consider NP-hard sets for several polynomial time reductions and add false positives to the sets.

Moreover, we study the effects of more general kinds of faulty data. We investigate how polynomial time reductions can handle combinations of both, false positives and false negatives.

This also explains why we consider NP-hard sets instead of restricting ourselves to NP-complete sets. As we allow arbitrary kinds of faulty data, we cannot guarantee that the resulting set remains within NP.

Our research is related to the notion of *program self-correction* which was introduced by Blum, Luby, and Rubinfeld [BLR93]. That notion addresses a fundamental question regarding software reliability: Can one increase the reliability of existing software without understanding the way it works? More precisely, let P be a program that is designed to solve a problem L. However, we do not know whether P is correct. Is it possible to write an auxiliary program M that uses P such that if P errs only on a small fraction of the inputs, then with high probability M corrects the errors made by P? So M has to find the right answer with high probability by calling P on several inputs.

Our investigations of the consequences of faulty data are related to a *deterministic* variant of self-correction which we will introduce in the next section. In this case, the error probability of the wrapping machine M must be 0, i.e., M must achieve certainty about the question of whether the input belongs to L. As in the original definition, we also demand that M runs in polynomial time.

3.1 Weak Deterministic Self-Correction

We introduce the notion of weak deterministic self-correction which is a deterministic variant of (probabilistic) self-correction [BLR93]. The prefix *weak* indicates that our notion of deterministic self-correction does not necessarily imply probabilistic self-correction in the sense of Blum, Luby, and Rubinfeld [BLR93]. The difference is as follows: For weak deterministic self-correction, we require that a sparse amount of errors can be corrected by a deterministic polynomial-time corrector. For probabilistic self-correction however, a probabilistic polynomial-time corrector must be able to correct up to $2^n/p(n)$ errors for some polynomial p.

Definition 3.1 *L is* weakly deterministically self-correctable *if for every polynomial q there exists a polynomial-time machine M such that $L \leq_{\mathrm{T}}^{\mathrm{p}} P$ via M whenever the census of $L \triangle P$ is bounded by q. If M queries nonadaptively, then L is* nonadaptively weakly deterministically self-correctable.

The set P in the definition formalises a program for L that errs on at most $q(n)$ inputs of length n. So L is weakly deterministically self-correctable if there exists an auxiliary machine M that corrects *all* programs that err on at most $q(n)$ inputs of length n. The next theorem shows that such a universal M already exists if the single programs can be corrected with possibly different machines. This establishes a connection between weak deterministic self-correction and robustness against false positives.

Theorem 3.2 *L is weakly deterministically self-correctable if $L \leq^{\mathrm{p}}_{\mathrm{T}} L \triangle S$ holds for all sparse sets S.*

Proof. $\Rightarrow$: This follows directly from Definition 3.1.

$\Leftarrow$: Assume that L is not weakly deterministically self-correctable. So there exists a polynomial q such that

$$\forall \text{polynomial-time machine } M,\ \exists T \subseteq \Sigma^* \ [\mathrm{census}_T \leq q \text{ and } L \neq L(M^{L \triangle T})]. \tag{3.1}$$

We construct a sparse S such that $L \not\leq^{\mathrm{p}}_{\mathrm{T}} L \triangle S$. The construction is stagewise where in step i we construct a finite set S_i such that $S_1 \subseteq S_2 \subseteq \cdots$ and $S =_{\mathrm{def}} \bigcup_{i \geq 1} S_i$. Let $M_1, M_2, \ldots$ be an enumeration of all deterministic, polynomial-time Turing machines such that M_i runs in time $n^i + i$. Let $S_0 = \emptyset$. For $i \geq 1$, the set S_i is constructed as follows:

Choose n sufficiently large such that $S_{i-1} \subseteq \Sigma^{<n}$ and changing the oracle with respect to words of length $\geq n$ will not affect the computations that were simulated in earlier steps. Choose a finite $T_i \subseteq \Sigma^{\geq n}$ and an $x_i \in \Sigma^*$ such that $\mathrm{census}_{T_i} \leq q$ and

$$x_i \in L \Leftrightarrow x_i \notin L(M_i^{L \triangle (S_{i-1} \cup T_i)}). \tag{3.2}$$

Let $S_i =_{\mathrm{def}} S_{i-1} \cup T_i$.

We argue that the choice of T_i is possible. If not, then for all finite $T_i \subseteq \Sigma^{\geq n}$ where $\mathrm{census}_{T_i} \leq q$ and all $x_i \in \Sigma^*$ it holds that

$$x_i \in L \Leftrightarrow x_i \in L(M_i^{L \triangle (S_{i-1} \cup T_i)}).$$

Let M be the polynomial-time machine obtained from M_i when queries of length $< n$ are answered according to $(L \triangle S_{i-1}) \cap \Sigma^{<n}$ (which is a finite set). So for all T where $\mathrm{census}_T \leq q$ and all $x_i \in \Sigma^*$ it holds that

$$x_i \in L(M^{L \triangle T}) \Leftrightarrow x_i \in L(M_i^{L \triangle (S_{i-1} \cup (T \cap \Sigma^{\geq n}))}) \Leftrightarrow x_i \in L(M_i^{L \triangle (S_{i-1} \cup T')}) \Leftrightarrow x_i \in L,$$

where

$$T' = T \cap \Sigma^{\geq n} \cap \Sigma^{\leq |x_i|^i + i}.$$

Hence $L = L(M^{L \triangle T})$ for all T where $\mathrm{census}_T \leq q$. So M contradicts (3.1). It follows that the choice of T_i is possible and hence also the construction of S.

The equivalence (3.2) ensures that

$$\forall i \geq 1 \ [x_i \in L \Leftrightarrow x_i \notin L(M_i^{L \triangle S})]$$

and hence $L \not\leq^{\mathrm{p}}_{\mathrm{T}} L \triangle S$. $\square$

Corollary 3.3 *L is nonadaptively weakly deterministically self-correctable $\Leftrightarrow$ $L \leq_{\mathrm{tt}}^{\mathrm{p}} L \triangle S$ for all sparse S.*

Proof. This is shown with the same proof as Theorem 3.2, except that all machines query nonadaptively. □

3.2 Partly Corrupt NP-Hard Sets

In this section we investigate how polynomial reductions can cope with sparse amounts of false data in sets that are hard for NP with respect to various reducibilities. In Section 3.2.1 we show that altering sparse information in m-hard sets results in sets that are at least tt-hard. In particular, all m-complete sets are nonadaptively weakly deterministically self-correctable. In Section 3.2.2 we prove that adding a sparse amount of false positives to dtt-hard sets results in sets that are at least T-hard. However, it remains open whether dtt-complete sets are weakly deterministically self-correctable. At the end of Section 3.2.2, we give evidence that this open problem is rather difficult to solve.

Finally we show in Section 3.2.4 that many-one reductions, bounded truth-table reductions, and disjunctive truth-table reductions are provably too weak to handle false positives in SAT.

Several proofs in this section are based upon the left-set technique by Ogihara and Watanabe [OW91].

3.2.1 Many-One Reductions

Here we alter sparse information in m-hard sets for NP. Under the assumption P $\neq$ NP, the resulting sets are still ctt-hard when we add a sparse amount of false positives. Under the same assumption, we obtain that m-hard sets for NP remain dtt-hard when we remove a sparse amount of reasonable information, i.e., when we allow a sparse amount of false negatives.

Without the assumption P $\neq$ NP, we can show that the resulting sets are at least tt-hard. On the technical side we extend an idea from [GPSZ06] which shows how many-one queries to NP-hard sets can be reformulated. In this way, for a given query we can generate a polynomial number of different, but equivalent queries (Lemma 3.4). From this we easily obtain the conditional ctt-hardness, dtt-hardness and the unconditional tt-hardness of the altered NP-hard set. As a corollary, all m-complete sets for NP are nonadaptively weakly deterministically self-correctable.

Lemma 3.4 *Let L be $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard for* NP *and let $B \in$ NP. Then there exists a polynomial r such that for every polynomial q there is a polynomial-time algorithm $\mathcal{A}$ such that $\mathcal{A}$ on input x,*

- *either correctly decides the membership of x in B or*
- *outputs $k = q(r(|x|))$ pairwise disjoint $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$ such that for all $i \in [1, k]$, it holds that*

$$x \in B \Leftrightarrow y_i \in L.$$

Proof. Choose $R \in$ P and a polynomial p such that $x \in B$ if and only if there exists a $w \in \Sigma^{p(|x|)}$ such that $(x, w) \in R$. For $x \in B$, let w_x be the lexicographically greatest such witness. The following set is in NP.

$$\mathrm{Left}(B) = \{(x, y) \,\big|\, x \in B, |y| = p(|x|), y \leq w_x\}.$$

So there is a many-one reduction f from Left(B) to L. In particular, there exists a polynomial r such that for all $x \in \Sigma^*$ and all $y \in \Sigma^{p(|x|)}$, $|f(x, y)| \leq r(|x|)$. Choose a polynomial q. We now describe the algorithm $\mathcal{A}$.

```
 1.    m := p(n)
 2.    if (x, 1^m) ∈ R then accept
 3.    l := 0^m
 4.    if f(x, l) = f(x, 1^m) then reject
 5.    Q = {f(x, l)}
 6.    while |Q| ≤ q(r(n)) do
 7.       choose a ∈ Σ^m such that l ≤ a ≤ 1^m, f(x, a) ∈ Q,
          and f(x, a + 1) ∉ Q
 8.       l := a + 1
 9.       if (x, a) ∈ R then accept
10.       if f(x, l) = f(x, 1^m) then reject
11.       Q = Q ∪ {f(x, l)}
12.    end while
13.    output Q
```

Algorithm 3.1: $\mathcal{A}$, input x, $|x| = n$

Observe that algorithm $\mathcal{A}$ places a string $f(x, l)$ in Q only if $f(x, l) \neq f(x, 1^m)$. Thus $f(x, 1^m)$ is never placed in Q. So in step 8, $f(x, l) \in Q$ and $f(x, 1^m) \notin Q$. Therefore, with

a binary search we find the desired a in polynomial time. Each iteration of the while loop adds a new string to Q or decides the membership of x in B. Thus the algorithm works in polynomial time and when it outputs some Q, then $|Q| = q(r(|x|))$ and words in Q have lengths $\leq r(n)$.

Claim 3.5 *If the algorithm outputs some Q, then for all $y \in Q$, $x \in B \Leftrightarrow y \in L$.*

Proof of the claim. If $x \notin B$, then for all $c \in [0^m, 1^m]$, $(x, c) \notin \mathrm{Left}(B)$. Observe that the algorithm places a string y in Q only if $y = f(x, a)$ where $a \in [0^m, 1^m]$. Since f is a many-one reduction from $\mathrm{Left}(B)$ to L, no string from Q belongs to L.

From now on we assume $x \in B$. We prove the claim by induction. Initially, $Q = \{f(x, 0^m)\}$. Clearly, $x \in B \Leftrightarrow (x, 0^m) \in \mathrm{Left}(B)$. Since f is a many-one reduction from $\mathrm{Left}(B)$ to L, the claim holds initially. Assume that the claim holds before an iteration of the while loop. The while loop finds a node a such that $f(x, a) \in Q$, but $f(x, a+1) \notin Q$. From $f(x, a) \in Q$ and $x \in B$ it follows (by induction hypothesis) that $f(x, a) \in L$. Thus $(x, a) \in \mathrm{Left}(B)$ which implies $a \leq w_x$. At this point the algorithm checks whether a is a witness of x. If so, then it accepts and halts. Otherwise, we have $a + 1 \leq w_x$. Thus $(x, a+1) \in \mathrm{Left}(B)$ and $f(x, a+1) \in L$. So the claim also holds after the iteration of the while loop. ◇

Claim 3.6 *If the algorithm accepts x (resp., rejects x), then $x \in B$ (resp., $x \notin B$).*

Proof of the claim. The algorithm accepts x only if it finds a witness of x. Thus if the algorithm accepts, then $x \in B$. The algorithm rejects only if $f(x, l) = f(x, 1^m)$. Note that $f(x, l) \in Q$, so by the previous claim, $x \in B \Leftrightarrow f(x, l) \in L$. Observe that $(x, 1^m) \notin \mathrm{Left}(B)$. Thus $f(x, l) = f(x, 1^m) \notin L$ and hence $x \notin B$. ◇

This finishes the proof of the lemma. □

Theorem 3.7 *The following statements are equivalent.*

1. $\mathrm{P} \neq \mathrm{NP}$
2. *If L is $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard for* NP *and S is sparse, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-hard for* NP.

Proof. $\neg 1 \Rightarrow \neg 2$: If $\mathrm{P} = \mathrm{NP}$, then $L = \Sigma^* - \{0\}$ and $S = \{0\}$ are counter examples for statement 2.

$1 \Rightarrow 2$: Assume $\mathrm{P} \neq \mathrm{NP}$ and let L and S be as in statement 2. If $\overline{L}$ is sparse, then there exist sparse coNP-hard sets and hence $\mathrm{P} = \mathrm{NP}$ [For79]. So it follows that $\overline{L}$ is not sparse and $L \cup S \neq \Sigma^*$. Hence there exist elements $x_0 \notin L \cup S$ and $x_1 \in L \cup S$.

Let $B \in \mathrm{NP}$; we show $B \leq^{\mathrm{p}}_{\mathrm{ctt}} L \cup S$. First, choose the polynomial r according to Lemma 3.4. Let q be a polynomial such that $|S \cap \Sigma^{\leq n}| < q(n)$. Lemma 3.4 provides an algorithm $\mathcal{A}$

that on input x either correctly decides the membership of x in B, or outputs $k = q(r(|x|))$ pairwise disjoint $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$ such that for all $i \in [1, k]$, $(x \in B \Leftrightarrow y_i \in L)$. Define the following polynomial-time-computable function.

$$g(x) =_{\text{def}} \begin{cases} x_0 & : \text{ if } \mathcal{A}(x) \text{ rejects} \\ x_1 & : \text{ if } \mathcal{A}(x) \text{ accepts} \\ (y_1, \ldots, y_k) & : \text{ if } \mathcal{A}(x) \text{ returns } Q = \{y_1, \ldots, y_k\} \end{cases}$$

Note that in the last case, $k = q(r(|x|))$ and the y_i have lengths $\leq r(|x|)$. So at least one of the y_i does not belong to S. From $\mathcal{A}$'s properties stated in Lemma 3.4 it follows that $B \leq^{\text{p}}_{\text{ctt}} L \cup S$ via g. □

Similarly, we can show the following:

Theorem 3.8 *The following statements are equivalent.*

1. $\text{P} \neq \text{NP}$
2. *If L is $\leq^{\text{p}}_{\text{m}}$-hard for* NP *and S is sparse, then $L - S$ is $\leq^{\text{p}}_{\text{dtt}}$-hard for* NP.

Proof. The proof is basically the same as the proof of Theorem 3.7. Lemma 3.4 provides an algorithm which either decides membership for an input x directly or outputs sufficiently many elements (i.e., more than census$_S$-many elements) which are equivalent with respect to their membership in L. Hence it follows that whenever the algorithm cannot solve the problem directly, there is at least one element from $\overline{S}$ among the elements which the algorithm outputs. This yields the dtt-reduction. □

It is not difficult to see that a similar approach also yields the following: Unless $\text{P} = \text{NP}$, $\leq^{\text{p}}_{\text{m}}$-hard sets for NP remain $\leq^{\text{p}}_{\text{m}}$-hard when the sparse information S subtracted is in P: Applying the algorithm in Lemma 3.4, we simply have to check for each element in Q whether or not it belongs to S. Since $S \in \text{P}$, this can be done in polynomial time. As soon as an element from outside S is found, the m-reduction simply outputs this element. This observation already follows from [GPSS06]. There the authors prove that NP-complete sets are robust against sparse *p-selective* sets. Theirs is a stronger result but also requires a far more complicated argumentation.

We now consider combinations of false negatives and false positives.

Theorem 3.9 *If L is $\leq^{\text{p}}_{\text{m}}$-hard for* NP *and S is sparse, then $L \triangle S$ is $\leq^{\text{p}}_{\text{tt}}$-hard for* NP.

Proof. For $B \in \text{NP}$ we show $B \leq^{\text{p}}_{\text{tt}} L \triangle S$. First, choose the polynomial r according to Lemma 3.4. Let q be a polynomial such that $2 \cdot |S \cap \Sigma^{\leq n}| < q(n)$. Lemma 3.4 provides

an algorithm $\mathcal{A}$ that on input x either correctly decides the membership of x in B, or outputs $k = q(r(|x|))$ pairwise disjoint $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$ such that for all $i \in [1,k]$, $(x \in B \Leftrightarrow y_i \in L)$.

We describe a polynomial-time oracle machine M on input x: If $\mathcal{A}(x)$ accepts, then M accepts. If $\mathcal{A}(x)$ rejects, then M rejects. Otherwise, $\mathcal{A}(x)$ returns elements $y_1, \ldots, y_k \in \Sigma^{\leq r(|x|)}$. M queries all these elements and accepts if and only if at least $k/2$ of the answers were positive.

Clearly, if $\mathcal{A}(x)$ accepts or rejects, then $(x \in B \Leftrightarrow M(x)$ accepts). So assume that $\mathcal{A}(x)$ returns elements y_i. S contains less than $q(r(|x|))/2 = k/2$ words of length $\leq r(|x|)$. So more than $k/2$ of the y_i do not belong to S. Hence, for more than $k/2$ of the y_i it holds that

$$x \in B \quad \Leftrightarrow \quad y_i \in L \quad \Leftrightarrow \quad y_i \in L \triangle S.$$

Therefore, x belongs to B if and only if at least $k/2$ of the y_i belong to $L \triangle S$. This shows that $B \leq^{\mathrm{p}}_{\mathrm{tt}} L \triangle S$ via M. □

Corollary 3.10 *All $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets for* NP *are nonadaptively weakly deterministically self-correctable.*

Proof. Let L be $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for NP. By Corollary 3.9, for all sparse S, $L \leq^{\mathrm{p}}_{\mathrm{tt}} L \triangle S$. By Corollary 3.3, L is nonadaptively weakly deterministically self-correctable. □

3.2.2 Disjunctive Truth-Table Reductions

In this section we analyse how disjunctive truth-table reductions can handle false positives. We show that the union of dtt-hard sets with arbitrary sparse sets is always T-hard. Moreover, we explain why the similar question for false-negatives (or even combinations of both) is very difficult. In the restricted case of 2dtt-hard sets however, we can show that such sets stay dtt-hard when we subtract a sparse amount of data, i.e., when we allow a sparse amount of false negatives.

Theorem 3.11 *Let L be $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for* NP*, and let S be a sparse set. Then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*.*

Proof. Let $L \subseteq \Sigma^*$ and $S \subset \Sigma^*$ be as above, and let M be a nondeterministic Turing machine whose running time on input x is bounded by polynomial p. Without loss of generality, we assume that on input x, M develops precisely $2^{p(|x|)}$ nondeterministic computation paths. Each path can hence be identified by a word $z \in \{0,1\}^{p(|x|)}$. For a path $z \in \{0,1\}^{p(|x|)}, z \neq 1^{p(|x|)}$, we denote the path on the right of z with $z+1$.

Let A be the language accepted by M. We will show that $A \leq_{\mathrm{T}}^{\mathrm{p}} L \cup S$. The left-set of A is defined as

$$\mathrm{Left}(A) =_{\mathrm{def}} \{(x, y) \mid \text{ there exists a } z \geq y \text{ such that } M \text{ accepts } x \text{ along } z\}.$$

From $A \in \mathrm{NP}$ it follows that $\mathrm{Left}(A) \in \mathrm{NP}$. Since L is $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-hard for NP, there exists a function f such that $\mathrm{Left}(A) \leq_{\mathrm{dtt}}^{\mathrm{p}} L$ via $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$, $f \in \mathrm{FP}$.

By the definition of $\leq_{\mathrm{dtt}}^{\mathrm{p}}$ it holds that $(x, y) \in \mathrm{Left}(A) \Leftrightarrow f(x, y) \cap L \neq \emptyset$.

Without loss of generality, we assume that M does neither accept on its first computation path nor on its last path. Furthermore, we define $f_+(x, y) =_{\mathrm{def}} f(x, y) \cap (L \cup S)$.

Let q be a polynomial such that for all $x \in \Sigma^*$ and for all $y \in \{0,1\}^{p(|x|)}$ it holds that

$$q(|x|) > \sum_{i=0}^{|f(x,y)|} \mathrm{census}_S(i)$$

We will construct a deterministic polynomial time oracle machine N such that the following holds for all x:

$$\begin{aligned} x \in A &\Leftrightarrow \exists y \in \{0,1\}^{p(|x|)} \big((x, y) \in \mathrm{Left}(A)\big) \\ &\Leftrightarrow \exists y \in \{0,1\}^{p(|x|)} (f(x, y) \cap L \neq \emptyset) \\ &\Leftrightarrow N^{L \cup S} \text{ accepts } x. \end{aligned}$$

We describe how N works on input $x \in \Sigma^*$.

```
 1. i := 0
 2. z_i := 1^{p(|x|)}      //current position in tree, start with rightmost path
 3. F_i := f_+(x, z_i)      //positively answered oracle queries
 4. while i < q(|x|)
 5.     if f_+(x, 0^{p(|x|)}) − F_i = ∅ then reject
 6.     determine z_{i+1} ∈ {0,1}^{p(|x|)}, z_{i+1} < z_i such that (f_+(x, z_{i+1}) − F_i) ≠ ∅
        and (f_+(x, z_{i+1} + 1) − F_i) = ∅
 7.     if M accepts along z_{i+1} then accept
 8.     F_{i+1} := F_i ∪ f_+(x, z_{i+1})      //cull new element from S − L
 9.     i := i + 1
10. end while
11. reject     //this statement is never reached
```

Algorithm 3.2: OPTM N, input x

We show that N is a polynomial time machine: As the number of passes of the while loop is bounded by a polynomial, it suffices to argue that step 6 can be performed in polynomial time. Note that N can compute the set $f_+(x,z)$ by querying the oracle $L \cup S$ for all elements in $f(x,z)$. Step 6 is an easy binary search: Start with $z_1 := 0^{p(|x|)}$ and $z_2 := 1^{p(|x|)}$. Let z' be the middle element between z_1 and z_2. If $\big(f_+(x,z') - F_i\big) = \emptyset$ then $z_2 := z'$ (i.e., the binary search continues on the left) else $z_1 := z'$ (i.e., the binary search continues on the right). Choose the middle element between z_1 and z_2 and repeat the above steps until a suitable path is found. Consequently, we obtain that N runs in polynomial time.

We now argue that the algorithm is correct, i.e., N accepts x if and only if $x \in A$.

For the only-if part, let us assume that N accepts x. If N accepts in line 7 then it has found an accepting path of M on input x. Hence, $x \in A$. This proves the only-if part.

We now prove the if-part. Let $x \in A$, so there exists a rightmost accepting path of M on input x, say z_{right}. As M does neither accept on the leftmost nor on the rightmost path, it holds that $0^{p(|x|)} < z_{\text{right}} < 1^{p(|x|)}$.

We explain that during the execution of the while loop, the accepting path z_{right} is found.

Claim 3.12 *For $0 \leq i \leq q(|x|)$, if z_{right} was not found during the first i iterations of the while loop, then the following holds after i iterations:*

1. $\#F_i \geq i$
2. $z_i > z_{\text{right}}$
3. $F_i \subseteq S - L$
4. $f_+(x, 0^{p(|x|)}) - F_i \neq \emptyset$

Proof of the claim. We prove the claim by induction over i. Let $i = 0$. Since M does not accept on its rightmost path, it follows that $F_0 \cap L = \emptyset$ and hence $F_0 = f_+(x, 1^{p(|x|)}) \subseteq S - L$. Moreover, $z_0 = 1^{p(|x|)} > z_{\text{right}}$. As $x \in L$, it follows that $f_+(x, 0^{p(|x|)}) \cap L \neq \emptyset$. Hence $f_+(x, 0^{p(|x|)}) - F_0 \neq \emptyset$.

Assume the claim does hold for an $i \in \{0, \ldots, q(|x|) - 1\}$. So z_{right} was not found during the first i iterations of the while loop.

Observe that since M accepts on path z_{right}, it holds for all $z' \in \{0,1\}^{p(|x|)}$ that

- $z' \leq z_{\text{right}} \Rightarrow f(x,z') \cap L \neq \emptyset \Rightarrow f_+(x,z') \cap L \neq \emptyset$ and
- $z' > z_{\text{right}} \Rightarrow f(x,z') \cap L = \emptyset \Rightarrow f_+(x,z') \cap L = \emptyset$.

Since $i < q(|x|)$, the algorithm proceeds with the i+1-th iteration. By the induction hypothesis, it holds that $f_+(x, 0^{p(|x|)}) - F_i \neq \emptyset$, so the condition in line 5 is not satisfied, hence the while-loop is not left prematurely.

N then determines z_{i+1} such that $z_{i+1} < z_i$, $f_+(x, z_{i+1}) - F_i \neq \emptyset$, and $f_+(x, z_{i+1}+1) - F_i = \emptyset$. Clearly, such a z_{i+1} must exist since $f_+(x, z_{\text{right}})$ (which is on the left of z_i) contains at least one element from L which cannot have been culled before because $F_i \subseteq S - L$ by the induction hypothesis. The same holds true for all $f_+(x, z')$ where $z' < z_{\text{right}}$. If $z_{i+1} = z_{\text{right}}$, this means that the algorithm has found z_{right} in the $i+1$-th iteration of the while loop. In this case, we are done.

So let us assume for the sake of contradiction that $z_{i+1} < z_{\text{right}}$. Then $f_+(x, z_{i+1}+1) \cap L \neq \emptyset$. This is a contradiction because $f_+(x, z_{i+1} + 1) \subseteq F_i \subseteq S - L$. For this reason, z_{i+1} cannot be the path chosen in the i+1-th iteration. It follows that $z_{i+1} > z_{\text{right}}$. This implies $f_+(x, z_{i+1}) \subseteq S - L$.

Recall that $f_+(x, z_{i+1}) - F_i \neq \emptyset$. This means that $f_+(x, z_{i+1})$ contains an element from $S - L$ that has not been culled before, i.e., an element which is not in F_i. It follows that $\#F_{i+1} \geq \#F_i + 1 \geq i+1$. Finally, $f_+(x, z_{i+1}) \cap L = \emptyset$ implies that $f_+(x, 0^{p(|x|)}) - F_{i+1} \neq \emptyset$. This proves the claim. ◇

By Claim 3.12 either z_{right} is found during the first $q(|x|)$ iterations of the while loop or $F_{q(|x|)}$ contains at least $q(|x|)$ elements from $S - L$.

Together with $q(|x|) > \sum_{i=0}^{|f(x,y)|} \text{census}_S(i)$, we obtain that $S - L$ cannot contain this many elements. We conclude that $z_{\text{right}}$is found during the first $q(|x|)$ iterations of the while loop. This proves the theorem. □

Contrary to Section 3.2.1 we do not know how dtt-reductions react towards false negatives. For that reason, we cannot deduce that dtt-complete sets are weakly deterministically self-correctable. However, we can provide evidence that the question is indeed difficult. We explain that it is related to the longstanding open question [HOW92] of whether the existence of sparse dtt-complete sets implies P = NP.

Corollary 3.13 *If dtt-complete sets for* NP *are weakly deterministically self-correctable, then the existence of sparse dtt-complete sets for* NP *implies* P = NP.

Proof. We assume that dtt-complete sets for NP are weakly deterministically self-correctable and that there exists a sparse set L such that L is dtt-complete for NP. Since L is weakly deterministically self-correctable, it follows from Theorem 3.2 that for all sparse sets S, $L \leq_{\text{T}}^{\text{p}} L \triangle S$. It follows that $L \leq_{\text{T}}^{\text{p}} L \triangle L$ and hence $L \leq_{\text{T}}^{\text{p}} \emptyset$. This implies P = NP. □

For the restricted case of 2dtt-reducibility however, we can characterise what happens when the reduction encounters false negatives. We show that 2dtt-hard sets for NP stay dtt-hard when we subtract arbitrary sparse information.

Theorem 3.14 *If L is $\leq^{\mathrm{p}}_{\text{2-dtt}}$-hard for* NP *and S is a sparse set such that $L \not\subseteq S$, then $L-S$ is $\leq^{\mathrm{p}}_{\text{dtt}}$-hard for* NP*.*

Proof. Let $L \subseteq \Sigma^*$ and $S \subset \Sigma^*$ be as above, and let M be a nondeterministic Turing machine whose running time on input x is bounded by polynomial p. Without loss of generality, we assume that on input x, M develops precisely $2^{p(|x|)}$ nondeterministic computation paths. Each path can hence be identified by a word $z \in \{0,1\}^{p(|x|)}$. For a path $z \in \{0,1\}^{p(|x|)}, z \neq 0^{p(|x|)}$, we denote the path on the left of z with $z-1$.

Let A be the language accepted by M. We will show that $A \leq^{\mathrm{p}}_{\text{dtt}} L - S$. The left-set of A is defined as

$$\mathrm{Left}(A) =_{\mathrm{def}} \{(x,y) \,\big|\, \text{there exists a } z \geq y \text{ such that } M \text{ accepts } x \text{ along } z\}.$$

From $A \in \mathrm{NP}$ it follows that $\mathrm{Left}(A) \in \mathrm{NP}$. Since L is $\leq^{\mathrm{p}}_{\text{2-dtt}}$-hard for NP, there exists a function f such that $\mathrm{Left}(A) \leq^{\mathrm{p}}_{\text{2-dtt}} L$ via $f : \Sigma^* \to \Sigma^* \times \Sigma^*$, $f \in \mathrm{FP}$. So $f(x,y)$ is a tuple consisting of two elements from Σ^*. For simplicity, we denote these elements with $f_1(x,y)$ and $f_2(x,y)$, respectively.

By the definition of $\leq^{\mathrm{p}}_{\text{2-dtt}}$ it holds that

$$(x,y) \in \mathrm{Left}(A) \Leftrightarrow (f_1(x,y) \in L) \vee (f_2(x,y) \in L).$$

Let q be a polynomial such that for all $x \in \Sigma^*$ and for all $y \in \{0,1\}^{p(|x|)}$ it holds that

$$q(|x|) > \sum_{i=0}^{|f(x,y)|} \mathrm{census}_S(i)$$

We now describe a procedure `collect(x)` which runs in polynomial time and outputs a set of words. We will show that the following holds for all x:

$$\begin{aligned} x \in A &\Leftrightarrow \exists y \in \{0,1\}^{p(|x|)} \big((x,y) \in \mathrm{Left}(A)\big) \\ &\Leftrightarrow \texttt{collect(x)} \text{ outputs at least one word from } L - S \end{aligned}$$

Without loss of generality, we assume that M does neither accept on its first computation path nor on its last path. Furthermore, let $y_1 \in L - S$.

```
 1. z := 0^{p(|x|)}      //current position in tree
 2. F := {f(x,z)}      //set of culled tuples
 3. F_lastrun := F      //elements culled in last run of inner while loop
 4. G := ∅      //set of pivot elements
 5. while j ≤ q(|x|) do
 6.     i := 0
 7.     while i ≤ q(|x|)^2 do
 8.        if f(x,1^{p(|x|)}) ∈ F then goto 26
 9.        determine z' ∈ {0,1}^{p(|x|)}, z' > z where f(x,z') ∉ F ∧ f(x,z'-1) ∈ F
10.        if M accepts along z'-1 then output {y_1}, goto 27
11.        F := F ∪ {f(x,z')}      //store new tuple
12.        F_lastrun := F_lastrun ∪ {f(x,z')}
13.        z := z'
14.        i := i + 1
15.     end while
16.     determine c ∈ Σ*−G such that #{c' | (c,c') ∈ F_lastrun ∨ (c',c) ∈ F_lastrun}
        is maximal      //compute new pivot element
17.     G := G ∪ {c}      // store new pivot element
18.     if {f_1(x,1^{p(|x|)}), f_2(x,1^{p(|x|)})} ∩ G ≠ ∅ then goto 26
19.     determine z'' ∈ {0,1}^{p(|x|)}, z'' > z such that {f_1(x,z''), f_2(x,z'')} ∩ G = ∅
        and {f_1(x,z''−1), f_2(x,z''−1)} ∩ G ≠ ∅
20.     if M accepts along z''-1 then output {y_1}, goto 27
21.     F := F ∪ {f(x,z'')}
             // If G ⊆ L, then f(x,z'') contains a word from L.
22.     z := z''
23.     j := j + 1
24.     F_lastrun := ∅
25. end while
26. output {a | ∃b ∈ Σ*((a,b) ∈ F ∨ (b,a) ∈ F)}
27. end procedure
```

Algorithm 3.3: collect(x), the 2dtt-algorithm

Claim 3.15 *The algorithm* `collect(x)` *runs in polynomial time.*

Proof of the claim. As the number of passes of the outer and inner while loops is bounded by a polynomial, it remains to be argued that steps 9, 16, and 19 can be performed in polynomial time. Steps 9 and 19 are easy binary searches: Start with $z_1 := 0^{p(|x|)}$ and $z_2 := 1^{p(|x|)}$. Let z' be the middle element between z_1 and z_2. If $f(x, z') \in F$ then $z_1 := z'$ else $z_2 := z'$. Choose the middle element between z_1 and z_2 and repeat the above steps until a suitable path is found. The explanation for step 19 is analogous. Step 16 searches for the element occurring most often among the tuples collected in the last round. This can clearly be done in polynomial time. Consequently, the algorithm `collect(x)` runs in polynomial time. ◇

We now argue that the algorithm is correct, i.e., it outputs a word from $L - S$ if and only if $x \in A$.

The only-if part is straightforward: Let us assume $x \notin A$. This means that M on input x does not produce an accepting path, so the if-conditions in lines 10 and 20 are never satisfied. Clearly, $x \notin A$ implies $(x, z) \notin \text{Left}(A)$ for all $z \in \{0,1\}^{p(|x|)}$. Consequently, $\{f_1(x, z), f_2(x, z)\} \cap L = \emptyset$ for all $z \in \{0,1\}^{p(|x|)}$. So none of the collected tuples contains an element from L. Hence the algorithm only outputs words from $\overline{L}$. This proves the only-if part.

For the if-part, we assume that $x \in A$. So there exists a rightmost, i.e., lexicographically largest path $z_{\text{right}} \in \{0,1\}^{p(|x|)}$ such that M accepts x along z_{right}, and all paths on the right of z_{right} reject.

Consequently, the following holds for all $0^{p(|x|)} \leq z \leq z_{\text{right}}$:

- $(x, z) \in \text{Left(A)}$
- $\{f_1(x, z), f_2(x, z)\} \cap L \neq \emptyset$

The general idea of the algorithm is as follows. As we have assumed $x \in A$, each tuple $f(x, z)$ where z is a path on the left of z_{right} contains an element in L. The algorithm tries to cull more than $q(|x|)$ distinct elements from L. If it successfully does so, we can be sure that one element from $L - S$ is output at the end, since we have chosen $q(|x|)$ sufficiently large such that not all elements culled can be in the sparse set S.

We now explain the algorithm in more detail. In the inner while loop, the algorithm culls tuples $f(x, z)$, starting with $z = 0^{p|x|}$.

It searches the next tuple $f(x, z')$ doing a binary search.

Claim 3.16 *If* $z \leq z_{\text{right}}$, *then* $z' \leq z_{\text{right}}$.

Proof of the claim. Let us assume that the binary search starts on the left of z_{right}, i.e., $z < z_{\text{right}}$. As $f(x,z) \in F$ and $f(x,1^{p(|x|)}) \notin F$, there has to exist a path z' satisfying $f(x,z') \notin F$ and $f(x,z'-1) \in F$. As argued above, a path z' satisfying this condition can be found in polynomial time. Note however, that we cannot guarantee to find one particular z', for example the leftmost z'. We might very well miss paths satisfying the condition during the search. However, we can neglect this, since the condition $f(x,z'-1) \in F$ ensures that we cannot have missed the rightmost accepting path z_{right}: Since $z < z_{\text{right}}$, all tuples culled so far correspond to paths on the left of z (and hence also on the left of z_{right}). Consequently, each tuple in F contains an element from L. Hence, the same holds true for the tuple $f(x,z'-1)$. So $z'-1$ is not the rightmost accepting path. Since the algorithm then checks whether M accepts along $z'-1$, we have ensured that $z' \leq z_{\text{right}}$. If $z = z_{\text{right}}$, then both elements of the tuple $f(x,z+1)$ are outside L, and the same holds for all tuples corresponding to paths further right. Hence, the binary search will end with $z' = z+1$, and will therefore discover the accepting path, which then causes the algorithm to terminate. ◇

Observe that the property proven in the last claim is crucial in order to ensure that every tuple collected contains an element from L. Notice that the condition in line 8 cannot be satisfied if $j = 0$, i.e., in the first run of the outer while loop: Since we have assumed that M does not accept along $1^{p(|x|)}$, $f(x,1^{p(|x|)})$ does not contain a word from L. Hence, $f(x,1^{p(|x|)})$ cannot be among the collected tuples since all tuples culled so far correspond to paths on the left of z_{right}. Consequently, if the inner while loop was not left prematurely, i.e., no accepting path of M was found and the condition in line 8 was never satisfied, then we have collected $q(|x|)^2 + 1$-many different tuples after the inner while loop has ended.[1] Note that the condition in line 8 can only be satisfied if a tuple (z_1, z_2) where $z_1, z_2 \notin L$ was added to F in an earlier round. Claim 3.16 then tells us that in this earlier iteration, the binary search must have started on the wrong side of z_{right}.

After the binary search, the algorithm determines the element c occurring most often in the tuples collected in the last execution of the inner while loop.

Using a binary search, the algorithm then searches for a path z'' such that

$$\{f_1(x,z''), f_2(x,z'')\} \cap G = \emptyset \text{ and } \{f_1(x,z''-1), f_2(x,z''-1)\} \cap G \neq \emptyset. \tag{3.3}$$

Claim 3.17 *If $G \subseteq L$, then there exists a z'' satisfying (3.3). Furthermore, if M does not accept x along $z''-1$, then $f(x,z'')$ contains an element from $L - G$.*

Proof of the claim. $G \subseteq L$ implies that all pivot elements culled so far are in L. If M does not accept along $z''-1$, it follows that $z'' \leq z_{\text{right}}$. As z'' satisfies (3.3), it hence follows that $f(x,z'')$ contains an element from $L - G$. ◇

[1] Observe that in the worst case however, we may still have only collected one element from L, because elements can appear in tuples more than once.

As long as no accepting path of M is found and the outer while loop is not left prematurely in line 18, each iteration of the outer while loop adds a new element from L to F. Observe that the same element is never added twice. Claim 3.17 ensures that as long as the pivot element c chosen in line 16 belongs to L, the element added to F in line 21 does also belong to L. Moreover, if the outer while loop is left in line 18 this implies that in an earlier iteration, a pivot element outside L must have been added to G, because we have assumed that M does not accept on $1^{p(|x|)}$, its rightmost computation path. Observe that in case an accepting path of M is found during one of the binary searches, we are done: The algorithm then outputs $y_1 \in L$.

So we from now on assume that the algorithm does not find an accepting path of M.

- **Case 1:** Assume none of the conditions in lines 8 and 18 becomes satisfied while the algorithm is working on input x.

  As argued above, each full run of the inner while loop culls $q(|x|^2)$-many tuples while each iteration of the outer while loop adds a new element c to G.

  For $0 \leq j \leq q(|x|)$, let c_j be the pivot element chosen in the j-th iteration of the outer while loop.

  - **Case a:** $\{c_0, \ldots, c_{q(|x|)}\} \subseteq L$.

    By applying Claim 3.17 inductively, we obtain that after the last iteration of the outer while loop, F contains at least $q(|x|)+1$ many different elements from L. So the algorithm outputs at least one element from $L - S$.

  - **Case b:** $\{c_0, \ldots, c_{q(|x|)}\} \not\subseteq L$.

    Choose the smallest j such that $c_j \notin L$. As c_j was chosen to be the pivot element in the j-th iteration of the outer while loop, c_j was the element occurring most often in $F_{\text{lastrun},j}$, i.e., in the tuples culled in the preceding execution of the inner while loop. Let $m = \#\{c' \mid (c_j, c') \in F_{\text{lastrun},j} \vee (c', c_j) \in F_{\text{lastrun},j}\}$.

    - **Case I:** $m \geq q(|x|)$.

      Let $(c_j, c'_1), \ldots, (c_j, c'_m)$ be the tuples in $F_{\text{lastrun},j}$ which contain c_j. As we have chosen j to be the smallest index where $c_j \notin L$, it follows that $c'_1, \ldots c'_m \in L$. All these elements were added to F. So we obtain that in this case, F contains at least $q(|x|) > \sum_{i=0}^{|f(x,y)|} \text{census}_S(i)$ different elements from L. We conclude that the algorithm outputs at least one element from $L - S$.

    - **Case II:** $m < q(|x|)$.

This implies that no element occurs in more than $q(|x|)-1$ tuples collected in $F_{\text{lastrun},j}$. Each tuple in $F_{\text{lastrun},j}$ contains an element from L (recall that we have chosen j to be minimal). All tuples in $F_{\text{lastrun},j}$ differ in at least one component, and $\#F_{\text{lastrun},j} = q(|x|)^2 + 1$. So it follows that among the tuples in $F_{\text{lastrun},j}$ there have to be at least $q(|x|)$ different elements from L. Again, we can conclude that the algorithm outputs at least one element from $L - S$.

- **Case 2:** One of the conditions in lines 8 and 18 becomes satisfied while the algorithm is working on input x.

  Recall that we have assumed $x \in L$ and that M does not accept along $1^{p(|x|)}$. So $z_{\text{right}} < 1^{p(|x|)}$. Due to Claim 3.16, it is easy to see that the condition in line 8 can only become satisfied if a pivot element outside L was chosen in an iteration of the outer while loop. A similar argument holds for the condition in line 18. If a pivot element outside L was chosen in one of the iterations of the outer while loop, the same reasoning as in case b yields that the algorithm outputs at least one element from $L - S$.

We have shown that the following holds for all x:

$$x \in A \quad \Leftrightarrow \quad \texttt{collect(x)} \text{ outputs at least one word from } L - S$$

This clearly is a dtt-reduction. So $L - S$ is $\leq^{\text{p}}_{\text{dtt}}$-hard for NP. □

We immediately obtain:

Corollary 3.18 *Let L be $\leq^{\text{p}}_{\text{m}}$-hard for* NP *and let S be a sparse set such that $L \not\subseteq S$. Then $L - S$ is $\leq^{\text{p}}_{\text{dtt}}$-hard for* NP.

3.2.3 Conjunctive Truth-Table Reductions

Theorem 3.19 *If L is $\leq^{\text{p}}_{\text{ctt}}$-hard for* NP *and S is sparse, then $L - S$ is $\leq^{\text{p}}_{\text{T}}$-hard for* NP.

Proof. The proof is somewhat similar to the proof of Theorem 3.11. Let $L \subseteq \Sigma^*$ and $S \subset \Sigma^*$ be as above, and let M be a nondeterministic Turing machine whose running time on input x is bounded by polynomial p. Without loss of generality, we assume that on input x, M develops precisely $2^{p(|x|)}$ nondeterministic computation paths. Each path can hence be identified by a word $z \in \{0,1\}^{p(|x|)}$. For a path $z \in \{0,1\}^{p(|x|)}, z \neq 0^{p(|x|)}$, we denote the path on the left of z with $z - 1$.

Let A be the language accepted by M. We will show that $A \leq^{\text{p}}_{\text{T}} L - S$. The left-set of A is defined as

$$\text{Left}(A) =_{\text{def}} \{(x, y) \mid \text{ there exists a } z \geq y \text{ such that } M \text{ accepts } x \text{ along } z\}.$$

From $A \in \mathrm{NP}$ it follows that $\mathrm{Left}(A) \in \mathrm{NP}$. Since L is $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-hard for NP, there exists a function f such that $\mathrm{Left}(A) \leq^{\mathrm{p}}_{\mathrm{ctt}} L$ via $f : \Sigma^* \to \mathcal{P}(\Sigma^*)$, $f \in \mathrm{FP}$. By the definition of $\leq^{\mathrm{p}}_{\mathrm{ctt}}$ it holds that $(x, y) \in \mathrm{Left}(A) \Leftrightarrow f(x, y) \subseteq L$.

Without loss of generality, we assume that M does neither accept on its first computation path nor on its last path.

Furthermore, we define

$$f_-(x, y) \quad =_{\mathrm{def}} \quad f(x, y) \cap \overline{(L - S)}$$

Let q be a polynomial such that for all $x \in \Sigma^*$ and for all $y \in \{0,1\}^{p(|x|)}$ it holds that

$$q(|x|) > \sum_{i=0}^{|f(x,y)|} \mathrm{census}_S(i)$$

We will construct a deterministic polynomial time oracle machine N such that the following holds for all x:

$$\begin{aligned} x \in A &\Leftrightarrow \exists y \in \{0,1\}^{p(|x|)} \big((x, y) \in \mathrm{Left}(A)\big) \\ &\Leftrightarrow \exists y \in \{0,1\}^{p(|x|)} (f(x, y) \subseteq L) \\ &\Leftrightarrow N^{L-S} \text{ accepts } x. \end{aligned}$$

We describe how N works on input $x \in \Sigma^*$. The machine N uses the oracle $L - S$ to compute the set $f_-(x, z)$ for several $z \in \{0,1\}^{p(|x|)}$.

```
 1. i := 0
 2. z_i := 0^{p(|x|)}      //current position in tree, start with leftmost path
 3. F_i := f_-(x, z_i)      //set containing elements outside L − S
 4. while i < q(|x|)
 5.     if F_i ⊇ f_-(x, 1^{p(|x|)}) then reject
 6.     determine z_{i+1} ∈ {0,1}^{p(|x|)}, z_{i+1} > z_i such that (f_-(x, z_{i+1}) − F_i) ≠ ∅
        and ((f_-(x, z_{i+1} − 1) − F_i) = ∅
 7.     if M accepts along z_{i+1} then accept
 8.     F_{i+1} := F_i ∪ f_-(x, z_{i+1})
 9.     i := i + 1
10. end while
11. reject
```

Algorithm 3.4: OPTM N, input x

We show that N is a polynomial time machine: As the number of passes of the while loop is bounded by a polynomial, it suffices to argue that steps 3 and 6 can be performed in polynomial time. Note that N can compute the set $f_-(x,z)$ by querying the oracle $L-S$ for all elements in $f(x,z)$. Step 6 is an easy binary search: Start with $z_1 := 0^{p(|x|)}$ and $z_2 := 1^{p(|x|)}$. Let z' be the middle element between z_1 and z_2. If $\big(f_-(x,z_{i+1}) - F_i\big) = \emptyset$ then $z_1 := z'$ else $z_2 := z'$. Choose the middle element between z_1 and z_2 and repeat the above steps until a suitable path is found. Consequently, we obtain that N runs in polynomial time.

We now argue that the algorithm is correct, i.e., N accepts x if and only if $x \in A$.

For the only-if part, let us assume $x \notin A$. This means that M on input x does not produce an accepting path, so the if-condition in line 7 is never satisfied. So N does not accept x. This proves the only-if part.

Let $x \in A$, so there exists a rightmost accepting path of M on input x, say z_{right}. Hence, for all $z \in \{0,1\}^{p(|x|)}$, it holds that $z \leq z_{\text{right}} \Rightarrow f(x,z) \subseteq L$.

Claim 3.20 *For $0 \leq i < q(|x|)$, if no accepting path of M was found during the first i iterations of the while loop, then it holds that $\#F_i \geq i$. Furthermore, it holds that $f_-(x,1^{p(|x|)}) \not\subseteq F_i$, i.e., the condition in line 5 is not satisfied.*

Proof of the claim. We prove the claim by induction over i.

Let $i = 0$. Then $F_0 = f_-(x,0^{p(|x|)}) \subseteq L$ and $\#F_0 \geq 0$. Since we have assumed that M does not accept along $1^{p(|x|)}$, it follows that $f(x,1^{p(|x|)})$ contains (at least) one element from $\Sigma^* - L$, say y. It follows that $y \in f_-(x,1^{p(|x|)})$, hence $f_-(x,1^{p(|x|)}) \not\subseteq F_0$.

Assume the claim does hold for an $i \in \{0,\ldots,q(|x|)-2\}$. So no accepting path was found during the first i iterations of the while loop and $f_-(x,1^{p(|x|)}) \not\subseteq F_i$. Since $i+1 < q(|x|)$, the while loop is entered for the $i+1$-th time. N then determines z_{i+2} such that $z_{i+2} > z_{i+1}$ and $\big(f_-(x,z_{i+2}) - F_{i+1}\big) \neq \emptyset$ and $\big((f_-(x,z_{i+2}-1) - F_{i+1}\big) = \emptyset$. If an accepting path is found during this binary search, i.e., in the $i+1$-th iteration of the while loop, then we are done. Otherwise, as $\big(f_-(x,z_{i+2}) - F_{i+1}\big) \neq \emptyset$, it holds that $\#F_{i+2} > \#F_{i+1} > \#F_i \geq i$. For the same reason as above, $f_-(x,1^{p(|x|)}) \not\subseteq F_{i+1}$ clearly holds. ◇

From Claim 3.20, it immediately follows that either an accepting path is found during the first $q(|x|)$ iterations of the while loop or $F_{q(|x|)}$ contains at least $q(|x|)$ elements from $\overline{L-S}$. As we have assumed $x \in A$, it follows that $F_{q(|x|)} \subseteq L \cap S$. Together with $q(|x|) > \sum_{i=0}^{|f(x,y)|} \text{census}_S(i)$, we obtain that an accepting path of M is found during the first $q(|x|)$ iterations of the while loop. This proves the theorem. □

3.2.4 Non-Robustness Against Sparse Sets of False Positives

So far we showed that partly corrupt NP-hard sets cannot become to easy because they stay hard with respect to more general reducibilities. Now we consider reductions that are provably too weak to handle corrupt information. Under the assumption P $\neq$ NP we show that many-one reductions, bounded truth-table reductions, and disjunctive truth-table reductions are weak in this sense. More precisely, altering sparse information in SAT can result in sets that are not $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard, not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard, and not $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for NP. On the other hand, Corollary 3.23 shows that similar results for $\leq^{\mathrm{p}}_{\mathrm{ctt}}$, $\leq^{\mathrm{p}}_{\mathrm{tt}}$, and $\leq^{\mathrm{p}}_{\mathrm{T}}$ would imply the existence of NP-complete sets that are not paddable and hence refute the Isomorphism Conjecture (confer Chapter 2, Section 2.2.1). This explains why such results are difficult to prove.

Theorem 3.21 *The following statements are equivalent.*

1. P $\neq$ NP
2. *There exists a sparse* S *such that* SAT $\cup\, S$ *is not* $\leq^{\mathrm{p}}_{\mathrm{btt}}$*-hard for* NP.
3. *There exists a sparse* S *such that* SAT $\cup\, S$ *is not* $\leq^{\mathrm{p}}_{\mathrm{dtt}}$*-hard for* NP.

Proof. **1 $\Rightarrow$ 2:** Assume P $\neq$ NP and let $M_1, M_2, \ldots$ be an enumeration of polynomial-time oracle Turing machines such that M_i runs in time $n^i + i$ and queries at most i strings (so the machines represent all $\leq^{\mathrm{p}}_{\mathrm{btt}}$-reduction functions). We construct an increasing chain of sets $S_1 \subseteq S_2 \subseteq \cdots$ and finally let $S =_{\mathrm{def}} \bigcup_{i\geq 1} S_i$. Let $S_0 =_{\mathrm{def}} \{\varepsilon\}$ and define S_k for $k \geq 1$ as follows:

1. let n be greater than k and greater than the length of the longest word in S_{k-1}
2. let $T =_{\mathrm{def}} (\mathrm{SAT} \cap \Sigma^{\leq n}) \cup S_{k-1} \cup \Sigma^{>n}$
3. choose a word x such that $M_k^T(x)$ accepts if and only if $x \notin$ SAT
4. let Q be the set of words that are queried by $M_k^T(x)$ and that are longer than n
5. let $S_k =_{\mathrm{def}} S_{k-1} \cup Q$

We first observe that the x in step 3 exists: If not, then $L(M_k^T) = \mathrm{SAT}$ and T is cofinite. Hence SAT $\in$ P which is not true by assumption. So the described construction is possible.

If a word w of length j is added to S in step k (i.e., $w \in S_k - S_{k-1}$), then in all further steps, no words of length j are added to S (i.e., for all $i > k$, $S_i \cap \Sigma^j = S_k \cap \Sigma^j$). In the definition of S_k it holds that $|Q| \leq k \leq n$. So in step 5, at most n words are added to S and these words are of length greater than n. Therefore, for all $i \geq 0$, $|S \cap \Sigma^i| \leq i$ and hence S is sparse.

Assume $\mathrm{SAT} \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for NP. So there exists a $k \geq 1$ such that $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{btt}} \mathrm{SAT} \cup S$ via M_k. Consider the construction of S_k and let n, T, x, and Q be the corresponding variables. In all steps $i \geq k$, S will be only changed with respect to words of lengths greater than n. Therefore, $S \cap \Sigma^{\leq n} = S_{k-1}$ and hence

$$\forall w \in \Sigma^{\leq n}, (w \in \mathrm{SAT} \cup S \;\Leftrightarrow\; w \in T). \tag{3.4}$$

If q is an oracle query of $M_k^T(x)$ that is longer than n, then $q \in Q$ and hence $q \in S_k \subseteq S$. So $q \in \mathrm{SAT} \cup S$ and $q \in T$. Together with (3.4) this shows that the computations $M_k^T(x)$ and $M_k^{\mathrm{SAT} \cup S}(x)$ are equivalent. From step 3 it follows that $M_k^{\mathrm{SAT} \cup S}(x)$ accepts if and only if $x \notin \mathrm{SAT}$. This contradicts the assumption that M_k reduces SAT to $\mathrm{SAT} \cup S$. Hence $\mathrm{SAT} \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for NP.

2 ⇒ 1: If P = NP, then for all sparse S, $\mathrm{SAT} \cup S$ is trivially $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete for NP.

1 ⇔ 3: Analogous to the equivalence of 1 and 2; we only sketch the differences. We use an enumeration of $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-reduction machines (i.e., machines that nonadaptively query an arbitrary number of strings and that accept if at least one query is answered positively). Moreover, we change the definition of S_k in step 5 such that

$$S_k =_{\mathrm{def}} \begin{cases} S_{k-1} & : \text{ if } Q = \emptyset \\ S_{k-1} \cup \{q\} & : \text{ if } Q \neq \emptyset, \text{ where } q = \max(Q). \end{cases}$$

Observe that this ensures that S is sparse.

Assume $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{dtt}} \mathrm{SAT} \cup S$ via M_k. If no query of $M_k^T(x)$ is longer than n, then $M_k^T(x)$ and $M_k^{\mathrm{SAT} \cup S}(x)$ are equivalent computations and hence $L(M_k^{\mathrm{SAT} \cup S}) \neq \mathrm{SAT}$ by step 3. Otherwise, there exists a query that is longer than n. Let q be the greatest such query and note that $q \in S_k \subseteq S$. This query gets a positive answer in the computation $M_k^T(x)$. So the computation accepts and by step 3, $x \notin \mathrm{SAT}$. In the computation $M_k^{\mathrm{SAT} \cup S}(x)$, the query q also obtains a positive answer and hence the computation accepts. So also in this case, $L(M_k^{\mathrm{SAT} \cup S}) \neq \mathrm{SAT}$. This shows that $\mathrm{SAT} \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for NP. □

This shows that while $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard and $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard sets do not become too easy when false positives are added (as they stay NP-hard with respect to more general reducibilities, confer Section 3.2.1, and Section 3.2.2), they are not robust against sparse sets of false positives. The next result says that the situation is different for paddable hard sets.

Proposition 3.22 *Let L be paddable and let S be sparse.*

1. *If L is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for* NP*, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for* NP*.*
2. *If L is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*.*
3. *If L is $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-hard for* NP*, then $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-hard for* NP*.*

Proof. We start with the first statement. Let M be a polynomial-time oracle Turing machine that witnesses $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{tt}} L$ and let p be a polynomial which is a bound for the running time of M. Without loss of generality we may assume that the words queried by $M(x)$ are pairwise different. Let f be a padding function for L and let q be a polynomial bounding both, the computation time for f and the census of S.

We describe a machine M' with oracle $L \cup S$ on input x: First, M' simulates $M(x)$ until the list of all queries $Q = (q_1, \ldots, q_m)$ is computed. Let $k = q(q(2p(|x|)))$ and let $Q_j = (f(q_1, j), \ldots, f(q_m, j))$ for $j \in [0, 2k]$. M' queries all words in Q_j for $j \in [0, 2k]$. Let A_j be the corresponding vectors of answers. For every $j \in [0, 2k]$, M' continues the simulation of M by answering the queries according to A_j. M' accepts if and only if the majority of the simulations accepts.

We argue that $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{tt}} L \cup S$ via M'. Note that all $f(q_i, j)$ are pairwise different, since f is injective. For sufficiently large x it holds that $|q_i| \leq p(|x|)$ and $|j| \leq p(|x|)$. So $|f(q_i, j)| \leq q(2p(|x|))$ and hence at most k of the words $f(q_i, j)$ belong to S. Therefore, for the majority of all $j \in [0, 2k]$, Q_j does not contain a word from S. From the padding property $f(q_i, j) \in L \Leftrightarrow q_i \in L$ it follows that for the majority of all $j \in [0, 2k]$, A_j equals the vector of answers occurring in the computation $M^L(x)$. Hence the majority of all simulations shows the same acceptance behavior as $M^L(x)$. This shows $\mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{tt}} L \cup S$ via M' and hence $L \cup S$ is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for NP. This shows the first statement.

The remaining statements are shown analogously, where in the $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-case M' accepts if and only if *all* simulations accept. □

With Theorem 3.21 we have explained that $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete, $\leq^{\mathrm{p}}_{\mathrm{btt}}$-complete, and $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets are not robust against sparse sets of false positives. The following corollary explains the difficulty of showing the same for $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-complete, $\leq^{\mathrm{p}}_{\mathrm{tt}}$-complete, and $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete sets.

Corollary 3.23 *1. If there exists a $\leq^{\mathrm{p}}_{\mathrm{tt}}$-complete set L in* NP *and a sparse S such that $L \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{tt}}$-hard for* NP*, then there exist $\leq^{\mathrm{p}}_{\mathrm{tt}}$-complete sets in* NP *that are not paddable.*

2. If there exists a $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete set L in NP *and a sparse S such that $L \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*, then there exist $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete sets in* NP *that are not paddable.*

3. If there exists a $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-complete set L in NP *and a sparse S such that $L \cup S$ is not $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-hard for* NP*, then there exist $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-complete sets in* NP *that are not paddable.*

3.3 Summary and Outlook

In this chapter we investigated the setting where sparse parts of NP-hard sets can be corrupt. We proved that

- the symmetric difference of m-hard sets and arbitrary sparse sets is always tt-hard. This implies that m-complete sets for NP are *nonadaptively weakly deterministically self-correctable.*
- the union of dtt-hard sets and arbitrary sparse sets is always T-hard.
- the difference of 2dtt-hard sets and arbitrary sparse sets is always dtt-hard.

These results show that $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard and $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard sets do not become too easy when false positives are added (as they stay NP-hard with respect to more general reducibilities). Also, $\leq^{\mathrm{p}}_{\text{2-dtt}}$-hard sets stay hard with respect to a more general reducibility when they encounter a sparse amount of false negatives. On the other hand, we showed that unless $\mathrm{P} = \mathrm{NP}$ there exist sparse sets S_1, S_2 such that $\mathrm{SAT} \cup S_1$ is not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for NP, and $\mathrm{SAT} \cup S_2$ is not $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-hard for NP.

We remark that one can also show that altering sparse information in btt-hard sets for NP results in sets that are still T-hard for NP. Roughly speaking, this can be accomplished by performing a modified Ogihara-Watanabe-like tree-pruning in the computation tree of a given NP-machine.

Theorem 3.24 *[GPT08] If B is $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard for* NP *and S is sparse, then $B \triangle S$ is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for* NP*.*

This implies that btt-complete sets for NP are weakly deterministically self-correctable.

Moreover, this result is related to the notion of *p-closeness*: Yesha [Yes83] defined two sets A and B to be close if the census of their symmetric difference, $A \triangle B$, is a slowly increasing function. Accordingly, A and B are p-close if the census of $A \triangle B$ is polynomially bounded. A is p-close to a complexity class $\mathcal{C}$, if there exists some $B \in \mathcal{C}$ such that A and B are p-close. Yesha [Yes83] poses the question of whether $\leq^{\mathrm{p}}_{\mathrm{m}}$- or $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard sets for NP can be p-close to P (assuming $\mathrm{P} \neq \mathrm{NP}$). Schöning [Sch86] showed that no $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard set for NP is p-close to P, unless $\mathrm{PH} = \Delta^{\mathrm{P}}_2$. Ogiwara [Ogi91] and Fu [Fu93] proved that no $\leq^{\mathrm{p}}_{\mathrm{m}}$-hard set for NP is p-close to P, unless $\mathrm{P} = \mathrm{NP}$. As a direct consequence of btt-complete sets for NP being weakly deterministically self-correctable, this can be strengthend as follows.

Corollary 3.25 *No $\leq^{\mathrm{p}}_{\mathrm{btt}}$-hard set for* NP *is p-close to* P*, unless* $\mathrm{P} = \mathrm{NP}$*.*

Chapter 4

Partitioning NP-Sets

In the last two chapters, we investigated questions concerning the complexity of unions of sets. Given two sets A and B with certain properties, we asked how complex the union $A \cup B$ is. In some respect, this chapter focuses on the inverse question.

Given *one* set A, can this set be partitioned into two sets each satisfying certain properties?

Clearly, the difficulty of this question depends heavily on the properties we require the partition to satisfy. If we for instance do not limit the computation time available for partitioning the set, then any recursive set can be partitioned into two parts of roughly the same size. The question is more difficult however when we demand that the partitioning has to be performed by a polynomial-time machine. Moreover, one can demand a partitioning where the single components and the original set are similar with respect to quality rather than quantity: Given a set A, is there a partitioning of A into disjoint sets B, C such that A, B, and C are all equivalent with respect to some polynomial-time reducibility?

This question is captured by the notion of *mitoticity*, a term which originates from *Mitosis* in biology, the process by which a cell duplicates its DNA in order to generate two identical daughter cells.

In computer science, this notion was originally introduced by Lachlan [Lac67] for recursively enumerable sets. Ambos-Spies [Amb84] then transferred this notion to the polynomial-time setting. A set A is *m-mitotic* if there is a set $S \in \mathrm{P}$ such that A, $A \cap S$, and $A \cap \overline{S}$ are all polynomial-time m-equivalent. Informally, if a set is m-mitotic, it can be split into two parts that both have the same complexity as the original set. T-mitoticity is defined analogously.

Two important open questions concering mitoticity have just recently been solved. First, Glasser et al. [GOP$^+$05] proved that all NP-complete sets are m-autoreducible. Autore-

ducibility is a notion which also originated from recursion theory. Trakthenbrot [Tra70] introduced the notion in recursion theory, and Ambos-Spies [Amb84] transferred it to the polynomial-time setting. A set A is autoreducible if it can be reduced to itself by a Turing machine with oracle A that does not ask its own input to the oracle. In a breakthrough result, Glasser et al. [GPSZ05] showed that m-mitoticity coincides with m-autoreducibility. Together with [GOP$^+$05], it follows that all m-complete sets for NP are m-mitotic.

Hence, it is a natural question to ask whether there exist *non-mitotic* sets in NP, i.e., sets that cannot be split. In this chapter we study the question of the existence of non-mitotic sets in NP. This is a nontrivial question, because there are no natural examples of non-mitotic sets. Natural NP-complete sets are all paddable. It is easy to see that all paddable sets are T-mitotic.

As explained above, it is known [GPSZ06] that all NP-complete sets are m-mitotic (and therefore T-mitotic). Also, nontrivial sets belonging to the class P are T-mitotic. So any unconditional proof of the existence of non-mitotic sets in NP would prove at the same time that P $\neq$ NP. This implies that we cannot expect unconditional results in this area. In contrast to that, Buhrman, Hoene, and Torenvliet [BHT98] unconditionally showed that EXP contains non-m-mitotic sets.

4.1 Separation of Mitoticity Notions

In this section we show that under a reasonable assumption on exponential-time classes, there exist non-m-mitotic sets in NP. However, in showing that the non-m-mitotic set we construct is nevertheless 1-tt mitotic, we in fact prove a stronger result. This relates our search for non-mitotic sets to separations of polynomial-time reducibilities within NP.

Ladner, Lynch, and Selman [LLS75] and Homer [Hom90, Hom97] ask for reasonable assumptions that imply separations of polynomial-time reducibilities within NP. In this section we present such an assumption and demonstrate that it allows a separation of mitoticity notions within NP. This implies a separation of the reducibilities $\leq^{\mathrm{p}}_{\mathrm{m}}$ and $\leq^{\mathrm{p}}_{1-\mathrm{tt}}$ within NP.

Theorem 4.1 *If* EEE $\neq$ NEEE $\cap$ coNEEE*, then there exists an* $L \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$ *that is 1-tt-mitotic but not m-mitotic.*

Proof. Choose $B \in (\mathrm{NEEE} \cap \mathrm{coNEEE}) - \mathrm{EEE}$. So there exists a constant $c \geq 1$ such that B and $\overline{B}$ are decidable in nondeterministic time $2^{2^{2^{c \cdot n}}}$.

We now identify Σ^* with the natural numbers: Let $t : \mathbb{N} \to \mathbb{N}$,

$$t(x) =_{\text{def}} 2^{2^{x^{2c}}}$$

be a tower function and let

$$\begin{aligned} A &=_{\text{def}} \{0^{t(n)} \,|\, n \geq 0\}, \\ C &=_{\text{def}} \{0^{t(x)} \,|\, x \in B\}. \end{aligned}$$

Note that $A \in \text{P}$.

Claim 4.2 *$C \in (\text{NP} \cap \text{coNP}) - \text{P}$.*

Proof of the claim. A membership test for C has to decide $x \in B$ on input $y = 0^{2^{2^{x^{2c}}}}$. The test $x \in B$ can be carried out in nondeterministic time

$$2^{2^{2^{c \cdot |x|}}} \leq 2^{2^{2^{c \cdot 2 \cdot \log x}}} = 2^{2^{x^{2c}}} = |y|.$$

Therefore, $C \in \text{NP}$ and analogously $C \in \text{coNP}$, since $B \in \text{coNEEE}$.

Assume $C \in \text{P}$. Then B can be decided as follows: On input x we construct the string $y = 0^{2^{2^{x^{2c}}}}$ and simulate the deterministic polynomial-time decision procedure for C.

Clearly, this algorithm decides C.

$$|y| = 2^{2^{x^{2c}}} \leq 2^{2^{(2^{|x|})^{2c}}} = 2^{2^{2^{(2c|x|)}}}$$

So the described algorithm has a running time that is polynomial in $2^{2^{2^{(2c|x|)}}}$. This shows $B \in \text{EEE}$ which contradicts the choice of B. Therefore, $C \notin \text{P}$ which proves Claim 4.2. $\diamond$

We define the language that we show to be 1-tt-mitotic, but not m-mitotic.

$$L = C \cup 0(\overline{C} \cap A)$$

Note that the union above is disjoint, since C consists of strings of length $t(n)$ while $0(\overline{C} \cap A)$ consists of strings of length $t(n) + 1$. Observe that $L \in (\text{NP} \cap \text{coNP}) - \text{P}$.

Claim 4.3 *L is 1-tt-mitotic.*

Proof of the claim. The separator is $S = A$. First, we describe the 1-tt-reduction from L to $L \cap S$ on input x: If $x \notin A \cup 0A$, then reject. If $x \in A$, then accept if and only if $x \in L \cap S$. Otherwise, accept if and only if $y \notin L \cap S$ where $x = 0y$. Second, we describe the 1-tt-reduction from $L \cap S$ to $L \cap \overline{S}$ on input x: If $x \notin S$, then reject. Otherwise, accept if and only if $0x \notin L \cap \overline{S}$. Finally, we describe the 1-tt-reduction from $L \cap \overline{S}$ to L on input x: If $x \in S$, then reject. Otherwise, accept if and only if $x \in L$. This shows that L is 1-tt-mitotic. $\diamond$

Claim 4.4 *L is not m-mitotic.*

Proof of the claim. Assume L is m-mitotic. Hence L is m-autoreducible [Amb84], i.e., $L \leq_{\mathrm{m}}^{\mathrm{p}} L$ via a reduction such that $f(x) \neq x$. Let p be a polynomial bounding the computation time of f. Choose the smallest number k such that for all $n \geq k$ it holds that $p(t(n)+1) < t(n+1)$. This choice is possible because

$$p(t(n)+1) \leq t(n)^d = \left(2^{2^{n^{2c}}}\right)^d = 2^{d \cdot 2^{n^{2c}}} \leq 2^{2^{d+n^{2c}}} < 2^{2^{n+n^{2c}}} \leq 2^{2^{(n+1)^{2c}}}$$

for a suitable constant $d \geq 1$. Define the finite set

$$L' =_{\mathrm{def}} \{w \mid |w| \leq t(k)+1 \text{ and } w \in L\}.$$

The following algorithm decides in polynomial time whether the input z belongs to L.

```
1. x := z
2. if |x| ≤ t(k) + 1 then accept if and only if x ∈ L'
3. if |f(x)| ≥ |x| then reject
4. x := f(x), goto 2
```

The algorithm runs in polynomial time, since each iteration decreases the length of x. Also, since f is an m-autoreduction, at any time it holds that

$$z \in L \Leftrightarrow x \in L. \tag{4.1}$$

So if we stop in line 2, then we accept if and only if $z \in L$. It remains to be argued for a stop in line 3.

Assume $z \in L$ but we reject in line 3; we will derive a contradiction. By (4.1), at the moment we reject, it holds that

$$x \in L \text{ and } |x| \geq t(k)+1 \tag{4.2}$$

In particular, $x \in A \cup 0A$, i.e., $x = 0^{t(n)}$ or $x = 0^{t(n)+1}$ for a suitable n. By definition of L,

$$0^{t(n)} \in L \Leftrightarrow 0^{t(n)+1} \notin L.$$

It follows that $f(x) \neq 0^{t(n)}$ and $f(x) \neq 0^{t(n)+1}$, since otherwise either $f(x) = x$ or $(0^{t(n)} \in L \Leftrightarrow 0^{t(n)+1} \in L)$. Note that $n \geq k$, since otherwise $|x| \leq t(n)+1 < t(k)+1$ which contradicts (4.2). Therefore, by the choice of k,

$$|f(x)| \leq p(|x|) \leq p(t(n)+1) < t(n+1).$$

However, apart from x there are no words in L that have a length in $[t(n), t(n+1)-1]$. It follows that $|f(x)| < |x|$, since $f(x)$ must belong to L. This contradicts our assumption that we reject in line 3. Therefore, if we stop in line 3, then $z \notin L$. So the algorithm above decides L in polynomial time. This is a contradiction. Therefore, L is not m-mitotic. $\diamond$

This proves the theorem. $\square$

Selman [Sel82] showed under the hypothesis $\mathrm{E} \neq \mathrm{NE} \cap \mathrm{coNE}$ that there exist $A, B \in \mathrm{NP} - \mathrm{P}$ such that A tt-reduces to B but A does not positive-tt-reduce to B. The separation of mitoticity notions given in the last theorem allows us to prove a similar statement:

Corollary 4.5 *If* $\mathrm{EEE} \neq \mathrm{NEEE} \cap \mathrm{coNEEE}$*, then there exist* $A, B \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$ *such that* $A \leq^{\mathrm{p}}_{1-\mathrm{tt}} B$*, but* $A \not\leq^{\mathrm{p}}_{\mathrm{m}} B$*.*

Proof. Take the set L from Theorem 4.1 and let $S \in \mathrm{P}$ be a separator that witnesses L's 1-tt-mitoticity, i.e., L, $L \cap S$, and $L \cap \overline{S}$ are pairwise 1-tt-equivalent. These sets cannot be pairwise m-equivalent, since otherwise L would be m-mitotic. This gives us the sets A and B. $\square$

However, a weaker assumption already separates 1-tt-reducibility from m-reducibility within NP.

Theorem 4.6 *If* $\mathrm{E} \neq \mathrm{NE} \cap \mathrm{coNE}$*, then there exist* $A, B \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$ *such that* $A \leq^{\mathrm{p}}_{1-\mathrm{tt}} B$*, but* $A \not\leq^{\mathrm{p}}_{\mathrm{m}} B$*.*

Proof. If $\mathrm{E} \neq \mathrm{NE} \cap \mathrm{coNE}$, then there exists a tally set $T \in \mathrm{NP} \cap \mathrm{coNP} - \mathrm{P}$ and there exists a p-selective set A such that $A \equiv^{\mathrm{p}}_{\mathrm{T}} T$ [Sel79]. Trivially, $A \leq^{\mathrm{p}}_{1-\mathrm{tt}} \overline{A}$, and since A is p-selective, and not in P, A is not m-reducible to $\overline{A}$. $\square$

4.2 Non-T-Mitotic Sets in NP

Buhrman, Hoene, and Torenvliet [BHT98] showed that EXP contains non-m-mitotic sets, but left open whether EXP contains non-T-mitotic sets. We are interested in constructing non-T-mitotic sets in NP. Recall that the existence of non-T-mitotic sets in NP would imply that $\mathrm{P} \neq \mathrm{NP}$, hence we cannot expect to prove their existence without a sufficiently strong hypothesis. Moreover, the same holds for the non-existence of non-m-mitotic sets in NP. Since it is known [BHT98] that EXP contains non-m-mitotic sets, this would imply that $\mathrm{NP} \neq \mathrm{EXP}$.

It is well known that mitoticity implies autoreducibility [Amb84], hence it suffices to construct non-T-autoreducible sets in NP.

Beigel and Feigenbaum [BF92] construct *incoherent sets* in NP under the assumption that NEEEXP $\not\subseteq$ BPEEEXP, the triple-exponential time variant of BPP. *Coherent sets* [Yao90] are sets that are autoreducible via probabilistic polynomial-time oracle Turing machines. In particular, incoherent sets are non-T-autoreducible.

With the next theorem, we show that there are non-T-autoreducible sets in NP under the weaker assumption that NEEE $\not\subseteq$ EEE. Observe that these sets are not necessarily incoherent.

We apply a similar method as in the proof of Theorem 4.1.

Theorem 4.7 *If* EEE $\neq$ NEEE*, then there exists* $C \in \mathrm{NP}-\mathrm{P}$ *such that* C *is not T-autoreducible.*

Proof. Choose $B \in \mathrm{NEEE} - \mathrm{EEE}$. So there exists a constant $c \geq 1$ such that B is decidable in nondeterministic time $2^{2^{2^{c \cdot n}}}$.

We now identify Σ^* with the natural numbers: Let $t : \mathbb{N} \to \mathbb{N}$,

$$t(x) =_{\mathrm{def}} 2^{2^{x^{2c}}}$$

be a tower function and let

$$\begin{aligned} A &=_{\mathrm{def}} \{0^{t(n)} \,|\, n \geq 0\}, \\ C &=_{\mathrm{def}} \{0^{t(x)} \,|\, x \in B\}. \end{aligned}$$

Note that $A \in \mathrm{P}$.

Claim 4.8 $C \in \mathrm{NP} - \mathrm{P}$.

Proof of the claim. Analogous to the proof of Claim 4.2 in Theorem 4.1. $\diamond$

We will now show that the set C is not T-autoreducible.

Let us assume that C is T-autoreducible. So there exists a deterministic polynomial time oracle Turing machine M' such that $L(M'^C) = C$. Furthermore, it holds for all x that during its work on input x, M' never queries the oracle C for x.

Let $k \geq 0$ such that the running-time of M' on inputs of length $n \geq 1$ is bounded by the polynomial n^k.

Observe that $t(n)^k <_{\mathrm{ae}} t(n+1)$. More precisely,

$$\big(n > \log(k) - 1\big) \implies t(n)^k = (2^{2^{n^{2c}}})^k < t(n+1) = 2^{2^{(n+1)^{2c}}}. \tag{4.3}$$

Let $\log(k) \leq m$, and assume that M' is running on input $0^{t(m)}$. Since M' is an oracle machine, it can query C for a string q. Observe that such a query q can have length at

most $t(m)^k$. We can assume that M' queries C only for strings from A (i.e. strings of the form $0^{t(i)}$ for $i \geq 0$). As $C \subseteq A$, these are the only queries that have a chance of getting a positive answer from C. Notice that M' is not allowed to query C for $0^{t(m)}$ because M' proves that C is T-autoreducible. Furthermore, due to (4.3), M' on input $0^{t(m)}$ cannot query C for $0^{t(m+1)}$ or longer strings. So M' on input $0^{t(m)}$ can only query C for strings in $\{0^{t(i)} \mid 0 \leq i < m\}$.

We construct a deterministic polynomial-time Turing machine M such that $L(M) = C$. On input x, M first checks whether $x \in A$, i.e., whether $x = 0^{t(n)}$ for some $n \geq 0$. If no such n exists, M rejects. Since this can easily be done in polynomial time, we assume that there exists an $n \geq 0$ such that M is running on input $0^{t(n)}$.

We define

$$E[i] = \begin{cases} 1, & \text{if } 0^{t(i)} \in C, \\ 0, & \text{if } 0^{t(i)} \notin C. \end{cases}$$

M will compute $E[0], E[1], \ldots, E[n]$ one after another and accept the input $0^{t(n)}$ if and only if $E[n] = 1$.

Since k is a constant, we can encode $E[0], E[1], \ldots, E[\log(k) - 1]$ into the program of M.

During its work on input $0^{t(n)}$, M will simulate M'. Notice that while M' is equipped with oracle C, M is not an oracle machine and hence cannot query an oracle while simulating M'. Instead, M will make use of the values $E[0], E[1], \ldots$ it has computed so far to answer possible oracle queries of M'.

Let $\log(k) \leq m \leq n$. We now describe how M computes $E[m]$ if it has access to $E[0], E[1], \ldots, E[m-1]$.

```
1. Compute 0^t(m).
2. Simulate M' on input 0^t(m).
   For every oracle query q of M' on input 0^t(m), proceed as follows:
   (a) Compute j ≥ 0 such that q = 0^t(j).          //Note that j < m.
   (b) If E[j] = 0, continue the simulation of M' with a negative
       answer to query q.
       If E[j] = 1, continue the simulation of M' with a positive
       answer to query q.
3. If M' accepts, set E[m] := 1, else set E[m] := 0.
```

Algorithm 4.1: Subroutine compute_E[m]

From our argumentation above it follows that for $0 \leq i \leq n$, the algorithm computes $E[i]$ correctly if it has access to $E[0], \ldots, E[i-1]$. Since M is running on input $0^{t(n)}$ and computes $E[0], E[1], \ldots, E[n]$ one after another, M clearly is a polynomial time machine and it holds that $L(M) = C$.

This proves $C \in \mathrm{P}$, which contradicts our assumption. Hence, such a machine M' cannot exist. So C is not T-autoreducible. □

Corollary 4.9 *If* $\mathrm{EEE} \neq \mathrm{NEEE}$*, then there exists* $C \in \mathrm{NP} - \mathrm{P}$ *such that* C *is not T-mitotic.*

Proof. T-mitoticity implies T-autoreducibility [Amb84]. Consequently, the set C in Theorem 4.7 cannot be T-mitotic since it is not T-autoreducible. □

With respect to Theorem 4.7, Buhrmann and Torenvliet [BT96] cite Beigel et al. [BBFG91] for a similar result. Nevertheless, we have presented our proof here because a proof is not given explicitly in [BBFG91].

Moreover, we can reuse Theorem 4.7 to show that under a stronger assumption, there are non-T-autoreducible sets in $(\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$.

Corollary 4.10 *If* $\mathrm{EEE} \neq \mathrm{NEEE} \cap \mathrm{coNEEE}$*, then there exists* $C \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$ *such that* C *is neither T-autoreducible nor T-mitotic.*

Proof. This can easily be seen by using the sets B and C from the proof of Theorem 4.1 in the proof of Theorem 4.7 instead of the ones constructed in the latter. □

4.3 Summary and Outlook

In this chapter we studied the question of the existence of non-mitotic sets in NP. We proved that if $\mathrm{EEE} \neq \mathrm{NEEE} \cap \mathrm{coNEEE}$, then there exists an $L \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$ that is not m-mitotic but in fact is 1-tt-mitotic. From this, it followed that under the same hypothesis, $(\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$ contains a non-mitotic set and that 1-tt-reducibility and m-reducibility differ on sets in NP. On the one hand, this consequence explains the need for a reasonably strong hypothesis. On the other hand we also showed that 1-tt-reducibility and m-reducibility separate within NP under the weaker hypothesis that $\mathrm{E} \neq \mathrm{NE} \cap \mathrm{coNE}$.

We then gave evidence for the existence of non-T-mitotic sets in NP. Unless $\mathrm{EEE} = \mathrm{NEEE}$, there exists a set $C \in \mathrm{NP} - \mathrm{P}$ such that C is not T-mitotic. We will now explain that under a strong hypothesis concerning n-generic sets, one can show that the notions of T-autoreducibility and T-mitoticity already differ within NP.

Ladner [Lad73] showed that T-autoreducibility and T-mitoticity coincide for computably enumerable sets. Under the strong assumption that NP ∩ coNP contains n-generic sets, one can show that the similar question in complexity theory has a negative answer. The notion of resource-bounded genericity was defined by Ambos-Spies, Fleischhack, and Huwig [AFH87]. We use the following equivalent definition [BM95, PS02], where $L(x)$ denotes L's characteristic function on x.

Definition 4.11 *For a set L and a string x let $L|x = \{y \in L \mid y < x\}$. A deterministic oracle Turing machine M is a* predictor *for a set L if for all x, $M^{L|x}(x) = L(x)$. L is* a.e. unpredictable in time $t(n)$*, if every predictor for L requires more than $t(n)$ time for all but finitely many x.*

Definition 4.12 *A set L is $t(n)$-*generic *if it is a.e. unpredictable in time $t(2^n)$.*

This is equivalent to demanding that for every oracle Turing machine M, if $M^{L|x}(x) = L(x)$ for all x, then the running time of M is at least $t(2^{|x|})$ for all but finitely many x.

Under the assumption that NP ∩ coNP contains n-generic sets, it is possible to show that 2-tt autoreducibility and T-mitoticity (and hence r-autoreducibility and r-mitoticity for every reduction r between 2-tt and T) do not coincide for NP.

Theorem 4.13 *[GSTZ07] If* NP ∩ coNP *contains n-generic sets, then there exists a tally set $S \in$* NP ∩ coNP *such that S is 2-tt-autoreducible but not T-mitotic.*

A summary of the results is shown in Table 4.1.

Assumption	Conclusion	Remark
EEE ≠ NEEE	$\exists A \in$ NP that is not T-autoreducible	$A \in$ NP − P
NP ∩ coNP contains n-generic sets	$\exists A \in$ NP that is 2-tt-autoreducible but not T-mitotic	$A \in$ (NP ∩ coNP) − P
EEE ≠ NEEE ∩ coNEEE	$\exists A \in$ NP that is 1-tt-mitotic but not m-mitotic	$A \in$ (NP ∩ coNP) − P
E ≠ NE ∩ coNE	$\exists A, B \in$ NP such that $A \leq^{\mathrm{p}}_{1-\mathrm{tt}} B$ but $A \not\leq^{\mathrm{p}}_{\mathrm{m}} B$	$A, B \in$ (NP ∩ coNP) − P

Table 4.1: Summary of results related to NP.

Part II

Uniform Computation Models

Chapter 5

The Leaf-Language Approach

The preceding chapters dealt with structural properties of complexity classes and their hard problems, mainly of the class NP. The second part of this thesis is also dedicated to complexity classes. We will however take a different perspective and shift our focus to the description of complexity classes and to the relations between complexity classes.

In some sense, after investigating the "interior" of complexity classes, we now look at the exterior and study the "big picture".

We have learnt that many complexity classes are defined by restricting the amount of time or space a deterministic Turing machine may consume when deciding a language, e.g. P, PSPACE, and EXP. Nondeterministic complexity classes like NP and NEXP are defined in a similar way by nondeterministic Turing machines. Contrary to that, complexity classes like UP, 1NP, and BPP are also defined by nondeterministic machines with a polynomial time bound, but with an acceptance behaviour altered in some way.

For instance, a language A is in 1NP if there exists a nondeterministic polynomial-time Turing machine M such that x is in A if and only if M on input x produces precisely one accepting path. So an input is rejected if the machine outputs zero accepting paths or at least two accepting paths. It is not difficult to see that the complements of NP-sets are all in 1NP, i.e., coNP $\subseteq$ 1NP.

The situation is more peculiar with the so-called *promise classes* UP and BPP. Let us have a look at the class UP. Similarly to the class 1NP, a language A is in UP if there exists a nondeterministic polynomial-time Turing machine M such that x is in A if M on input x produces precisely one accepting path and x is not in A if M does not produce an accepting path at all. What makes the class UP in some sense a new and unusual kind of complexity class is that not every polynomially bounded nondeterministic Turing machine can be the basis defining a language in UP. For a machine M to define a UP-language, it must have the property that on all inputs it either accepts by producing

precisely one accepting path among its nondeterministic computation paths, or rejects by producing rejecting paths only. In general, nondeterministic Turing machines do not show this behaviour, so only Turing machines who keep the *promise* of never producing more than one accepting path can be used to define languages in UP.

These were just a few examples: Over the past forty years, a variety of different acceptance notions have been introduced and investigated, some of which are very complicated. The concept of leaf languages provides a *uniform* way to characterise complexity classes. It was introduced by Bovet, Crescenzi, and Silvestri [BCS92] and independently by Vereshchagin [Ver93]. In the last decade, the leaf-language approach has gained a considerable amount of attention. Vollmer's [Vol99] and Wagner's [Wag04a] survey articles provide extensive coverage of the progress made in the field of leaf languages.

In the next section, we will give a formal definition of the leaf-language approach.

5.1 An Introduction to Leaf Languages

Let M be a nondeterministic polynomial-time bounded Turing machine such that every computation path outputs one letter from a fixed alphabet. Let $M(x)$ denote the computation tree of M on input x. Furthermore, let $\beta_M(x)$ be the concatenation of all leaf-symbols of $M(x)$ in the lexicographic ordering of the computation paths, i.e., the leftmost path outputs the first letter of $\beta_M(x)$ while the rightmost path outputs the last letter of $\beta_M(x)$.

A nondeterministic polynomial-time bounded Turing machine M is *balanced* if on all inputs, it produces a balanced computation tree in the following sense: On input x, there exists a path p with length $l(x)$ such that all paths on the left of p have length $l(x)$, and all paths on the right have length $l(x) - 1$. Observe that this is equivalent to demanding that there exists a polynomial-time computable function that on input (x, n) computes the n-th path of $M(x)$.

Definition 5.1 *Let Δ be a finite alphabet, and let $L_1, L_2 \subseteq \Delta^*$, $L_1, L_2 \notin \{\emptyset, \{\varepsilon\}\}$ such that $L_1 \cap L_2 = \emptyset$.*

1. *The* (unbalanced) leaf-language class $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L_1|L_2)_\Delta$ *is defined as follows:*

$$A \in \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L_1|L_2)_\Delta \quad \Leftrightarrow_{\mathrm{def}} \quad \text{there exists a NPTM } M \text{ such that for every } x \in \Sigma^*, \\ x \in A \to \beta_M(x) \in L_1 \text{ and } x \notin A \to \beta_M(x) \in L_2.$$

2. *The* balanced leaf-language class $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1|L_2)_\Delta$ *is defined as follows:*

$$A \in \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1|L_2)_\Delta \quad \Leftrightarrow_{\mathrm{def}} \quad \text{there exists a balanced NPTM } M \text{ such that for every} \\ x \in \Sigma^*, x \in A \to \beta_M(x) \in L_1 \text{ and } x \notin A \to \beta_M(x) \in L_2.$$

For leaf-language classes $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L_1|L_2)_\Delta$ and $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1|L_2)_\Delta$, we will often omit the subscript $_\Delta$ when Δ is the smallest alphabet such that $L_1 \cup L_2 \subseteq \Delta^*$. In these cases, it will be clear from the context what Δ is. If $L_2 = \Delta^* - L_1$, then we will also write $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L_1)$ and $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1)$, respectively.

Definition 5.2 *For any class of languages $\mathcal{C}$, we define*

1. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\mathcal{C}) = \bigcup_{B\in\mathcal{C}} \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(B)$ *and*
2. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(\mathcal{C}) = \bigcup_{B\in\mathcal{C}} \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(B)$.

A complexity class $\mathcal{D}$ is *unbalanced leaf-language definable* if there exists $\mathcal{C}$ such that $\mathcal{D} = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\mathcal{C})$. Analogously define *balanced leaf-language definability.*

Example 5.3 *The following are leaf-language characterisations of several well known complexity classes.*

1. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(1) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(1) = \mathrm{P}$
2. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(0^*1(0\cup 1)^*) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(0^*1(0\cup 1)^*) = \mathrm{NP}$
3. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(0^*)_{\{0,1\}} = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(0^*)_{\{0,1\}} = \mathrm{coNP}$
4. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(0^*10^*) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(0^*10^*) = 1\mathrm{NP}$
5. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(0^*10^*|0^*) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(0^*10^*|0^*) = \mathrm{UP}$
6. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\mathrm{REG}) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(\mathrm{REG}) = \mathrm{PSPACE}$
7. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}((11)^*) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}((0^*10^*1)^*0^*) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}((0^*10^*1)^*0^*) = \oplus\mathrm{P}$
8. $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}((11)^*) = \mathrm{P}$

5.1.1 A Connection between Complexity Theory and Formal Languages

Example 5.3 lists several regular languages and the complexity classes they describe via the leaf-language approach. In fact, there exists a close connection between leaf languages and formal languages. To explain this connection, we recapitulate the notions of starfree languages and the dot-depth hierarchy.

Recall that *starfree regular languages* (starfree languages for short) are regular languages that can be built up from single letters by using Boolean operations and concatenation (so iteration is not allowed), and SF denotes the class of starfree languages.

Brzozowski and Cohen [CB71, Brz76] introduced the *dot-depth hierarchy* (DDH for short) which is a parameterisation of the class of starfree languages. The dot-depth counts the

minimal number of nested alternations between Boolean operations and concatenation that is needed to define a language. The classes of the dot-depth hierarchy consist of languages that have the same upper bound for their dot-depth. For a class of languages $\mathcal{C}$, let $\mathrm{Pol}(\mathcal{C})$ denote $\mathcal{C}$'s closure under finite union and finite concatenation. Let $\mathrm{BC}(\mathcal{C})$ denote the Boolean closure of $\mathcal{C}$.

The classes (or levels) of the dot-depth hierarchy are defined as:

$$\begin{aligned}
\mathcal{B}_0 &=_{\mathrm{def}} \{L \subseteq \Sigma^* \mid \Sigma \text{ is a finite alphabet with at least two letters and } L \\
&\qquad\qquad \text{is a finite union of terms } v\Sigma^*w \text{ where } v, w \in \Sigma^*\} \\
\mathcal{B}_{n+\frac{1}{2}} &=_{\mathrm{def}} \mathrm{Pol}(\mathcal{B}_n) \\
\mathcal{B}_{n+1} &=_{\mathrm{def}} \mathrm{BC}(\mathcal{B}_{n+\frac{1}{2}})
\end{aligned}$$

The dot-depth of a language L is defined as the minimal m such that $L \in \mathcal{B}_m$ where $m = n/2$ for some integer n. Brzozowski [CB71] also formulated the *dot-depth problem*: Given a starfree language L, what is the dot-depth of L? Although partial results (i.e., decidability results for the lower levels) are known, the problem remains unsolved, despite the combined efforts of many researchers.

All levels of the dot-depth hierarchy are closed under union, intersection, taking inverse morphisms, and taking residuals [PP86, Arf91, PW97]. The dot-depth hierarchy is strict [BK78, Tho84] and exhausts the class of starfree languages [Eil76]. The levels of the dot-depth hierarchy provide a fine characterisation of the starfree languages. We now discuss the connection to the polynomial-hierarchy in complexity theory.

The following theorem shows that the dot-depth hierarchy and the polynomial-time hierarchy are closely related.

Theorem 5.4 ([HLS$^+$93, BV98, BKS99]) *The following holds for $n \geq 1$ and relative to all oracles.*

1. $\mathrm{P} = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(\mathcal{B}_0) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\mathcal{B}_0)$
2. $\Sigma^{\mathrm{P}}_{\mathrm{n}} = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(\mathcal{B}_{n-1/2}) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\mathcal{B}_{n-1/2})$
3. $\Pi^{\mathrm{P}}_{\mathrm{n}} = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(\mathrm{co}\mathcal{B}_{n-1/2}) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\mathrm{co}\mathcal{B}_{n-1/2})$
4. $\mathrm{BC}(\Sigma^{\mathrm{P}}_{\mathrm{n}}) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(\mathcal{B}_n) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(\mathcal{B}_n)$

In particular, the attraction of this connection comes from the fact that both the polynomial-time hierarchy and the dot-depth hierarchy are prominent and well-studied objects. Even more, with the P-NP problem and the dot-depth problem, they represent two of the most fundamental problems in theoretical computer science.

5.1.2 Oracle Separations

We will also consider relativised leaf-language classes which are denoted by $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}O}(\mathcal{C})$ and $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(\mathcal{C})$, where the superscript O indicates that the nondeterministic machine can query oracle O. In fact, a further advantage of the leaf-language approach is closely related to relativisation:

In their pioneering work for the leaf-language approach, Bovet, Crescenzi, and Silvestri [BCS92] and Vereshchagin [Ver93] independently introduced the notion of polylog-time reducibility (plt-reducibility for short). This is a reducibility for formal languages. We start with a formal definition.

Definition 5.5 *A function $f : \Sigma^* \to \Sigma^*$ is* polylog-time computable *if there exist two polynomial-time bounded oracle transducers $R : \Sigma^* \times \mathbb{N} \to \Sigma$ and $l : \Sigma^* \to \mathbb{N}$ such that for all x,*

$$f(x) = R^x(|x|, 1)R^x(|x|, 2) \cdots R^x(|x|, l^x(|x|))$$

where R and l access the input x as an oracle.

A language B is polylog-time reducible *(plt-reducible) to a language C, $B \leq_{\mathrm{m}}^{\mathrm{plt}} C$ for short, if there exists a polylog-time computable f such that for all x, it holds that*

$$x \in B \Leftrightarrow f(x) \in C.$$

This reducibility permits an unexpected connection between two seemingly independent questions:

1. Are given complexity classes separable by oracles?
2. Are given languages plt-reducible?

This connection is established by the following theorem:

Theorem 5.6 *[BCS92, Ver93] Suppose for given complexity classes $\mathcal{D}_1$ and $\mathcal{D}_2$, there exist languages L_1 and L_2 such that $\mathcal{D}_1 = \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_1)$ and $\mathcal{D}_2 = \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(L_2)$. Then it holds that*

$$L_1 \leq_{\mathrm{m}}^{\mathrm{plt}} L_2 \quad \Leftrightarrow \quad \forall O\big(\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}O}(L_1) \subseteq \mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}O}(L_2)\big).$$

In the same spirit, leaf languages allow concise oracle constructions. The connection via the BCSV-theorem reduces oracle constructions to their combinatorial core. In particular, we have neither to care about the detailed stagewise construction of the oracle, nor do we have to describe the particular coding of the single stages.

Chapter 6

Unbalanced Leaf-Language Classes

Unbalanced leaf languages form the most natural leaf-language model, as any nondeterministic Turing machine can be used to define a complexity class. Moreover, they sometimes allow simpler characterisations of complexity classes than balanced leaf languages do. For instance, we saw in Example 5.3 that in order to characterise the class $\oplus$P, we need a more complicated leaf language in the balanced case than we do in the unbalanced case. However, despite its simplicity, this approach suffers from one major drawback:

In Chapter 5, we explained the advantages of the famous BCSV-theorem (Theorem 5.6), which states that a language L_1 is plt-reducible to a language L_2 if and only if $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_1)$ is robustly contained in $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_2)$. *Robustly contained* means that the containment holds relative to all oracles.

For this equivalence however, it is crucial that balanced leaf-language classes are used. The theorem does not hold for the unbalanced model: Observe that languages $L, L' \subseteq \{0,1\}^*$ with $L =_{\mathrm{def}} \{w \,|\, |w| \text{ is odd}\}$ and $L' =_{\mathrm{def}} 0\{0,1\}^*$ form a counterexample, since $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) = \oplus\mathrm{P}$ is not robustly contained in $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L') = \mathrm{P}$ although L plt-reduces to L'.

Recently, Wagner [Wag04b] solved this problem by introducing the *polynomial-time tree reducibility* (ptt-reducibility, for short) which is an analogue of plt-reducibility for unbalanced leaf languages. He was able to show that this reduciblity admits a BCSV-theorem for unbalanced leaf-language classes.

Theorem 6.1 *[Wag04b] For nontrivial $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ the following are equivalent:*

1. $L_1 \leq^{\mathrm{ptt}}_{\mathrm{m}} L_2$
2. $\forall O(\mathrm{Leaf}^{\mathrm{p}^O}_{\mathrm{u}}(L_1) \subseteq \mathrm{Leaf}^{\mathrm{p}^O}_{\mathrm{u}}(L_2))$

In this chapter, we will analyse this new reducibility: We prove that restricted to regular languages, the levels 0, 1/2, 1, and 3/2 of the dot-depth hierarchy are closed under ptt-reducibility. We will explain that these results are also interesting in other respects: They indicate that the connection between dot-depth and polynomial-time hierarchy is closer than formerly known. More precisely, we show that on the lower levels, the dot-depth and the polynomial-time hierarchy *perfectly correspond.*

6.1 Perfect Correspondences

We already mentioned that Hertrampf et al. [HLS$^+$93], and Burtschick and Vollmer [BV98] proved that the levels of the polynomial-time hierarchy are connected with the levels of the dot-depth hierarchy.

For instance, Theorem 5.4.2 makes two statements. First, it states that $\mathcal{B}_{n-1/2}$ contains a language L_n such that $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_n) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_n) = \Sigma^{\mathrm{P}}_{\mathrm{n}}$. Second, it states that for $n \geq 1$ and $L \subseteq \Sigma^*$ it holds that

$$L \in \mathcal{B}_{n-1/2} \quad \Rightarrow \quad \forall O(\mathrm{Leaf}^{\mathrm{p}^O}_{\mathrm{b}}(L) \subseteq \Sigma^{\mathrm{P}^O}_{\mathrm{n}}), \tag{6.1}$$

$$L \in \mathcal{B}_{n-1/2} \quad \Rightarrow \quad \forall O(\mathrm{Leaf}^{\mathrm{p}^O}_{\mathrm{u}}(L) \subseteq \Sigma^{\mathrm{P}^O}_{\mathrm{n}}), \tag{6.2}$$

since the proofs do relativise.

However, these results do not answer the following questions: Are the languages in $\mathcal{B}_{n-1/2}$ precisely the starfree languages which characterise $\Sigma^{\mathrm{P}}_{\mathrm{n}}$ via the balanced leaf-language model, or does there exist $L \in \mathrm{SF}$ such that $L \notin \mathcal{B}_{n-1/2}$ but $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L) \subseteq \Sigma^{\mathrm{P}}_{\mathrm{n}}$? What is the situation for the unbalanced leaf-language model? The possible scenarios are depicted in Figure 6.1.

For balanced leaf languages, the (disappointing) answer is already known: The reverse of (6.1) does not hold, even if we demand L to be starfree: For every $n \geq 1$, there exists a starfree regular language $L_n \notin \mathcal{B}_{n-1/2}$ such that L_n plt-reduces to a language in $\mathcal{B}_{1/2}$ [Gla05]. So by Theorem 5.6 it holds that $\forall O(\mathrm{Leaf}^{\mathrm{p}^O}_{\mathrm{b}}(L_n) \subseteq \mathrm{NP}^O)$, but $L_n \notin \mathcal{B}_{n-1/2}$. Thereby, Scenario I (confer Figure 6.1) holds for the connection via balanced leaf languages. This reveals that although balanced-leaf languages do connect DDH and PH, the connection they establish is not perfect, most importantly, it is not a one-one connection between the levels of the hierarchies.

We now draw our attention to unbalanced leaf languages. There several known results indicate that the situation is different: For certain lower levels of the dot-depth hierarchy, much closer connections than those in Equation (6.2) are known. The following theorem summarises results by Borchert, Kuske, Stephan, and Schmitz:

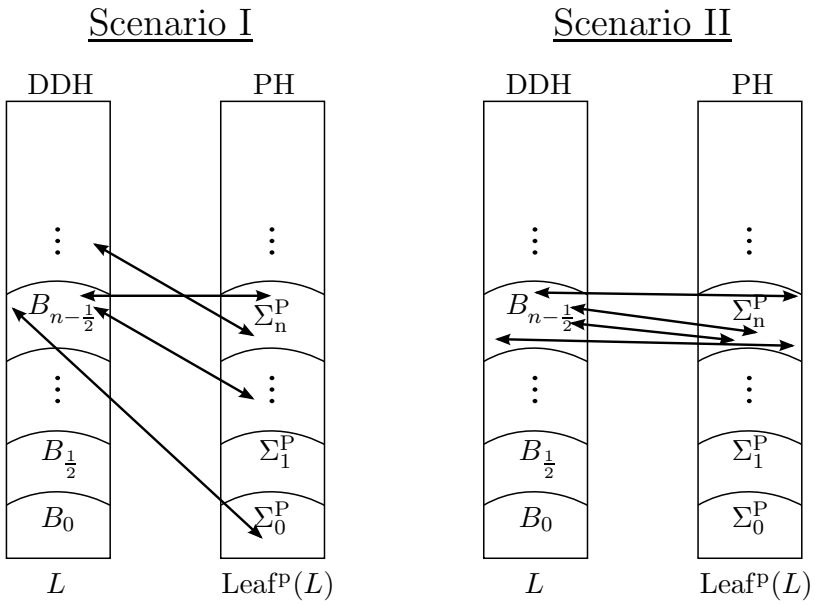


Figure 6.1: Connection between dot-depth hierarchy (DDH) and polynomial-time hierarchy (PH) via leaf languages. Which of the two scenarios does hold? Observe that due to Equations (6.1) and (6.2), arrows from left to right can never point upwards.

Theorem 6.2 ([Bor95, BKS99, Sch01]) *Let L be a regular language.*

1. *[Bor95] If $L \in \mathcal{B}_0$, then $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \subseteq \mathrm{P}$. If $L \notin \mathcal{B}_0$, then $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \supseteq \mathrm{NP}$ or $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \supseteq \mathrm{coNP}$ or $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \supseteq \mathrm{MOD}_p\mathrm{P}$ for a prime p.*
2. *[BKS99] If $L \in \mathcal{B}_{1/2}$, then $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \subseteq \mathrm{NP}$. If $L \notin \mathcal{B}_{1/2}$, then $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \supseteq \mathrm{coNP}$ or $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \supseteq \mathrm{co1NP}$ or $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \supseteq \mathrm{MOD}_p\mathrm{P}$ for a prime p.*
3. *[Sch01] If $L \in \mathcal{B}_{3/2}$, then $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \subseteq \Sigma^{\mathrm{P}}_2$. If $L \notin \mathcal{B}_{3/2}$, then $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \supseteq \forall\cdot\mathrm{UP}$ or $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \supseteq \mathrm{co}\exists!\cdot\mathrm{UP}$ or $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L) \supseteq \mathrm{MOD}_p\mathrm{P}$ for a prime p.*

For instance, by Equation (6.2), for all $L \in \mathcal{B}_{1/2}$ it holds that $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L)$ is robustly contained in NP. Theorem 6.2.2 states that the languages in $\mathcal{B}_{1/2}$ are in fact the *only* regular languages having this property. This means that for $\mathcal{B}_{1/2}$ and regular L, even the converse of (6.2) holds. We say that $\mathcal{B}_{1/2}$ and NP *perfectly correspond*:

$$L \in \mathcal{B}_{1/2} \quad \Leftrightarrow \quad \forall O(\mathrm{Leaf}^{\mathrm{p}O}_{\mathrm{u}}(L) \subseteq \mathrm{NP}^O)$$

By Wagner's new BCSV-theorem (Theorem 6.1) this is equivalent to the following:

Restricted to regular languages, $\mathcal{B}_{1/2}$ is closed under ptt-reducibility.

Here and in the following, this formulation means that $\mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_{1/2}) \cap \mathrm{REG} = \mathcal{B}_{1/2}$ where $\mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_{1/2})$ denotes $\mathcal{B}_{1/2}$'s closure under ptt-reducibility.

The example above shows that we can utilise Theorem 5.6 and Theorem 6.1 to make the notion of perfect correspondence precise:

Definition 6.3 *A class of regular languages $\mathcal{C}$ and a complexity class $\mathcal{D}$* perfectly correspond with respect to balanced leaf languages *if (restricted to regular languages) $\mathcal{C}$ is closed under plt-reducibility and* $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(\mathcal{C}) = \mathcal{D}$.

Definition 6.4 *A class of regular languages $\mathcal{C}$ and a complexity class $\mathcal{D}$* perfectly correspond with respect to unbalanced leaf languages *if (restricted to regular languages) $\mathcal{C}$ is closed under ptt-reducibility and* $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(\mathcal{C}) = \mathcal{D}$.

6.2 Polynomial-Time Tree Reducibility

In this section we provide an introduction to the polynomial-time tree reducibility. We first sketch the idea behind ptt-reducibility:

With polynomial-time tree reducibility (ptt-reducibility for short) Wagner [Wag04b] introduced the unbalanced analogue of polylog-time reducibility (plt-reducibility). For the representation of a balanced computation tree it suffices to think of a leaf-string such that each symbol is accessible in polylog-time in the length of the leaf-string. Representations of unbalanced computation trees are more complicated. Here the particular structure of the tree must be taken into account. This makes it necessary to define suitable representations of trees. Intuitively, a language B ptt-reduces to a language C if there exists a polynomial-time (in the height of the tree) computable function f that transforms trees such that for every tree t, the leaf-string of t belongs to B if and only if the leaf-string of $f(t)$ is in C.

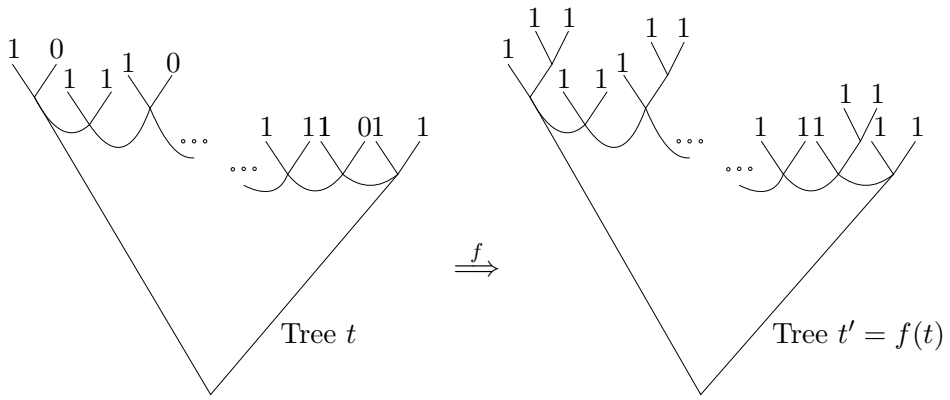


Figure 6.2: An example of a tree function f.

Example 6.5 *Let $\Sigma_1 = \{0,1\}$ and $\Sigma_2 = \{1\}$ be alphabets, and let $L_1 \subseteq \Sigma_1^*$ be defined as $L_1 = (0^*10^*1)^*0^*$, the language of all words w over $\{0,1\}^*$ such that w contains an even number of 1's. Let $L_2 \subseteq \Sigma_2^*$ be defined as $L_2 = (11)^*$, the language of all words w over $\{1\}^*$ such that w has even length. Then L_1 ptt-reduces to L_2.*[1] *This can easily be seen: Imagine M to be a nondeterministic, polynomial-time Turing machine that outputs words from Σ_1^* as leaf-strings. Roughly speaking, in order to prove that L_1 ptt-reduces to L_2, we have to transform the computation tree t of M into a tree t' whose leaf-string $\beta(t')$ is a word from Σ_2^* and $\beta(t') \in L_2$ if and only if $\beta(t) \in L_1$. Since the desired tree-function f transforms trees and not machines, it needs to be independent of the program of M, i.e., it also has to work with any other machine that outputs words from Σ_1^* as leaf-strings. In our case, the transformation can be described as follows: For all paths in t that output 1, do not change anything. For all paths that output 0, do not output 0 but branch nondeterministically into two paths and output 1 on both paths. This is shown in Figure 6.2.*

We now formalise the notion of ptt-reducibility and start with representations of trees.

Let Σ be a finite alphabet. A triple $t = (T, h, m)$ is called a Σ-*tree* if

- $T \subseteq \{0,1\}^*$, the set of paths, is finite, and $\forall z \forall u((u \sqsubseteq z \wedge z \in T) \rightarrow u \in T)$
- mapping $h : T \rightarrow \Sigma$ assigns letters to paths,
- $m \in \mathbb{N}$ is the maximal length of a path, i.e., $\forall z(z \in T \rightarrow |z| \leq m)$.

Let T_Σ be the set of all Σ-trees. A *leaf* of t is a $z \in T$ such that there is no $u \in T$ with $z \sqsubset u$. For a Σ-tree $t = (T, h, m)$, we define the *leaf word* of t as $\beta(t) =_{\text{def}} h(z_1)h(z_2)\cdots h(z_s)$ where $\{z_1, z_2, \ldots, z_s\}$ is the set of all leaves of t and $z_1 < z_2 < \cdots < z_s$. Hence, the labels of the inner nodes have no effect on the leaf word.

We describe how a Σ-tree can be encoded as a language: Choose $r \geq 1$ such that $|\Sigma| \leq 2^r$, and let $e : \Sigma \rightarrow \{0,1\}^r$ be an injective mapping. A Σ-tree $t = (T, h, m)$ is encoded by the set $O_t =_{\text{def}} \{ze(h(z)) \,\big|\, z \in T\}$ and the number $m_t =_{\text{def}} m$.

Now we define functions that transform unbalanced computation trees.

Definition 6.6 *[Wag04b] Let Σ_1 and Σ_2 be finite alphabets. A function $f : T_{\Sigma_1} \rightarrow T_{\Sigma_2}$ is called a* polynomial-time tree function *(ptt-function for short) if there exist $k > 0$ and functions $g_1 : T_{\Sigma_1} \times \{0,1\}^* \times \mathbb{N} \rightarrow \{0,1\}$ and $g_2 : T_{\Sigma_1} \times \{0,1\}^* \times \mathbb{N} \rightarrow \Sigma_2$ such that:*

- *There exists a polynomial $p(\cdot,\cdot)$ such that $g_1(t,z,m)$ and $g_2(t,z,m)$ are computable in time $p(|z|,m)$ where the tree t is accessed as the oracle O_t.*
- *It holds that $f(t) = (T', h', m_t^k + k)$ where $T' =_{\text{def}} \{z \,\big|\, g_1(t,z,m_t) = 1\}$ and $h'(z) =_{\text{def}} g_2(t,z,m_t)$.*

[1] Note that L_1 does not plt-reduce to L_2.

Finally, we define polynomial-time tree reducibility.

Definition 6.7 *[Wag04b] For $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, we define $L_1 \leq_{\mathrm{m}}^{\mathrm{ptt}} L_2$ (L_1 is* ptt-reducible *to L_2) if there exists a ptt-function $f : T_{\Sigma_1} \to T_{\Sigma_2}$ such that for all $t \in T_{\Sigma_1}$,*

$$\beta(t) \in L_1 \leftrightarrow \beta(f(t)) \in L_2.$$

Proposition 6.8 *$\leq_{\mathrm{m}}^{\mathrm{ptt}}$ and $\leq_{\mathrm{m}}^{\mathrm{plt}}$ are incomparable.*

Proof. From Example 6.5 we know that $\leq_{\mathrm{m}}^{\mathrm{ptt}}$ does not imply $\leq_{\mathrm{m}}^{\mathrm{plt}}$. For the other direction, let $\Sigma = \{1\}$. It is easy to see that $(11)^* \leq_{\mathrm{m}}^{\mathrm{plt}} (1)$, but $(11)^* \not\leq_{\mathrm{m}}^{\mathrm{ptt}} (1)$. Hence, neither of the two reducibilities implies the other. □

Remark. Although plt- and ptt-reducibility are incomparable, the following straightforward modification of Definition 6.7 yields plt-reducibility: On the one hand, the tree function in Definition 6.7 can assume that the input tree is balanced, while on the other hand we require it to output a balanced tree. So this modification is neither a restriction nor a generalisation, which is consistent with Proposition 6.8.

6.3 The ptt-Reducibility and the Dot-Depth Hierarchy

By Theorem 5.4, the levels of the dot-depth hierarchy and the levels of the polynomial-time hierarchy are closely related. Note that this connection exists for both models, balanced and unbalanced leaf languages. In this section we discuss evidence showing that for the unbalanced model this connection is much closer than that stated in Theorem 5.4.

Recall that perfect correspondences are connections closer than those stated in Theorem 5.4: For a class of regular languages $\mathcal{C}$ and a complexity class $\mathcal{D}$ that perfectly correspond with respect to unbalanced leaf languages, we know that the languages in $\mathcal{C}$ are *precisely* those whose unbalanced leaf language classes are robustly contained in $\mathcal{D}$. Therefore, there can be no regular language L' outside $\mathcal{C}$ such that $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L')$ is robustly contained in $\mathcal{D}$.

Proposition 6.9 *If $\mathcal{C}$ perfectly corresponds to $\mathcal{D}$ with respect to balanced leaf languages, then for every regular $L \notin \mathcal{C}$ there exists an oracle relative to which $\mathrm{Leaf}_{\mathrm{b}}^{\mathrm{p}}(\mathcal{C}) \not\subseteq \mathcal{D}$. The similar statement holds for unbalanced leaf languages.*

Proof. Follows from Theorems 5.6 and 6.1. □

The levels of the dot-depth hierarchy and the levels of the polynomial-time hierarchy do not correspond perfectly with respect to balanced leaf languages. In particular, for $n \geq 1$, $\mathcal{B}_{n/2}$ is not closed under plt-reducibility even if we restrict ourselves to starfree regular languages.

Theorem 6.10 *For every $n \geq 1$, $\mathcal{B}_{n-1/2}$ does not correspond perfectly to $\Sigma_{\mathrm{n}}^{\mathrm{P}}$ with respect to balanced leaf languages.*

Proof. For every $n \geq 1$, there exists $L_n \in \mathrm{SF} - \mathcal{B}_{n-1/2}$ such that L_n plt-reduces to a language in $\mathcal{B}_{1/2}$ [Gla05]. □

In contrast, we will now show that restricted to regular languages, the classes $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, $\mathcal{B}_1$, and $\mathcal{B}_{3/2}$ are closed under ptt-reducibility. In particular, these classes perfectly correspond to the classes of the polynomial-time hierarchy. While for $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, and $\mathcal{B}_{3/2}$ the latter can be derived from known results [Bor95, BKS99, Sch01], this is a new result for $\mathcal{B}_1$.

Theorem 6.11 $\mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_0) \cap \mathrm{REG} = \mathcal{B}_0$.

Proof. It suffices to argue for the inclusion from left to right. Assume there exists $L \in \mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_0) \cap \mathrm{REG}$ such that $L \notin \mathcal{B}_0$. So there exists $L' \in \mathcal{B}_0$ such that $L \leq_{\mathrm{m}}^{\mathrm{ptt}} L'$. By Theorem 6.1, if follows that $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L) \subseteq \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L')$ holds for all oracles O. From [Bor95] we obtain that for all oracles O, $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L') \subseteq \mathrm{P}^O$. Moreover, it also follows [Bor95] that one of the following statements holds true for all oracles O: $\mathrm{coNP}^O \subseteq \mathrm{P}^O$, or $\mathrm{NP}^O \subseteq \mathrm{P}^O$, or $\mathrm{MOD}_p\mathrm{P}^O \subseteq \mathrm{P}^O$ for some prime p. Note that $\mathrm{UP}^O \subseteq \mathrm{NP}^O$ and $\mathrm{UP}^O \subseteq \mathrm{MOD}_p\mathrm{P}^O$ holds relative to all oracles. Now observe that the famous oracle construction by Baker, Gill, and Solovay [BGS75] separates both coNP from P and UP from P. This is a contradiction. Hence no such language L can exist in $\mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_0) \cap \mathrm{REG}$. □

Theorem 6.12 $\mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_{1/2}) \cap \mathrm{REG} = \mathcal{B}_{1/2}$.

Proof. It suffices to argue for the inclusion from left to right. Assume there exists $L \in \mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_{1/2}) \cap \mathrm{REG}$ such that $L \notin \mathcal{B}_{1/2}$. So there exists $L' \in \mathcal{B}_{1/2}$ such that $L \leq_{\mathrm{m}}^{\mathrm{ptt}} L'$. Hence for all oracles O, $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L') \subseteq \mathrm{NP}^O$. By Borchert, Kuske, and Stephan [BKS99], for all oracles O, $\mathrm{coUP}^O \subseteq \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L)$. By Theorem 6.1, for all oracles O, $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L) \subseteq \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L')$ and therefore, $\mathrm{coUP}^O \subseteq \mathrm{NP}^O$. This contradicts an oracle construction by Eppstein et al. [EHTY92]. □

We now prove a technical lemma which will enable us to show that $\mathcal{B}_1$ and the Boolean closure of NP correspond perfectly.

The decidability of $\mathcal{B}_1$ is due to Knast [Kna83]. Schmitz [Sch01] transformed the algebraic conditions given by Knast into a forbidden pattern (Figure 6.3), which we will exploit in the proof of the lemma.

For the next lemma, we need two notations: $\mathrm{P}^{\mathrm{NP}[\log n]}$ says that the P-machine can query the NP-oracle at most logarithmically often. Furthermore, $\mathrm{P}_{\|}^{\mathrm{NP}}$ says that the P-machine has to query the oracle nonadaptively.

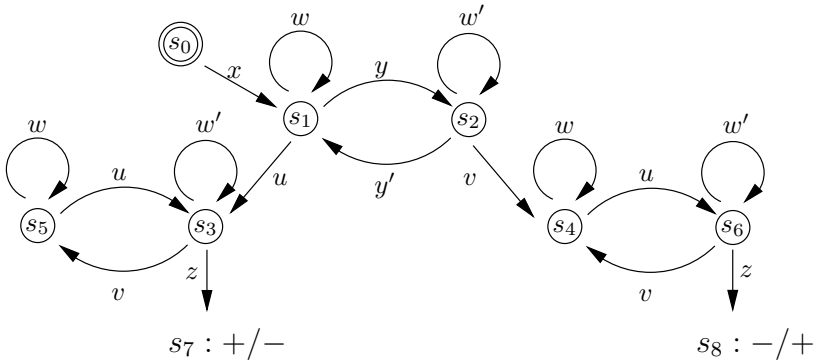


Figure 6.3: Pattern P_1 where w, w' are nonempty words. Nonexistence of this pattern characterises $\mathcal{B}_1$.

Lemma 6.13 *Let $L \in \mathrm{REG} \setminus \mathcal{B}_1$. Then there exists an oracle B such that $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}\,B}(L) \not\subseteq \mathrm{P}^{\mathrm{NP}[\epsilon \cdot \log n]^B}$ for all $\epsilon < 1$.*

Proof. Let A be an alphabet with $\#A \geq 2$ and $L \subseteq \Sigma^*$ such that $L \in \mathrm{REG} \setminus \mathcal{B}_1$. Hence, the minimal automaton of L contains pattern P_1 (see Figure 6.3) and there exist $u, v, x, y, y', z \in \Sigma^*$ and $w, w' \in \Sigma^+$ as apparent in Figure 6.3. Without loss of generality, we assume that the minimal automaton contains the first version of the pattern, i.e., state s_7 is accepting and state s_8 is rejecting. Let L_{P_1} be the language of all words in $x\{u, v, w, w', y, y'\}^* z$ such that the minimal automaton of L moves along the paths drawn in Figure 6.3 and finally reaches s_7. Let L'_{P_1} be the similar set of words leading to s_8. Clearly, $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L_{P_1}, L'_{P_1}) \subseteq \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(L)$. We construct B such that for all $\epsilon < 1$,

$$\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}\,B}(L_{P_1}, L'_{P_1}) \not\subseteq \mathrm{P}_{\|}^{\mathrm{NP}[n^\epsilon]^B}.$$

This implies that for all $\epsilon < 1$,

$$\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}\,B}(L_{P_1}, L'_{P_1}) \not\subseteq \mathrm{P}^{\mathrm{NP}[\epsilon \cdot \log(n)]^B}.$$

Let $e \notin \Sigma$ be a new letter. For $n \in \mathbb{N}$ let $\alpha_{0,n} < \alpha_{1,n} < \ldots < \alpha_{2^n-1,n}$ be the words of $\{0,1\}^n$ in lexicographical order.

For any set $D \subseteq \{0,1\}^*$ with characteristic function c_D, the *characteristic sequence of D restricted to words of length n* is defined as

$$C_D(n) =_{\mathrm{def}} c_D(\alpha_{0,n}) c_D(\alpha_{1,n}) \ldots c_D(\alpha_{2^n-2,n}) c_D(\alpha_{2^n-1,n}).$$

Such a characteristic sequence can be considered as a sequence of letters from $\Sigma \cup \{e\}$ where $\lceil \log(\#\Sigma + 1) \rceil$ bits of $C_D(n)$ encode a letter from $\Sigma \cup \{e\}$. Denote this new sequence by $C'_D(n)$ and observe that its length is greater than $2^{n-\#\Sigma}$. Let $C'_D(n)_{|\Sigma}$ be the sequence obtained by removing all e's from $C'_D(n)$.

We say that the sequence $C_D(n)$ is *valid for pattern P_1* if the following holds:

- $C'_D(n)$ does not contain a factor e^{n+1}, and
- $C'_D(n)_{|\Sigma} \in L_{P_1} \cup L'_{P_1}$.

We call a valid sequence *accepted (resp., rejected) by pattern* P_1 if it belongs to L_{P_1} (resp., L'_{P_1}). Hence, a valid sequence $C_D(n)$ encodes a sequence $C'_D(n)$ over $\Sigma \cup \{e\}$ which may contain only short e-blocks.

We will define a fast-growing tower function $t : \mathbb{N} \to \mathbb{N}$ such that $t(n+1) = 2^{t(n)}$ for $n \geq 0$. For an arbitrary oracle O, we define our witness language W^O as follows:

$$W^O =_{\text{def}} \{0^{t(n)} \mid n \geq 0 \text{ and } C_O(t(n)) \text{ is accepted by pattern } P_1\}$$

Throughout the construction we will ensure that for all n, the sequence $C_B(t(n))$ is valid for pattern P_1. This implies $W^B \in \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}\,B}(L_{P_1}, L'_{P_1})$: On input 0^m, an unbalanced machine first verifies that $m = t(n)$ for some n, and then produces a computation tree with leaf string $C'_O(m)$. Since $C'_O(m)$ only contains short blocks of e's, this machine can reorganise its computation tree such that all e's are removed from the leaf string. So it remains to show that $W^B \notin \mathrm{P}^{\mathrm{NP}[\epsilon \cdot \log(n)]^B}$.

Our oracle B will be defined as the union of (finite) oracle stages $B_i, i \geq 1$, which are constructed iteratively. Each stage B_n is characterised by oracle words of length $t(n)$ and therefore by the sequence $C_B(t(n))$. Let $B[k,j] =_{\text{def}} \bigcup_{k \leq i \leq j} B_i$ denote an interval of oracle stages.

We enumerate $\mathrm{P}_{\|}^{\mathrm{NP}[m^\epsilon]}$-machines as follows:

Consider an enumeration of all tuples (M, N, p, ϵ) such that M is a deterministic polynomial-time oracle Turing machine, N is a nondeterministic polynomial-time oracle Turing machine, p is a polynomial and $\epsilon < 1$. We interpret M as the base machine and N as the oracle machine.

By defining the first value $t(0)$ of the tower function sufficiently large and $t(n+1) =_{\text{def}} 2^{t(n)}$, we can ensure that the enumeration satisfies the following technical requirements. For the n-th tuple of the enumeration, (M, N, p, ϵ), all of the following holds:

1. $p(t(n)) \leq 2^{\log^2 t(n)}$
2. $3 \log^2 t(n) \leq t(n)^{(1-\epsilon)/2}$
3. $2^{t(n)}/2^{t(n)^{(1+\epsilon)/2}} \geq 2 \cdot \#A \cdot |ww'yy'uv|$
4. Let the running times of M and N be bounded by polynomials q and r, respectively. Then it holds that $r(q(n)) \leq p(n)$.
5. M on input x asks at most $|x|^\epsilon$ nonadaptive queries to the oracle $L(N)$.

Let (M, N, p, ϵ) be the n-th tuple in our enumeration and let $m = t(n)$. We diagonalise against (M, N, p, ϵ) through ensuring

$$L(M^{B[1,n],L(N^{B[1,n]})}) \neq W^{B[1,n]}. \tag{6.3}$$

Notice that M can access both oracles, $B[1, n]$ and $L(N^{B[1,n]})$.

We describe the main idea behind the diagonalisation against (M, N, p, ϵ): We start with an oracle B_n such that $C_B(m)$ is accepted by P_1. After that we simulate M with the so-far constructed oracle $(B[1, n])$ on input 0^m and determine segments in B_n that have to be reserved. If M rejects 0^m we are done for this stage. Otherwise we change B_n on non-reserved positions, such that $C_B(m)$ is still *valid* but now rejected by P_1 (here the e's compensate length differences).

We then repeat the simulation of M on input 0^m with the modified oracle and update the list of reserved segments. If M still accepts we are done, otherwise we modify non-reserved positions such that $C_B(m)$ remains valid but accepted by P_1 again. We will show that after $\epsilon \cdot \log m$ such rounds, M on input 0^m will err in its decision.

The detailed construction of the diagonalisation against (M, N, p, ϵ) follows.

We define

$$\beta =_{\text{def}} xw\gamma wuz,$$

such that $\gamma \in \{w, e\}^*$, γ does not contain a factor e^{m+1}, and $|\beta| = 2^{m-\#A}$. We start with $B_n \subseteq \{0, 1\}^m$ such that $C'_{B_n}(m) = \beta$. Clearly, $C_{B_n}(m)$ is accepted by pattern P_1: Whether a valid sequence is accepted or rejected is determined by the first occurrence of a word from $\{u, v\}$ in the encoded sequence; for u the sequence is accepted, for v it is rejected.

Let F denote the set of reserved segments; $F = \emptyset$ at the beginning. F is supposed to contain words of length m that we will not modify in the further construction. Simulate $M^{B[1,n]}$ on input 0^m. If M rejects, (6.3) is fulfilled and the construction of stage B_n is complete. So assume M accepts. Let Q_1 be the set of M's queries to B_n on input 0^m. Thus, $\#Q_1 \leq p(m)$. Let $q_1, \ldots, q_k$ be M's nonadaptive queries to N where $k \leq m^\epsilon$. Let $Q_+ \subseteq \{q_1, \ldots, q_k\}$ be the set of positively answered queries. Hence, for $q \in Q_+$, the nondeterministic machine N on input q produces at least one accepting path. We define

$$Q_2 =_{\text{def}} \{q \mid \exists q' \in Q_+(N \text{ on input } q' \text{ queries } q \text{ on its leftmost accepting path})\}.$$

Observe that $\#Q_2 \leq p(m)^2$. We now set $F = F \cup Q_1 \cup Q_2$. Since $\#F \leq p(m)^3$ and $|C_{B_n}(m)| = 2^m$, there exist $2^m/p(m)^3$ consecutive words of length m that are not in F. These words represent a segment s in β. By the construction of β, $s \in \{w, e\}^*$. In the next step, s is replaced by a segment $s' \in y\{w', e\}^*v$ such that $|s'| = |s|$ and s does not contain a factor e^{m+1}. Observe that the purpose of e in this construction is to compensate

differences in the lengths of y, w, w' and v. After this modification, $C_{B_n}(m)$ is still valid but now rejected by P_1. Since all further modifications in later rounds will be restricted to the segment s', we reserve all the rest of the oracle at this stage, i.e., F now contains all words from $\{0,1\}^m$ except those encoding s'.

Again, we simulate $M^{B[1,n]}$ on input 0^m and now assume that it has noticed the deception and thus rejects. Let Q_3 be the set of queries to B_n during this simulation. Since $Q_2 \subseteq F$, no query in Q_+ can have flipped from positive to negative. Consequently, there have to be queries in $\{q_1, \ldots, q_k\} \setminus Q_+$ which have been answered positively by N during the second simulation of M. Let Q'_+ be the set of these queries. We repeat the above construction by defining the set

$$Q_4 =_{\text{def}} \{q \mid \exists q' \in Q'_+ (N \text{ on input } q' \text{ queries } q \text{ on its leftmost accepting path})\}.$$

We have $\#Q_3 \leq p(m)$ and $\#Q_4 \leq p(m)^2$.

Set $F = F \cup Q_3 \cup Q_4$. Hence, we still find

$$\frac{2^m}{p(m)^3 \cdot p(m)^3}$$

consecutive words of length m that are not in F. These correspond to a segment $s_1 \in \{w', e\}^*$ which has not been reserved yet. This segment is replaced by a segment $s'_1 \in y'\{w, e\}^* u$ with $|s_1| = |s'_1|$. This modification causes $C_{B_n}(m)$ to be accepted by P_1.

We can deceive M again by repeating the above procedure. After at most k rounds, no more of M's queries to N can flip from negative to positive. At that point, M cannot change its behavior any longer. Each round the size of the non-reserved area of $\{0,1\}^m$ is divided by at most $p(m)^3$. Hence after k rounds we still have a segment of size

$$\frac{2^m}{p(m)^{3k}} \geq \frac{2^m}{p(m)^{3m^\epsilon}} \geq \frac{2^m}{2^{3m^\epsilon(\log^2 m)}} \geq \frac{2^m}{2^{m^{(1-\epsilon)/2} m^\epsilon}} = \frac{2^m}{2^{m^{(1+\epsilon)/2}}} \geq 2 \cdot \#A \cdot |ww'yy'uv|.$$

Therefore, after k rounds we can still find a sufficiently large non-reserved area. We can then modify this segment to deceive M one final time. □

Utilising Theorem 6.1, we can translate this oracle separation into a statement about the ptt-closure of $\mathcal{B}_1$.

Theorem 6.14 $\mathcal{R}^{\text{ptt}}(\mathcal{B}_1) \cap \text{REG} = \mathcal{B}_1$.

Proof. It suffices to argue for the inclusion from left to right. Assume there exists $L \in \mathcal{R}^{\text{ptt}}(\mathcal{B}_1) \cap \text{REG}$ such that $L \notin \mathcal{B}_1$. So there exists $L' \in \mathcal{B}_1$ such that $L \leq_{\text{m}}^{\text{ptt}} L'$. By Theorem 6.1, for all oracles O, we then have $\text{Leaf}_{\text{u}}^{\text{p}O}(L) \subseteq \text{Leaf}_{\text{u}}^{\text{p}O}(L')$. Theorem 5.4 holds

relative to all oracles. Therefore, for all oracles O, it holds that $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}^O}(L') \subseteq BC(\mathrm{NP})^O$. This contradicts Lemma 6.13. □

As a consequence, we obtain the first gap theorem of leaf language definability above the Boolean closure of NP.

Corollary 6.15 *Let $\mathcal{D} = \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(\mathcal{C})$ for some $\mathcal{C} \subseteq \mathrm{REG}$. Then $\mathcal{D} \subseteq \mathrm{BC(NP)}$ or there exists an oracle O such that $\mathcal{D}^O \not\subseteq \mathrm{P}^{\mathrm{NP}[\epsilon \cdot \log n]^O}$ for all $\epsilon < 1$.*

Theorem 6.16 $\mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_{3/2}) \cap \mathrm{REG} = \mathcal{B}_{3/2}$.

Proof. It suffices to argue for the inclusion from left to right. Assume there exists $L \in \mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_{3/2}) \cap \mathrm{REG}$ such that $L \notin \mathcal{B}_{3/2}$. So there exists $L' \in \mathcal{B}_{3/2}$ such that $L \leq_{\mathrm{m}}^{\mathrm{ptt}} L'$. Hence for all oracles O, $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}^O}(L') \subseteq \Sigma_2^{p O}$. By Schmitz [Sch01], for all oracles O, $\forall!\cdot\exists!\cdot\mathrm{P}^O \subseteq \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}^O}(L)$. By Theorem 6.1, for all oracles O, $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}^O}(L) \subseteq \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}^O}(L')$ and therefore, $\forall!\cdot\exists!\cdot\mathrm{P}^O \subseteq \Sigma_2^{p O}$. This contradicts an oracle construction by Spakowski and Tripathi [ST07]. □

As stated in the Theorems 6.11, 6.12, 6.14, and 6.16, the classes $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, $\mathcal{B}_1$, and $\mathcal{B}_{3/2}$ are closed under ptt-reducibility if we restrict ourselves to regular languages. We explain this difference and show that the restriction to regular languages is crucial: For $k \geq 1$, $\mathcal{B}_{k/2}$ is not closed under ptt-reducibility.

Theorem 6.17 *There exists $B \in \mathrm{NP} \setminus \mathrm{REG}$ such that $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(B) \subseteq \mathrm{NP}$.*

Proof. We use the pairing function $\langle \cdot, \cdot \rangle$ that is defined as follows for letters a_i and b_i.

$$\langle a_1 a_2 \cdots a_k, b_1 b_2 \cdots b_l \rangle =_{\mathrm{def}} 0a_1 0a_2 \cdots 0a_k 1b_1 1b_2 \cdots 1b_l$$

Let $N_1, N_2, \ldots$ be an enumeration of nondeterministic polynomial-time bounded Turing machines such that N_i on inputs of length n has running time $n^i + i$. We may assume that given i, one can determine the machine N_i in polynomial-time in $|i|$.

Every word appears as a leaf string of a suitable computation. This changes if we demand that the leaf string is generated by a short input. A word w is called *honestly generated* if it is generated by a machine N_i on input of a sufficiently small word x. We make this precise with the definition of B which consists of all honestly generated words.

$$B =_{\mathrm{def}} \{w \mid (\exists i \leq |w|/2)(\exists x \in \Sigma^*, |x|^i + i < |w|)[\beta_{N_i}(x) = w]\}$$

Assume we are given w, i, and x as above. The running time of N_i on x is $|x|^i + i < |w|$. Therefore, in time $O(|w|^2)$ we can determine the machine N_i, can simulate the first $|w|$ computation paths of $N_i(x)$, and can test whether $\beta_{N_i}(x) = w$. This shows $B \in \mathrm{NP}$.

Let $n \geq 2$ and $1 \leq i \leq n/2$. We estimate $\#(B \cap \Sigma^n)$ as follows.

$$\#(B \cap \Sigma^n) \leq \sum_{i=1}^{n/2} \#\{x \in \Sigma^* \,\big|\, |x| \leq (n-i-1)^{1/i}\} \leq \sum_{i=1}^{n/2} 2^{n-i} = 2^n \sum_{i=1}^{n/2} 2^{-i} < 2^n$$

This shows that at least one word of any length belongs to $\overline{B}$. In particular, $\overline{B}$ is infinite.

We argue that $B \notin$ REG. For this we start with the description of a nondeterministic machine N on input $\langle \mathcal{M}, k\rangle$ where k is a natural number and $\mathcal{M}$ is a deterministic finite automaton. First, N deterministically computes nonempty words u, v, z such that for all $i \geq 0$, $uv^i z \notin L(\mathcal{M})$. If such words do not exist, then N generates the leaf string 0. Otherwise, in a nondeterministic way N generates the leaf string $uv^k z$. Observe that the words u, v, z, if they exist, can be computed in polynomial-time which shows that N is polynomial-time bounded. Therefore, $N = N_j$ for some $j \geq 1$.

Assume $B \in$ REG, i.e., $B = L(\mathcal{M})$ for some finite automaton $\mathcal{M}$. Choose l sufficiently large such that $l \geq 2j$ and $l > |\langle \mathcal{M}, l\rangle|^j + j$. Let $x =_{\text{def}} \langle \mathcal{M}, l\rangle$ and $w =_{\text{def}} \beta_{N_j}(x)$. Since $\overline{B}$ is infinite, there exist nonempty words u, v, z such that for all $i \geq 0$, $uv^i z \notin L(\mathcal{M})$. Therefore, for suitable such words it holds that $w = uv^l z \notin L(\mathcal{M})$. So $j \leq |w|/2$ and $|w| > |x|^j + j$. It follows that $w \in B - L(\mathcal{M})$ which contradicts the assumption $B = L(\mathcal{M})$ and which shows $B \notin$ REG.

Finally we show $\text{Leaf}^{\text{p}}_{\text{u}}(B) \subseteq$ NP. Fix any $j \geq 1$ and let $L = \{x \,\big|\, \beta_{N_j}(x) \in B\}$. It suffices to show $L \in$ NP. Let x be an arbitrary word of length ≥ 2. Define $w =_{\text{def}} \beta_{N_j}(x)$ and observe

$$\begin{aligned} x \in L &\Leftrightarrow w \in B \\ &\Leftrightarrow (|x|^j + j < |w|) \vee (|x|^j + j \geq |w| \wedge w \in B). \end{aligned}$$

The first $|x|^j + j$ letters of the leaf string w can be determined in polynomial-time in $|x|$. So the condition $|x|^j + j < |w|$ is decidable in polynomial-time in $|x|$. If $|x|^j + j \geq |w|$, then $w \in B$ can be decided in nondeterministic polynomial-time in $|x|$. Hence the condition on the right-hand side is decidable in NP which shows $L \in$ NP. □

Corollary 6.18 *1. There exists $B \in \text{NP} \setminus \text{REG}$ such that $B \in \mathcal{R}^{\text{ptt}}(\mathcal{B}_{1/2})$.*

2. For every $k \geq 1$, $\mathcal{B}_{k/2}$ is not closed under $\leq^{\text{ptt}}_{\text{m}}$-reducibility.

Proof. Let $C =_{\text{def}} (0 \cup 1)^* 1 (0 \cup 1)^*$ and define B as in Theorem 6.17. There we show $B \in \text{NP} \setminus \text{REG}$ and $\text{Leaf}^{\text{p}}_{\text{u}}(B) \subseteq$ NP. The argument for the latter inclusion is relativisable. Therefore, for all oracles O, $\text{Leaf}^{\text{p}O}_{\text{u}}(B) \subseteq \text{NP}^O = \text{Leaf}^{\text{p}O}_{\text{u}}(C)$. By Theorem 6.1, $B \leq^{\text{ptt}}_{\text{m}} C$ and hence $B \in \mathcal{R}^{\text{ptt}}(\mathcal{B}_{1/2})$. This shows the first statement and the second one follows immediately. □

We have shown that for $k \geq 1$, $\mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_{k/2})$ contains non-regular languages. However, this cannot happen in the case of $\mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_0)$. We will address this issue in the outlook section at the end of this chapter.

In order to give us an impression of what ptt-reductions can do (recall that we have just learnt that ptt-reductions can reduce languages from NP − REG to $\mathcal{B}_{1/2}$), we now state an upper bound for the complexity of the $\leq_{\mathrm{m}}^{\mathrm{ptt}}$-closure of regular languages.

Theorem 6.19 $\mathcal{R}^{\mathrm{ptt}}(\mathrm{REG}) \subseteq \bigcup_{k\geq 1} \mathrm{DSPACE}(\log^k n)$.

Proof. Let $L \in \mathcal{R}^{\mathrm{ptt}}(\mathrm{REG})$, i.e., there exists $L' \in \mathrm{REG}$ such that $L \leq_{\mathrm{m}}^{\mathrm{ptt}} L'$ via ptt-function f. So there exist $k > 0$ and functions g_1 and g_2 as in Definition 6.6. Both functions are polynomial-time computable when the tree is accessed as an oracle. For a word x, let t_x denote the balanced binary tree that has the leaf string x.

Let $m = \lceil \log |x| \rceil^k + k$. We describe an algorithm that computes $\beta(f(t_x))$: Consider all strings z of length $\leq m$ in lexicographical order. If $g_1(t_x, z, \lceil \log |x| \rceil) = 1$, then output $g_2(t_x, z, \lceil \log |x| \rceil)$. Consider the next string z.

This algorithm computes $\beta(f(t_x))$, since it exactly simulates f. If t_x is accessed as oracle, then $g_1(t_x, z, \lceil \log |x| \rceil)$ and $g_1(t_x, z, \lceil \log |x| \rceil)$ are computable in polynomial time in $\log |x|$. Given x, an oracle access to t_x can be simulated in logarithmic space. Therefore, the algorithm above can be simulated in polylogarithmic space in $|x|$. Given $\beta(f(t_x))$, we can test in constant space whether $\beta(f(t_x)) \in L'$. The theorem follows, since $x \in L \Leftrightarrow \beta(t_x) \in L \Leftrightarrow \beta(f(t_x)) \in L'$. □

Due to this theorem, we can now specify the complexity of non-regular sets C such that $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(C) \subseteq \mathrm{NP}$. Recall that for regular sets, we already know by Theorem 6.12 that only languages in $\mathcal{B}_{1/2}$ come into question. Accordingly it is unlikely that such sets are NP-complete. In particular, this applies to the set B that was used in Theorem 6.17 and Corollary 6.18.

Corollary 6.20 *Let C be a set. Then the following holds: If $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(C) \subseteq \mathrm{NP}^O$ for all oracles O, then $C \in \bigcup_{k\geq 1} \mathrm{DSPACE}(\log^k n)$.*

Proof. For all oracles O, $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(C) \subseteq \mathrm{NP}^O = \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(0^*1\{0,1\}^*)$. So $C \leq_{\mathrm{m}}^{\mathrm{ptt}} 0^*1\{0,1\}^*$ and hence $C \in \mathcal{R}^{\mathrm{ptt}}(\mathrm{REG}) \subseteq \bigcup_{k\geq 1} \mathrm{DSPACE}(\log^k n)$. □

We remark that since $\mathrm{PSPACE} = \mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(\mathrm{REG})$ [HLS+93], the last corollary remains valid if we replace NP with PSPACE.

6.4 Summary and Outlook

In this chapter, we analysed the ptt-reducibility and proved that restricted to regular languages, the levels 0, 1/2, 1, and 3/2 of the dot-depth hierarchy are closed under ptt-reducibility. We explained that these results indicate that the connection between dot-depth and polynomial-time hierarchy is closer than formerly known: We showed that on the lower levels, the dot-depth and the polynomial-time hierarchy correspond perfectly, i.e., languages with a certain dot-depth precisely characterise a certain level of the polynomial-time hierarchy:

- $\mathcal{B}_0$ perfectly corresponds to P with respect to unbalanced leaf languages.
- $\mathcal{B}_{1/2}$ perfectly corresponds to NP with respect to unbalanced leaf languages.
- $\mathcal{B}_{3/2}$ perfectly corresponds to Σ_2^{P} with respect to unbalanced leaf languages.

We showed that this is equivalent to proving that restricted to regular languages, the classes $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, and $\mathcal{B}_{3/2}$ are closed under ptt-reducibility.

Furthermore, we showed that restricted to regular languages, $\mathcal{B}_1$ is closed under ptt-reducibility. From this we obtained a new perfect correspondence:

- $\mathcal{B}_1$ perfectly corresponds to the Boolean closure of NP with respect to unbalanced leaf languages.

In summary, we have shown that the following holds for every regular language L :

$$\begin{aligned} L \in \mathcal{B}_0 &\Leftrightarrow \forall O(\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L) \subseteq \mathrm{P}^O) \\ L \in \mathcal{B}_{1/2} &\Leftrightarrow \forall O(\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L) \subseteq \mathrm{NP}^O) \\ L \in \mathcal{B}_1 &\Leftrightarrow \forall O(\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L) \subseteq \mathrm{BC}(\mathrm{NP})^O) \\ L \in \mathcal{B}_{3/2} &\Leftrightarrow \forall O(\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L) \subseteq \Sigma_2^{\mathrm{P}O}) \end{aligned}$$

We remark that the perfect correspondence between $\mathcal{B}_0$ and P can be improved further. Unlike all classes $\mathcal{B}_{n/2}$ for $n \geq 1$ (confer Theorem 6.17), one can show that the class $\mathcal{B}_0$ is closed under ptt-reducibility even without the restriction to regular languages.

Theorem 6.21 *[GTW06]* $\mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_0) = \mathcal{B}_0$.

This special case yields an even tighter connection of $\mathcal{B}_0$ and P: Not only that $\mathcal{B}_0$ and P perfectly correspond, but it even holds that for any language $L \notin \mathcal{B}_0$ (this includes all non-regular languages) there exists an oracle O such that $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}O}(L) \supsetneq \mathrm{P}^O$.

As the dot-depth hierarchy perfectly corresponds to the polynomial-time hierarchy on the lower levels, we consider this as evidence that restricted to regular languages, *all* levels of the dot-depth hierarchy might be closed under ptt-reducibility:

Conjecture 6.22 *For every* $k \geq 0$, $\mathcal{R}^{\mathrm{ptt}}(\mathcal{B}_{k/2}) \cap \mathrm{REG} = \mathcal{B}_{k/2}$.

However, whether this holds remains a challenging open question. We think that a particular difficulty will be the lack of forbidden pattern characterisations for the higher levels of the dot-depth hierarchy. Although a complete characterisation of the dot-depth hierarchy's levels in terms of forbidden patterns would be very desirable, this seems to be an extremely difficult (if not impossible) task. Such a uniform characterisation would solve the dot-depth-problem, which is often considered to be one of the most important problems in formal languages [Pin98].

Chapter 7

ε-Leaf-Language Classes

In the last chapter we learned that unbalanced leaf languages establish a close connection between the lower levels of the dot-depth hierarchy and the polynomial-time hierarchy, and we conjectured that this connection can be extended to the higher levels of the hierarchies. Also, we discussed that balanced leaf languages do not allow a similarly close connection.

In this chapter we offer a useful completion of the known leaf-language concepts. This is the concept of *ε-leaf languages*. It is inspired by the observation that rejecting paths of nondeterministic computations act as *neutral* elements. In this sense we allow nondeterministic transducers to output not only single letters but also to output the empty word ε which is the neutral element of Σ^*.

We consider the following aspects to be the main advantages of this approach:

First, we can assume that the nondeterministic machines involved have balanced computation trees. This allows us to write down proofs in a more concise way as we do not have to care about the representation of complicated objects like trees. This is an advantage over the unbalanced leaf-language model. Second, we will show that this new model allows us to establish a tight connection between the polynomial-time hierarchy and the Straubing-Thérien hierarchy [Str81, Thé81, Str85], another very important hierarchy of starfree languages. In turn, this is an advantage over the balanced leaf-language model. In a sense, the ε-leaf-language approach combines the advantages of the other two leaf-language notions.

7.1 The ε-Leaf-Language Model

In this section, we introduce a new leaf-language model, the ε-model. After the formal definition we introduce the pte-reducibility which allows us to formulate and prove an

analogue of the BCSV-theorem. Furthermore, we show that the new model connects the polynomial-time hierarchy to the Straubing-Thérien hierarchy.

Throughout the chapter, we will often compare the new ε-model to the existing models. For the sake of brevity, we will refer to the unbalanced leaf-language model as *u-model*, to the balanced leaf-language model as *b-model*, and to the new ε-model as *e-model*, respectively. Analogously, we use the terms *u-class*, *b-class*, and *e-class* when we talk about complexity classes defined by the different leaf-language models.

Recall the definitions of the various subword relations.

Example 7.1 *It holds that* $10 \preceq_3 1110$ *and* $\{0, 1, 10\} \preceq_7 1110$.

Recall that whenever L and K are disjoint languages over the alphabet Σ, we also write $(L|K) \subseteq \Sigma^*$ as an abbreviation. Hence, whenever we talk about a pair $(L|K) \subseteq \Sigma^*$ of languages, we assume that L and K are disjoint.

Definition 7.2 *Let* $(L|K) \subseteq \Sigma^*$. *The class* $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L|K)_{\Sigma}$ *consists of all languages* A *for which there exists a nondeterministic polynomial time transducer* M *producing on every computation path a symbol from* Σ *or the empty word* ε *such that the following holds:*

$$\begin{aligned} x \in A &\Rightarrow \beta_M(x) \in L, \\ x \notin A &\Rightarrow \beta_M(x) \in K. \end{aligned}$$

For a leaf-language class $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L_1|L_2)_{\Sigma}$ we will often omit the subscript $_{\Sigma}$ when Σ is the smallest alphabet such that $L_1 \cup L_2 \subseteq \Sigma^*$. In these cases, it will be clear from the context what Σ is. If $L_2 = \Sigma^* - L_1$, then we will also write $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L_1)$. Notice that it makes no difference whether we use balanced or unbalanced computation trees: Paths that are too short can easily be extended by attaching subtrees where all leaves but the leftmost leaf output the empty word ε. So for convenience we may assume that paths can not only output single letters but also arbitrary words.

Example 7.3 *The following holds:*

1. $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(11^*|\varepsilon) = \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(0^*1(0 \cup 1)^*|0^*) = \mathrm{NP}$.
2. $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(1) = 1\mathrm{NP}$.
3. $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(1|\varepsilon) = \mathrm{UP}$.
4. $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(0^*)_{\{0,1\}} = \mathrm{coNP}$.

It is immediately clear that the u-model and the b-model are restrictions of the e-model.

Proposition 7.4 *For all leaf languages* $B \subseteq \Sigma^*$, *it holds that*

$$\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(B) \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(B) \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(B).$$

Moreover, it is intuitively clear that the presence of the neutral element ε gives the class $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(B)$ some inherent nondeterministic power which makes $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(B)$ seemingly bigger than P. We will discuss this issue later on and we will identify UP $\cap$ coUP as a lower bound (we obtain stronger bounds if we restrict ourselves to regular languages B).

7.1.1 Polylogarithmic-Time ε-Reducibility

We now define the polylogarithmic-time ε-reducibility (pte-reducibility, for short), the counterpart of plt-reducibility (in the balanced leaf-language model) and ptt-reducibility (in the unbalanced leaf-language model).

Recall that a function g is computable in polylogarithmic time if there exists $k \geq 1$ such that $g(x)$ can be computed in time $\mathcal{O}(\log^k |x|)$ by a Turing machine which accesses the input as an oracle.

For a finite alphabet Σ and $a \notin \Sigma$, define a homomorphism $h_{\Sigma,a} : (\Sigma \cup \{a\})^* \to \Sigma^*$ by $h_{\Sigma,a}(b) =_{\mathrm{def}} b$ for $b \in \Sigma$ and $h_{\Sigma,a}(a) =_{\mathrm{def}} \varepsilon$.

Definition 7.5 *Let $(L|K) \subseteq \Sigma_1^*, (L'|K') \subseteq \Sigma_2^*$ and $a \notin \Sigma_1^* \cup \Sigma_2^*$. Then $(L|K) \leq^{\mathrm{pte}}_{\mathrm{m}} (L'|K')$ if and only if there exists a function $f : (\Sigma_1 \cup \{a\})^* \to (\Sigma_2 \cup \{a\})^*$ such that*

- *there exist functions $g : ((\Sigma_1 \cup \{a\})^* \times \mathbb{N}) \to \Sigma_2 \cup \{a\}$, $h : (\Sigma_1 \cup \{a\})^* \to \mathbb{N}$ computable in polylogarithmic time such that for all $x \in (\Sigma_1 \cup \{a\})^*$ it holds that $f(x) = g(x,1)g(x,2)\ldots g(x,h(x))$,*
- *for all $x \in (\Sigma_1 \cup \{a\})^*$, $\big(h_{\Sigma_1,a}(x) \in L \Rightarrow h_{\Sigma_2,a}(f(x)) \in L'\big)$,*
- *for all $x \in (\Sigma_1 \cup \{a\})^*$, $\big(h_{\Sigma_1,a}(x) \in K \Rightarrow h_{\Sigma_2,a}(f(x)) \in K'\big)$.*

If $(L|K) \leq^{\mathrm{pte}}_{\mathrm{m}} (L'|K')$ and $K = \Sigma_1^* - L$ and $K' = \Sigma_2^* - L'$, we write $L \leq^{\mathrm{pte}}_{\mathrm{m}} L'$ as abbreviation.

The following leads to an alternative definition.

Definition 7.6 *For any language $L \subseteq \Sigma^*$ and $a \notin \Sigma$, we define $L_a \subseteq (\Sigma \cup \{a\})^*$ as*

$$L_a =_{\mathrm{def}} \{a^{m_0} w_1 a^{m_1} w_2 \ldots a^{m_{n-1}} w_n a^{m_n} \,\big|\, m_0, \ldots, m_n \geq 0, w_i \in \Sigma, w_1 w_2 \ldots w_n \in L\}.$$

The following is an immediate consequence of the definition of $\leq^{\mathrm{pte}}_{\mathrm{m}}$:

Proposition 7.7 *For leaf languages $(L|K) \subseteq \Sigma_1^*$, $(L'|K') \subseteq \Sigma_2^*$ and $a \notin \Sigma_1 \cup \Sigma_2$, it holds that $(L|K) \leq^{\mathrm{pte}}_{\mathrm{m}} (L'|K')$ if and only if $(L_a|K_a) \leq^{\mathrm{plt}}_{\mathrm{m}} (L'_a|K'_a)$.*

Lemma 7.8 *For $(L|K) \subseteq \Sigma^*$ and $a \notin \Sigma$, $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L|K) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_a|K_a) = \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(L_a|K_a) = \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L_a|K_a)$.*

Proof. It suffices to show that $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L|K) \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_a|K_a)$ and $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L_a|K_a) \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L|K)$. For the first inclusion, let $A \in \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L|K)$ via the nondeterministic transducer M, which outputs symbols from $\Sigma \cup \{\varepsilon\}$. M can easily be transformed into a transducer M' which proves that $A \in \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(L_a|K_a)$: M' works like M, but whenever M outputs ε, M' outputs a. For the second inclusion, let again M be the nondeterministic transducer which proves $A \in \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L_a|K_a)$. Observe that letters a in the leaf string of M on an input x have no influence on whether x belongs to A or not. Hence, we can transform M into a machine M' that outputs ε whenever M outputs a. Hence, $A \in \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L|K)$. □

We obtain the following BCSV-like theorem for the e-model.

Theorem 7.9 *Let $(L|K) \subseteq \Sigma_1^*$ and $(L'|K') \subseteq \Sigma_2^*$. Then the following statements are equivalent:*

1. $(L|K) \leq^{\mathrm{pte}}_{\mathrm{m}} (L'|K')$.
2. *For all oracles O it holds that* $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L|K)^O \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L'|K')^O$.

Proof. The following equivalences hold:

$$\begin{aligned}
(L|K) \leq^{\mathrm{pte}}_{\mathrm{m}} (L'|K') \quad &\overset{\text{I.}}{\Leftrightarrow} \quad (L_a|K_a) \leq^{\mathrm{plt}}_{\mathrm{m}} (L'_a|K'_a), \\
&\overset{\text{II.}}{\Leftrightarrow} \quad \forall O, \mathrm{Leaf}^{\mathrm{p}O}_{\mathrm{b}}(L_a|K_a) \subseteq \mathrm{Leaf}^{\mathrm{p}O}_{\mathrm{b}}(L'_a|K'_a), \\
&\overset{\text{III.}}{\Leftrightarrow} \quad \forall O, \mathrm{Leaf}^{\mathrm{p}O}_{\varepsilon}(L|K) \subseteq \mathrm{Leaf}^{\mathrm{p}O}_{\varepsilon}(L'|K').
\end{aligned}$$

Note that I. holds because of Proposition 7.7, II. holds because of the BCSV-theorem [BCS92, Ver93], and III. holds because Lemma 7.8 is relativisable. □

7.1.2 A Connection to the Straubing-Thérien Hierarchy

Brzozowski and Cohen [CB71, Brz76] introduced the dot-depth hierarchy which we discussed in the last chapter. Straubing and Thérien [Str81, Thé81, Str85] introduced a modification that is more appropriate for the algebraic theory of languages but still covers the important aspects of the dot-depth hierarchy. This hierarchy is called the Straubing-Thérien hierarchy (STH).

Perrin and Pin [PP86] proved a logical characterisation of the STH. We use this characterisation as a definition since it uses a natural logic on words and it shows nice parallels to the definition of the polynomial-time hierarchy.

Formulas of the first-order logic FO[<] consist of first-order quantifiers, Boolean operators, the binary relation symbol $<$, and unary relation symbols π_a for each letter a.

A sentence ϕ is satisfied by a word w if ϕ evaluates to true where variables are interpreted as positions in w and $\pi_a x$ is interpreted as "letter a appears at position x in w".

A language B is FO[$<$] definable if there exists a sentence ϕ such that for all words w, $w \in L$ if and only if ϕ is satisfied by w.

A Σ_k^{FO}-sentence (resp., Π_k^{FO}-sentence) is a sentence of FO[$<$] that is in prenex normal form, starts with an existential (resp., universal) quantifier, and has at most $k-1$ quantifier alternations.

A language belongs to the class Σ_k^{FO} (resp., Π_k^{FO}) of the STH if it can be defined by a Σ_k^{FO}-sentence (resp., Π_k^{FO}-sentence). Δ_k^{FO} denotes the intersection of Σ_k^{FO} and Π_k^{FO}.

The next theorem shows a connection between the STH and the PH via the e-model. As we discussed in the last chapter, a similar connection for the existing b- and u-models was proven by Hertrampf et al. [HLS$^+$93], Burtschick and Vollmer [BV98], and Borchert et al. [BLS$^+$05].

Theorem 7.10 *The following statements hold for all $k \geq 1$:*

1. $\mathrm{Leaf}_\varepsilon^p(\Sigma_k^{FO}) = \Sigma_k^P$
2. $\mathrm{Leaf}_\varepsilon^p(\Pi_k^{FO}) = \Pi_k^P$
3. $\mathrm{Leaf}_\varepsilon^p(\Delta_k^{FO}) = \Delta_k^P$

Proof. Let $B \subseteq \Sigma^*$ and choose a new letter $a \notin \Sigma$. We show: if $B \in \Sigma_k^{FO}$, then $B_a \in \Sigma_k^{FO}$. Let ϕ be a Σ_k^{FO}-sentence defining B. Assume $\phi = Q_1 i_1 Q_2 i_2 \cdots Q_n i_n \psi$ where the Q's are quantifiers, the i's are variables, and ψ is quantifier-free. Now replace the quantifiers $Q_n, \ldots, Q_1$ and the formulas α they range on:

$$\begin{aligned} \exists i\, \alpha \quad &\text{is replaced by} \quad \exists i\, (\neg \pi_a i \wedge \alpha) \\ \forall i\, \alpha \quad &\text{is replaced by} \quad \forall i\, (\pi_a i \vee \alpha) \end{aligned}$$

Denote the resulting formula by ϕ'. Observe that ϕ' defines B_a and that ϕ' can be converted to a Σ_k^{FO}-sentence. Hence $B_a \in \Sigma_k^{FO}$. The same argument shows (i) if $L \in \Pi_k^{FO}$, then $L_a \in \Pi_k^{FO}$ and (ii) if $L \in \Delta_k^{FO}$, then $L_a \in \Delta_k^{FO}$.

If we consider the u-model instead of the e-model, then the statements of the theorem are known [BV98, BSS99, BLS$^+$05]. By $\mathrm{Leaf}_u^p(B) \subseteq \mathrm{Leaf}_\varepsilon^p(B)$, it suffices to argue for the inclusions from left to right. Let $L \in \mathrm{Leaf}_\varepsilon^p(\Sigma_k^{FO})$, i.e., there exists $B \in \Sigma_k^{FO}$ such that $L \in \mathrm{Leaf}_\varepsilon^p(B)$. By Lemma 7.8, $L \in \mathrm{Leaf}_u^p(B_a)$ where a is a new letter. As argued above, $B_a \in \Sigma_k^{FO}$ and therefore, $L \in \mathrm{Leaf}_u^p(\Sigma_k^{FO}) \subseteq \Sigma_k^P$. The inclusions $\mathrm{Leaf}_\varepsilon^p(\Pi_k^{FO}) \subseteq \Pi_k^P$ and $\mathrm{Leaf}_\varepsilon^p(\Delta_k^{FO}) \subseteq \Delta_k^P$ follow analogously. □

7.2 Gap Theorems and Perfect Correspondences

In the last section we proved that the e-model connects the STH to the PH in a similar way as the b- and u-models connect the DDH to the PH. In the last chapter we introduced the notion of a tighter connection which we called *perfect correspondence*. We proved that via balanced leaf languages, the DDH and the PH do not correspond perfectly. Contrary to that, we were able to show that the lower levels of both hierarchies do in fact correspond via unbalanced leaf languages. We now transfer the notion of perfect correspondence to the e-model. Observe that the following definition is an analogue of Definitions 6.3 and 6.4.

Definition 7.11 *A class of regular languages $\mathcal{C}$ and a complexity class $\mathcal{D}$* perfectly correspond with respect to ε-leaf languages *if (restricted to regular languages) $\mathcal{C}$ is closed under pte-reducibility and* $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(\mathcal{C}) = \mathcal{D}$.

In the next sections, we will show that the e-model establishes perfect correspondences between the lower levels of the STH and the PH. We will in fact prove slightly stronger results which we interpret as *gap theorems*. Recall that in order to show that a class of regular languages $\mathcal{C}$ and a a complexity class $\mathcal{D}$ perfectly correspond with respect to ε-leaf languages, we have to prove the following amongst other things:

$$\begin{gathered}\text{Whenever we chose a regular language } L \text{ outside } \mathcal{C},\\ \text{then } \mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L) \text{ is not robustly contained in } \mathrm{Leaf}_\varepsilon^{\mathrm{p}}(\mathcal{C}).\end{gathered}$$

Assume that no matter how $L \in \mathrm{REG} - \mathcal{C}$ is chosen, we can prove that $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L) \supseteq \mathcal{D}'$ for some complexity class $\mathcal{D}'$ which is oracle separable from $\mathcal{D}$.

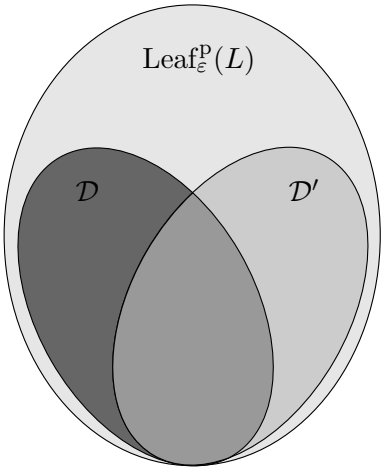


Figure 7.1: Complexity classes $\mathcal{D}$ and $\mathcal{D}'$ are incomparable relative to an oracle. Whenever $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L) \supset \mathcal{D}$, it follows that $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L) \supset \mathcal{D} \cup \mathcal{D}'$.

Then this is a *gap theorem* of leaf-language definability between $\mathcal{D}$ and $\mathcal{D}'$: Every ε-leaf-language definable class is either contained in $\mathcal{D}$ or it contains $\mathcal{D}'$. If such a class strictly contains $\mathcal{D}$, then it must also contain $\mathcal{D}'$. This situation is depicted in Figure 7.1.

7.3 Gap Theorems for NP, Δ_2^{P}, and Σ_2^{P}

In this section we use existing forbidden-pattern characterisations to obtain lower bounds for certain e-classes. From this we derive gap theorems for NP, Δ_2^{P}, and Σ_2^{P}. A summary of these results can be found in Table 7.1 at the end of this chapter.

Pin and Weil [PW97] proved the following characterisation of level Σ_1^{FO} of the STH.

Proposition 7.12 ([PW97]) *The following are equivalent for any language A.*

1. $A \in \Sigma_1^{\mathrm{FO}}$
2. $\forall v, w \in \Sigma^*[v \preceq w \Rightarrow c_A(v) \leq c_A(w)]$
3. $\forall v, w \in \Sigma^* \forall a \in \Sigma[c_A(vw) \leq c_A(vaw)]$

Using this characterisation, we can apply the same technique as [Bor95] and [BKS99] and obtain the following lower bounds for the e-class of languages outside Σ_1^{FO}. Since this requires only straightforward modifications to the proofs of Borchert and Borchert, Kuske, and Stephan, we omit the proof here.

Theorem 7.13 *Let A be an arbitrary language.*

1. *If $A \notin \Sigma_1^{\mathrm{FO}}$, then* coUP $\subseteq \mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(A)$.
2. *If $A \in \mathrm{REG} - \Sigma_1^{\mathrm{FO}}$, then* $\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(A)$ *contains at least one of the following classes:* coNP, co1NP, $\mathrm{MOD}_p\mathrm{P}$ *for a prime p.*
3. *If $A \in \mathrm{SF} - \Sigma_1^{\mathrm{FO}}$, then* $\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(A)$ *contains at least one of the following classes:* coNP, co1NP.

In combination with Theorem 7.10 we obtain a gap theorem for NP.

Corollary 7.14 *Let B be a nontrivial language.*

1. *The e-class of B either is contained in* NP, *or contains* coUP.
2. *If $B \in$* REG, *then the e-class of B either is contained in* NP, *or contains at least one of the following classes:* coNP, co1NP, $\mathrm{MOD}_p\mathrm{P}$ *for a prime p.*
3. *If $B \in$* SF, *then the e-class of B either is contained in* NP, *or contains at least one of the following classes:* coNP, co1NP.

Proof. Follows from Theorems 7.10 and 7.13. □

Now we can prove general lower bounds for e-classes. In particular, no complexity class below UP is definable with this concept.

Corollary 7.15 *Let A be a nontrivial language.*

1. $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A)$ *contains at least one of the following classes:* UP, coUP.
2. *If* $A \in \mathrm{REG}$, *then* $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A)$ *contains at least one of the following classes:* NP, coNP, $\mathrm{MOD}_p\mathrm{P}$ *for a prime* p.
3. *If* $A \in \mathrm{SF}$, *then* $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A)$ *contains at least one of the following classes:* NP, coNP.

Proof. Assume $\varepsilon \notin A$. By assumption, there exists a nonempty $w \in A$. Any UP machine can be modified such that rejecting paths output ε and accepting paths output w. This shows $\mathrm{UP} \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A)$. Analogously, the assumption $\varepsilon \in A$ implies $\mathrm{coUP} \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A)$.

If A also belongs to REG, then $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A)$ or $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(\overline{A})$ contains at least one of the following classes: coNP, co1NP, $\mathrm{MOD}_p\mathrm{P}$ for a prime p. Hence $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A)$ contains at least one of the following classes: NP, coNP, $\mathrm{MOD}_p\mathrm{P}$ for a prime p. If A even belongs to SF, then the same argument shows that $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A)$ contains at least one of the following classes: NP, coNP. □

Under reasonable assumptions there is no regular A such that A's e-class lies strictly between coNP and 1NP. By symmetry, the same holds for NP and co1NP.

Corollary 7.16 *Let $A \in \mathrm{REG}$ be a nontrivial language. Assume that* $\mathrm{NP} \not\subseteq \mathrm{1NP}$ *and* $\mathrm{MOD}_p\mathrm{P} \not\subseteq \mathrm{1NP}$ *for all primes p. Then the following implication holds.*

$$\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A) \subsetneq \mathrm{1NP} \Rightarrow \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A) \subseteq \mathrm{coNP}.$$

Proof. If $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A) \not\subseteq \mathrm{coNP}$, then $A \notin \Pi_1^{\mathrm{FO}}$. By Theorem 7.13, $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(A)$ contains at least one of the following classes: NP, 1NP, $\mathrm{MOD}_p\mathrm{P}$ for a prime p. □

Starting with a forbidden-pattern characterisation for Σ_2^{FO} [PW97] (Figure 7.2) we prove a lower bound for the e-class of Σ_2^{FO}. Again, this yields a gap theorem, this time for Σ_2^{P} (Corollary 7.19). Schmitz [Sch01] announces a similar result for the u-class of Σ_2^{FO}, but does not give a proof. The proof we give here uses a similar technique as [Bor95] and [BKS99]. In our proof, we will utilise a recent result by Spakowski and Tripathi [ST07]. This result concerns the unambiguous alternation hierarchy, which was introduced by Niedermeier and Rossmanith [NR98]. We give an alternative characterisation of the unambiguous alternation hierarchy which is attributed to unpublished work of Hemaspaandra [NR98].

For any complexity class $\mathcal{C}$, define $\exists^{\mathrm{u}}\cdot\mathcal{C}$ as the class of languages L such that there exist a polynomial p and $L' \in \mathcal{C}$ such that for all x,

$$\begin{aligned} x \in L &\Rightarrow \text{there exists exactly one } y \in \Sigma^{p(|x|)} \text{ such that } (x,y) \in L' \\ x \notin L &\Rightarrow \text{there exists no } y \in \Sigma^{p(|x|)} \text{ such that } (x,y) \in L'. \end{aligned}$$

Analogously, $\forall^{\mathrm{u}}\cdot\mathcal{C}$ is the class of languages L such that there exist a polynomial p and $L' \in \mathcal{C}$ such that for all x,

$$\begin{aligned} x \in L &\Rightarrow \text{for all } y \in \Sigma^{p(|x|)},\ (x,y) \in L' \\ x \notin L &\Rightarrow \text{there exists exactly one } y \in \Sigma^{p(|x|)} \text{ such that } (x,y) \notin L'. \end{aligned}$$

The following definition is attributed to unpublished work of Hemaspaandra [NR98].

Definition 7.17 *The following are the levels of the unambiguous alternation hierarchy.*

$$\begin{aligned} \mathrm{AU}\Sigma_0^{\mathrm{P}} = \mathrm{AU}\Pi_0^{\mathrm{P}} &=_{\mathrm{def}} \mathrm{P} \\ \mathrm{AU}\Sigma_{k+1}^{\mathrm{P}} &=_{\mathrm{def}} \exists^{\mathrm{u}}\cdot\mathrm{AU}\Pi_k^{\mathrm{P}} \quad \textit{for } k \geq 0 \\ \mathrm{AU}\Pi_{k+1}^{\mathrm{P}} &=_{\mathrm{def}} \forall^{\mathrm{u}}\cdot\mathrm{AU}\Sigma_k^{\mathrm{P}} \quad \textit{for } k \geq 0. \end{aligned}$$

Spakowski and Tripathi [ST07] construct an oracle relative to which for every $n \geq 1$, level n of the unambiguous alternation hierarchy is not contained in Π_n^{P}.

Theorem 7.18 *If $A \in \mathrm{REG} - \Sigma_2^{\mathrm{FO}}$, then $\mathrm{AU}\Pi_2^{\mathrm{P}} \subseteq \mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(A)$.*

Proof. Pin and Weil [PW97] proved the following forbidden pattern characterisation of Σ_2^{FO}: A regular language belongs to Σ_2^{FO} if and only if the transition graph of its minimal automaton does not contain the subgraph shown in Figure 7.2. So by assumption, A's minimal automaton contains this graph.

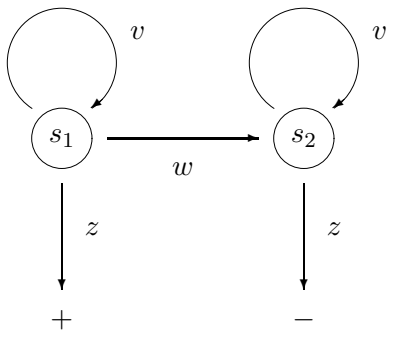


Figure 7.2: Forbidden pattern for Σ_2^{FO} where $w \preceq v$.

Let $L \in \mathrm{AU}\Pi_2^{\mathrm{P}}$, i.e., there exist $B \in \mathrm{P}$ and polynomials p and q such that for all x,

$$\begin{aligned} x \in L &\Rightarrow \forall y \in \Sigma^{p(|x|)}, \exists! z \in \Sigma^{q(|x|)}[(x,y,z) \in B], \\ x \notin L &\Rightarrow \text{there exists } y \in \Sigma^{p(|x|)} \text{ such that the following holds:} \\ &\quad \text{(i)} \ \forall z \in \Sigma^{q(|x|)}[(x,y,z) \notin B], \\ &\quad \text{(ii)} \ \forall u \in \Sigma^{p(|x|)} - \{y\}, \exists! z \in \Sigma^{q(|x|)}[(x,u,z) \in B]. \end{aligned}$$

We describe a nondeterministic machine M on input x: First, M nondeterministically guesses $y \in \Sigma^{p(|x|)}$. Now M splits into $|v|$ paths which we associate with the letters of v. Consider the first occurrence of w as a subword of v. The paths that are associated with the positions involved in this occurrence output the respective letters of v and stop. On all other paths (i.e., those which are not involved in the first occurrence of w in v) the computation is continued as follows: Assume we are on a path that is associated with letter c in v. M nondeterministically guesses $z \in \Sigma^{q(|x|)}$. If $(x,y,z) \in B$, then output ε and stop. Otherwise, output c and stop.

In order to determine the leaf string $\beta_M(x)$ we first consider certain factors of this string. More precisely, let β_u be the leaf string that is produced by the paths that guess $y = u$. Note that $\beta_M(x) = \beta_0\beta_1 \cdots \beta_{2^{p(|x|)}}$.

Assume $u \in \Sigma^{p(|x|)}$ such that $\exists! z \in \Sigma^{q(|x|)}[(x,u,z) \in B]$. Consider the path where M guesses $y = u$. In the next steps, M splits into $|v|$ paths associated with the letters of v. The paths involved in the first occurrence of w in v will output the respective letters from v. Each remaining path continues the computation. By assumption, there exists exactly one z such that $[(x,y,z) \in B]$. The path guessing that z will output the respective letter in v, while all other paths will output ε. Therefore, $\beta_u = v$.

Assume $u \in \Sigma^{p(|x|)}$ such that $\forall z \in \Sigma^{q(|x|)}[(x,u,z) \notin B]$. Consider the path where M guesses $y = u$. Again M splits into $|v|$ paths. The paths involved in the occurrence of w will output the respective letters from v. However, now there is no z such that $[(x,y,z) \in B]$ and therefore, all remaining paths output ε. It follows that $\beta_u = w$.

Now let us consider $\beta_M(x)$. If $x \in L$, then for all u, $\exists! z \in \Sigma^{q(|x|)}[(x,u,z) \in B]$. Therefore, all u, $\beta_u = v$ and it follows that $\beta_M(x) \in v^*$. Otherwise, $x \notin L$. So there exists $y \in \Sigma^{p(|x|)}$ such that (i) $\forall z \in \Sigma^{q(|x|)}[(x,y,z) \notin B]$ and (ii) for all $u \neq y$, $\exists! z \in \Sigma^{q(|x|)}[(x,u,z) \in B]$. Therefore, (i) $\beta_y = w$ and (ii) for all $u \neq y$, $\beta_u = v$. It follows that $\beta_M(x) \in v^*wv^*$. So we obtained:

$$\begin{aligned} x \in L &\Rightarrow \beta_M(x) \in v^* \\ x \notin L &\Rightarrow \beta_M(x) \in v^*wv^* \end{aligned}$$

Let y be a word leading from the initial state to s_1 in the minimal automaton of A. Let M' be the modification of M that on the left additionally outputs y and on the right additionally outputs z. Hence, $x \in L$ if and only if $\beta_{M'}(x) \in A$. This shows $L \in \mathrm{Leaf}_{\varepsilon}^{\mathrm{P}}(A)$. □

Corollary 7.19 *Let B be a nontrivial, regular language. The e-class of B either is contained in Σ_2^{P}, or contains $\mathrm{AU}\Pi_2^{\mathrm{P}}$.*

Proof. Follows from Theorems 7.10 and 7.18. □

In addition, Theorem 7.18 also yields a lower bound for the e-class of Δ_2^{FO}:

Corollary 7.20 *If $A \in \mathrm{REG} - (\Sigma_2^{\mathrm{FO}} \cap \Pi_2^{\mathrm{FO}})$, then $\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(A)$ contains at least one of the following classes: $\mathrm{AU}\Pi_2^{\mathrm{P}}$, $\mathrm{AU}\Sigma_2^{\mathrm{P}}$.*

Proof. By assumption, A or $\overline{A}$ is outside Σ_2^{FO}. By Theorem 7.18, $\mathrm{AU}\Pi_2^{\mathrm{P}} \subseteq \mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(A)$ or $\mathrm{AU}\Pi_2^{\mathrm{P}} \subseteq \mathrm{coLeaf}_{\varepsilon}^{\mathrm{p}}(A)$. The latter is equivalent to $\mathrm{AU}\Sigma_2^{\mathrm{P}} \subseteq \mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(A)$. □

The following is a gap theorem for Δ_2^{P}. It holds for both the u- and the e-model.

Corollary 7.21 *Let B be a nontrivial, regular language.*

1. *The e-class of B either is contained in Δ_2^{P}, or contains at least one of the following classes: $\mathrm{AU}\Sigma_2^{\mathrm{P}}$, $\mathrm{AU}\Pi_2^{\mathrm{P}}$.*

2. *The u-class of B either is contained in Δ_2^{P}, or contains at least one of the following classes: $\mathrm{AU}\Sigma_2^{\mathrm{P}}$, $\mathrm{AU}\Pi_2^{\mathrm{P}}$.*

Proof. The first statement is an immediate consequence of Theorem 7.10 and Corollary 7.20. For the second statement, let $\mathcal{B}_{3/2}$ denote level 3/2 of the dot-depth hierarchy [CB71, PW97]. If $A \in \mathrm{REG} - (\mathcal{B}_{3/2} \cap \mathrm{co}\mathcal{B}_{3/2})$, then $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(A)$ contains at least one of the following classes: $\mathrm{AU}\Sigma_2^{\mathrm{P}}$, $\mathrm{AU}\Pi_2^{\mathrm{P}}$ [Sch01]. Borchert et al. [BLS$^+$05] mention that $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(\mathcal{B}_{3/2} \cap \mathrm{co}\mathcal{B}_{3/2}) = \Delta_2^{\mathrm{P}}$ can be obtained by an extension of their method. □

7.4 A Characterisation of 1NP

In this section we analyse the class 1NP in detail and prove a gap theorem for this class. This case is more challenging since we cannot utilise an existing forbidden-pattern characterisation. With Theorem 7.31 we give such a characterisation for the class of languages corresponding to 1NP. Additionally, this theorem shows that with this class we have in fact identified *all* languages whose e-class is robustly contained in 1NP. This lets us derive a gap theorem for 1NP.

For a given language L, we define the following conditions:

P1: There exist words $u \in L$, $v \notin L$, and $w \in L$ such that $u \preceq v \preceq w$.

P2: There exist $k \geq 2$ and nonempty words $u, v, w \in L$ such that $\{u, v\} \preceq_k w$ and $(\forall x)[x \prec u \text{ or } x \prec v \Rightarrow x \notin L]$.[1]

We interpret the patterns P1 and P2 as forbidden patterns and define a class of languages U which neither fulfill P1 nor P2:

$$\mathrm{U} =_{\mathrm{def}} \{L : \text{P1 and P2 fail for } L\}$$

We will see later on that U is in fact a class of regular languages which precisely characterises the class 1NP in the e-model of leaf languages. The next two lemmas show that the e-class of a language which fulfills P1 or P2 is quite powerful.

Lemma 7.22 *Let $L \subseteq \Sigma^*$ such that L satisfies P1. Then* $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L) \supseteq \mathrm{UP}\dot{\vee}\mathrm{coUP}$.

Proof. Let $L \subseteq \Sigma^*$ such that there exist words $u \in L$, $v \notin L$, and $w \in L$ such that $u \preceq v \preceq w$, i.e. L satisfies pattern P1. Furthermore, let $A \in \mathrm{UP}\dot{\vee}\mathrm{coUP}$. Hence $A = B \cup C$ where $B \in \mathrm{UP}$, $C \in \mathrm{coUP}$, and $B \subseteq \overline{C}$. Let M_B be the UP-machine accepting B, and let M_C be the coUP-machine accepting C. Observe that whenever M_B on an input x produces an accepting path (and thus accepts the input in an UP-sense), M_C also produces an accepting path and hence rejects (in an coUP-sense).

In order to prove $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L) \supseteq \mathrm{UP}\dot{\vee}\mathrm{coUP}$, we show how to construct a nondeterministic polynomial-time Turing machine M such that for all x:

$$\begin{aligned} x \notin A &\implies \beta_M(x) = v \\ x \in B &\implies \beta_M(x) = w \\ x \in C &\implies \beta_M(x) = u \end{aligned}$$

Since $u \preceq v \preceq w = w_1 \dots w_k$, we can mark the letters of one fixed occurrence of u in w, we do the same with one fixed occurrence of v in w. Let $I_u \subsetneq \{1, \dots, k\}$ be the indices of letters in w that are marked to belong to u, and let $I_v \subsetneq \{1, \dots, k\}$ be the indices of letters in w that are marked to belong to v. Note that $\#I_v = |v|$, $\#I_u = |u|$, and $I_u \subsetneq I_v$.

For $1 \leq i \leq k$, we construct Turing machines M_i as follows:

- If $i \in I_u$, M_i develops only one path and outputs w_i on this path.
- If $i \in I_v - I_u$, M_i simulates machine M_C on the same input. On every rejecting path of M_C, M_i outputs ϵ, if an accepting path exists, this path outputs w_i.
- If $i \notin I_v$, M_i simulates machine M_B on the same input. On every rejecting path of M_B, M_i outputs ϵ, if an accepting path exists, this path outputs w_i.

[1] Note that in P2, the words u and v can be the same.

Turing machine M is constructed as follows: On input x, M branches into k nondeterministic paths. On path i, M then simulates M_i on input x. Notice that M can only produce leaf strings from $\{u, v, w\}$. It is easy to see that M satisfies the above condition: It holds that $x \in A \Leftrightarrow \beta_M(x) \in L$, and hence $A \in \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L)$. □

Lemma 7.23 *Let $L \subseteq \Sigma^*$ such that L satisfies P2. Then* $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L) \supseteq \mathrm{UP} \vee \mathrm{UP}$.

Proof. Let $L \subseteq \Sigma^*$ such that there exists $k \geq 2$ and nonempty words $u, v, w = w_1 \dots w_l \in L$ such that $\{u, v\} \preceq_k w$ and $\forall x\big((x \prec u \text{ or } x \prec v) \Rightarrow x \notin L\big)$. If $u \preceq_0 w$ we set $u := v$, if $v \preceq_0 w$ we set $v := u$. We obtain $\{u, v\} \preceq_k w$ and $u \preceq w, v \preceq w$. Observe that since L satisfies P2, the empty word ε cannot be in L, which has to have nonempty minimal words. Furthermore, let $A \in \mathrm{UP} \vee \mathrm{UP}$. Hence $A = B \cup C$ where $B \in \mathrm{UP}$ and $C \in \mathrm{UP}$. Let M_B be the UP-machine accepting B, and let M_C be the UP-machine accepting C.

In order to prove $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L) \supseteq \mathrm{UP} \vee \mathrm{UP}$, we show how to construct a nondeterministic polynomial-time Turing machine M such that the following holds for all x:

$$\begin{aligned} x \notin A &\implies \beta_M(x) \prec u \\ x \in B - C &\implies \beta_M(x) = u \\ x \in C - B &\implies \beta_M(x) = v \\ x \in B \cap C &\implies \beta_M(x) = w \end{aligned}$$

Observe that no proper subword of u can be in L, since P2 requests that u and v are minimal words in L. Consequently, constructing a machine M as above yields $x \in A \Leftrightarrow \beta_M(x) \in L$ and hence $A \in \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L)$.

Since $u \preceq w$ and $v \preceq w$, we can mark the letters of one fixed occurrence of u in w, we do the same with one fixed occurrence of v in w.[2] Let $I_u \subsetneq \{1, \dots, l\}$ be the indices of letters in w that are marked to belong to u, and let $I_v \subsetneq \{1, \dots, l\}$ be the indices of letters in w that are marked to belong to v. Observe that $I_u \neq I_v$ and hence $\#(I_u \cap I_v) < \min(|u|, |v|)$.

For $1 \leq i \leq l$, we construct Turing machines M_i as follows:

I. If $i \in I_u \cap I_v$, M_i develops only one path and outputs w_i on this path.

II. If $i \in I_u - I_v$, M_i simulates machine M_B on the same input. On every rejecting path of M_B, M_i outputs ϵ, if an accepting path exists, this path outputs w_i.

III. If $i \in I_v - I_u$, M_i simulates machine M_C on the same input. On every rejecting path of M_C, M_i outputs ϵ, if an accepting path exists, this path outputs w_i.

IV. If $i \notin I_u \cup I_v$, M_i outputs w_i if and only if M_B and M_C (running on the same input as M_i) produce an accepting path, otherwise M_i outputs ε.

[2] If $u = v$, we fix two different occurrences of u in w.

Turing machine M is constructed as follows: On input x, M branches into l nondeterministic paths. On paths i for $1 \leq i \leq l$, M then simulates M_i on input x.

We consider the four different possibilities for the behavior of M on an input x. We do this by analysing the behavior of the machines M consists of. Notice that depending on u, v, w, there might not be any machines of types I and IV.

Case 1: $x \notin A$. Hence, UP-machines M_B and M_C do not accept x, i.e. both produce only rejecting paths. Clearly, all machines of type II, III, and IV only output empty words. If $I_u \cap I_v \neq \emptyset$, precisely those letters of w are output that belong simultaneously to the marked occurrence of u and to the marked occurrence of v. Let $i_1 < i_2 < \ldots < i_{\#(I_u \cap I_v)}$ be the elements of $I_u \cap I_v$, then $\beta_M(x) = w_{i_1} \ldots w_{i_{\#(I_u \cap I_v)}}$. As we then have $\beta_M(x) \prec u$, we can conclude that $\beta_M(x) \notin L$, since no subword of u is element of L. If $I_u \cap I_v = \emptyset$, $\beta_M(x) = \varepsilon$. Again, we conclude $\beta_M(x) \notin L$ since $\varepsilon \notin L$.

Case 2: $x \in B - C$, i.e. M_B produces an accepting path on input x whereas M_C produces only rejecting paths. This means all machines of type III and IV output empty words. Recall that letters belonging to u and v simultaneously are created by machines of type I regardless of the input. So we obtain $\beta_M(x) = u$ and $\beta_M(x) \in L$.

Case 3: $x \in C - B$. Analogous to case 2.

Case 4: $x \in B \cap C$, i.e. M_B and M_C both produce an accepting path on input x. If $I_u \cup I_v = \{1, \ldots, l\}$, it is clear that $\beta_M(x) = w \in L$. $I_u \cup I_v \subsetneq \{1, \ldots, l\}$, the missing letters of w are produced by the machines of type IV. We again obtain $\beta_M(x) = w \in L$.

From the above case differentiation, we obtain $x \in A \Leftrightarrow \beta_M(x) \in L$ which proves $A \in \mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(L)$. □

The next lemma presents languages that define the classes 1NP and UP$\dot{\vee}$coUP in terms of leaf languages.

Lemma 7.24 *1.* $\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(1|(\varepsilon \cup 111^*)) = 1\mathrm{NP}$.
2. $\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}((\varepsilon \cup 12)|2) = \mathrm{UP}\dot{\vee}\mathrm{coUP}$.
3. $\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}((1 \cup 2 \cup 12)|\varepsilon) = \mathrm{UP} \vee \mathrm{UP}$.

Proof. 1. For $\supseteq$, simply modify the 1NP-machine such that every accepting path outputs 1 and every rejecting path outputs ε. For $\subseteq$, modify the ε-machine such that every path that outputs ε then rejects, and every path that outputs 1 then accepts.

2. $\supseteq$: Let $A \in \mathrm{UP}\dot{\vee}\mathrm{coUP}$, that means $A = B \cup \overline{C}$ where $B, C \in \mathrm{UP}$ and $B \subseteq C$. Let M_B, M_C be the UP-machines that prove $B, C \in \mathrm{UP}$. As $B \subseteq C$, it holds for all inputs x that whenever M_B on input x produces an accepting path M_C on input x also produces an accepting path. We now construct a nondeterministic Turing machine M as follows:

On input x, M first branches nondeterministically. On the left path, M simulates M_B on input x, on the right path, it simulates M_C on input x. All rejecting paths of these simulations output ε, the accepting path of M_B (if existent) outputs 1, the accepting path of M_C (if existent) outputs 2. It is easy to see that the following now holds:

$$\begin{aligned} \forall x, \quad & \beta_M(x) \in \{\varepsilon, 2, 12\}, \\ x \in B \quad \Rightarrow \quad & \beta_M(x) = 12, \\ x \in \overline{C} \quad \Rightarrow \quad & \beta_M(x) = \varepsilon, \\ x \notin A \quad \Rightarrow \quad & \beta_M(x) = 2. \end{aligned}$$

This proves $A \in \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}((\varepsilon \cup 12), 2)$.

$\subseteq$: Let $A \in \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}((\varepsilon \cup 12)|2)$ via the nondeterministic ε-machine M. Observe that $A = B \cup \overline{C}$, where $B =_{\mathrm{def}} \{x \,|\, \beta_M(x) = 12\}$ and $C =_{\mathrm{def}} \{x \,|\, 2 \preceq \beta_M(x)\}$. Clearly, $B, C \in \mathrm{UP}$ and $B \subseteq C$. Hence, $A \in \mathrm{UP} \dot{\vee} \mathrm{coUP}$.

3. Analogous. □

In order to show that for any language L, fulfillment of P1 suffices for the class $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L)$ to be not robustly contained in 1NP, we first prove that languages characterising $\mathrm{UP} \dot{\vee} \mathrm{coUP}$ cannot be pte-reduced to languages characterising 1NP.

Lemma 7.25 $((\varepsilon \cup 12)|2) \not\leq^{\mathrm{pte}}_{\mathrm{m}} (1|(\varepsilon \cup 111^*))$.

Proof. We assume that $(L|K) \leq^{\mathrm{pte}}_{\mathrm{m}} (L'|K')$, where $(L|K) = ((\varepsilon \cup 12), 2)$ and $(L'|K') = (1, (\varepsilon \cup 111^*))$. Due to Proposition 7.7, this is equivalent to $(L_0|K_0) \leq^{\mathrm{plt}}_{\mathrm{m}} (L'_0, K'_0)$. Recall that $(L_0|K_0) =_{\mathrm{def}} ((0^* \cup 0^*10^*20^*)|0^*20^*)$ and $(L'_0|K'_0) =_{\mathrm{def}} (0^*10^*|(0^* \cup 0^*10^*1(0 \cup 1)^*))$. Assume $(L_0|K_0) \leq^{\mathrm{plt}}_{\mathrm{m}} (L'_0|K'_0)$ holds via plt-reduction f.

This means there exist functions g, h which are computable in time $c \cdot \log^k$ fur suitable $c, k \geq 0$ such that $f(x) = g(x,1)g(x,2)\ldots g(x,h(x))$ and the following holds:

$$\begin{aligned} x \in L_0 \quad \Rightarrow \quad & f(x) = g(x,1)g(x,2)\ldots g(x,h(x)) \in L'_0, \\ x \in K_0 \quad \Rightarrow \quad & f(x) = g(x,1)g(x,2)\ldots g(x,h(x)) \in K'_0. \end{aligned}$$

Let M_g be the deterministic polylog-time machine that computes g within the above time bound. We choose n sufficiently large such that $n > 2 \cdot c \cdot \log^k(n + c\log^k n) + 2$ and consider the input $w =_{\mathrm{def}} 0^n$. Since $w \in L_0$, there exists precisely one $1 \leq i \leq h(w)$ such that $g(w,i) = 1$. Hence, M_g on input (w,i) outputs 1, while it outputs 0 on input (w,k) for all other k. Since $n > 2 \cdot c \cdot \log^k(n + c\log^k n) + 2$, M_g on input (w,i) cannot have queried all positions in w. Let j be a position that is not queried by M_g on input (w,i). We then set $v =_{\mathrm{def}} 0^{j-1}20^{n-j}$. Notice that M_g still outputs 1 when ran on input (v,i) since w and v only differ on a position not queried by M_g on input (w,i). As $v \in K_0$, f has to output a

word from K_0'. Since $g(v,i)=1$, there has to be another i' such that $g(v,i')=1$. Due to $n > 2\cdot c\cdot \log^k(n+c\log^k n)+2$, we can easily find a position $j'<j$ such that M_g neither queries j' on input (v,i) nor on input (v,i'). Let $u =_{\text{def}} 0^{j'-1}10^{j-j'+1}20^{n-j}$. As we still have $g(u,i)=g(u,i')=1$, $f(u)\in K_0'$ although $u\in L_0$. By this contradiction, we have shown that no such f can exist. □

Lemma 7.26 *There exists an oracle O such that* $(\mathrm{UP}\dot\vee\mathrm{coUP})^O \not\subseteq \mathrm{1NP}^O$.

Proof. This follows directly from $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}((\varepsilon\cup 12)|2)=\mathrm{UP}\dot\vee\mathrm{coUP}$, $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(1|(\varepsilon\cup 111^*))=\mathrm{1NP}$ (Lemma 7.24), $((\varepsilon\cup 12)|2)\not\leq^{\mathrm{pte}}_{\mathrm{m}}(1|(\varepsilon\cup 111^*))$ (Lemma 7.25) and Theorem 7.9. □

Similarly to the above note, we now prove that languages characterising $\mathrm{UP}\vee\mathrm{UP}$ cannot be pte-reduced to languages characterising 1NP. This is a step towards showing that for any language L, fulfillment of P2 suffices for the class $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L)$ to be not robustly contained in 1NP.

Lemma 7.27 $((1\cup 2\cup 12)|\varepsilon)\not\leq^{\mathrm{pte}}_{\mathrm{m}}(1|(\varepsilon\cup 111^*))$.

Proof. We assume that $(L|K)\leq^{\mathrm{pte}}_{\mathrm{m}}(L'|K')$ where $(L|K)=((1\cup 2\cup 12)|\varepsilon)$ and $(L'|K')=(1|(\varepsilon\cup 111^*))$. Due to Proposition 7.7, this is equivalent to $(L_0|K_0)\leq^{\mathrm{plt}}_{\mathrm{m}}(L_0'|K_0')$. Recall that $(L_0|K_0)=_{\text{def}}((0^*10^*\cup 0^*10^*20^*\cup 0^*20^*)|0^*)$ and $(L_0'|K_0')=_{\text{def}}(0^*10^*|(0^*\cup 0^*10^*1(0\cup 1)^*))$. Assume $(L_0|K_0)\leq^{\mathrm{plt}}_{\mathrm{m}}(L_0'|K_0')$ holds via plt-reduction f.

This means there exist functions g,h which are computable in time $c\cdot\log^k$ for suitable $c,k\geq 0$ such that $f(x)=g(x,1)g(x,2)\dots g(x,h(x))$ and the following holds:

$$\begin{aligned} x\in L_0 &\Rightarrow f(x)=g(x,1)g(x,2)\dots g(x,h(x))\in L_0',\\ x\in K_0 &\Rightarrow f(x)=g(x,1)g(x,2)\dots g(x,h(x))\in K_0'. \end{aligned}$$

Let M_g be the deterministic polylog-time machine that computes g within the above time bound. We choose n sufficiently large such that $\frac{n}{2} > 2\cdot c\cdot\log^k(2n+c\log^k 2n)$ and consider words $x_i,y_i,z_{i,j}\in\{0,1,2\}^{2n}$: For $i,j\in\{1,\dots,n\}$, we define $x_i =_{\text{def}} 0^{i-1}10^{n-i}0^n$, $y_i =_{\text{def}} 0^n0^{i-1}20^{n-i}$, and $z_{i,j} =_{\text{def}} 0^{i-1}10^{n-i}0^{j-1}20^{n-j}$. Observe that for $i,j\in\{1,\dots,n\}$, x_i,y_i, and $z_{i,j}$ are all in L_0 and hence $f(x_i),f(y_i)$, and $f(z_{i,j})$ are all in L_0'. Therefore, we have

$$\forall a\in\{x,y\}\forall i\in\{1,\dots,n\}\exists! j : g(a_i,j)=1.$$

For $1\leq i\leq n$, we define

$$\begin{aligned} d(i) &=_{\text{def}} l, \text{ where } g(x_i,l)=1,\\ B(i) &=_{\text{def}} \{l\in\{1,\dots,2n\} \mid M_g \text{ on input } (x_i,d(i)) \text{ queries position } l \text{ in } x_i\}\\ e(i) &=_{\text{def}} l, \text{ where } g(y_i,l)=1,\\ C(i) &=_{\text{def}} \{l\in\{1,\dots,2n\} \mid M_g \text{ on input } (y_i,e(i)) \text{ queries position } l \text{ in } y_i\} \end{aligned}$$

Claim: There exist $i, j \in \{1, \ldots, n\}$ such that $i \notin C(j)$ and $n + j \notin B(i)$.

Proof of the claim. Assuming that our claim is wrong, we conclude that for all $(i, j) \in \{1, \ldots, n\}^2$, it holds that $i \in C(j)$ or $n+j \in B(i)$. Without loss of generality, we can assume that $i \in C(j)$ holds for at least half of all (i, j), i.e. for at least $\frac{n^2}{2}$ tuples.[3] Observe that there now exists $1 \leq j \leq n$ such that among these $\frac{n^2}{2}$ tuples, there are tuples $(i_1, j), (i_2, j), \ldots, (i_{n/2}, j)$ such that $i_1 < i_2 < \ldots < i_{n/2}$. Hence, it holds that $i_1 \in C(j), i_2 \in C(j), \ldots i_{n/2} \in C(j)$. This in turn implies that M_g on input $(y_j, e(j))$ queries at least $\frac{n}{2}$ positions in y_j. Since we have chosen n sufficiently large such that $\frac{n}{2} > 2 \cdot c \cdot \log^k(2n + c \log^k 2n)$, M_g cannot query all these positions. So we have contradicted our assumption and thus proven the claim. $\diamond$

By this, we know that there exist $i, j \in \{1, \ldots, n\}$ such that $i \notin C(j)$ and $n + j \notin B(i)$. Using a standard technique, we can show that $d(i) \neq e(j)$.

Let us assume for a moment that $d(i) = e(j)$. This means that on input $(x_i, d(i))$, M_g does not query position $n+j$ in x_i, and on input $(y_j, d(i))$, M_g does not query position i in y_j. Recall that x_i and y_i only differ on positions i and $n + j$. Since M_g cannot distinguish between $(x_i, d(i))$ and $(y_j, d(i))$ until it has queried either position i or position $n + j$ (and may not be allowed to do so, depending on whether it is running on $(x_i, d(i))$ or $(y_j, d(i))$), the only way to get out of the dilemma is to neither query position i nor position $n + j$. However, this implies that $g(x_i, d(i)) = g(y_j, e(j)) = g(0^{2n}, d(i)) = 1$. Moreover, M_g cannot distinguish whether it is running on input $(x_i, d(i))$, $(y_j, d(i))$, or $(0^{2n}, d(i))$. Since $0^{2n} \in K_0$, there exists (at least one) $e' \neq d(i)$ such that $g(0^{2n}, e') = 1$. Let p be a position in 0^{2n} such that M_g neither queries p when running on input $(0^{2n}, d(i))$, nor when running on input $(0^{2n}, e')$. Such a position exists since $\frac{n}{2} > 2 \cdot c \cdot \log^k(2n + c \log^k 2n)$. Consequently, $g(0^{p-1}10^{2n-p}, d(i)) = 1$ and $g(0^{p-1}10^{2n-p}, e') = 1$. Hence, $f(0^{p-1}10^{2n-p}) \in K_0'$ although $0^{p-1}10^{2n-p} \in L_0$. This is a contradiction, hence $d(i) \neq e(j)$.

Since $i \notin C(j)$ and $n + j \notin B(i)$ it follows that $g(x_i, d(i)) = g(y_j, e(j)) = g(z_{i,j}, d(i)) = g(z_{i,j}, e(j)) = 1$. From $d(i) \neq e(j)$, we can then conclude that $f(z_{i,j}) \in 0^*10^*1(0 \cup 1)^*$ and thus $f(z_{i,j}) \in K_0'$ although $z_{i,j} \in L_0$. This contradiction proves that no such f can exist; hence $(L_0|K_0) \not\leq_{\mathrm{m}}^{\mathrm{plt}} (L_0'|K_0')$ and by Proposition 7.7 $(L|K) \not\leq_{\mathrm{m}}^{\mathrm{pte}} (L'|K')$. $\square$

Lemma 7.28 *There exists an oracle O such that* $(\mathrm{UP} \vee \mathrm{UP})^O \not\subseteq \mathrm{1NP}^O$.

Proof. This follows directly from $\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}((1 \cup 2 \cup 12)|\varepsilon) = \mathrm{UP} \vee \mathrm{UP}$ (Lemma 7.24.3), $\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(1|(\varepsilon \cup 111^*)) = \mathrm{1NP}$ (Lemma 7.24), $((1 \cup 2 \cup 12)|\varepsilon) \not\leq_{\mathrm{m}}^{\mathrm{pte}} (1|(\varepsilon \cup 111^*))$ (Lemma 7.27) and Theorem 7.9. $\square$

We now know that e-classes of languages outside U are not in 1NP.

[3] Otherwise, $n+j \in B(i)$ has to hold for at least $\frac{n^2}{2}$ tuples. The reasoning is analogue.

The next theorem will enable us to understand the languages inside U better. As it turns out, we can avail ourselves of a well-known algebraic property of Σ^* to obtain a convenient characterisation of U.

Definition 7.29 *A partial ordering is a* well-partial ordering *if it contains no infinite descending sequence and no infinite antichain (i.e., a set of pairwise incomparable elements).*

Theorem 7.30 ([Hig52]) $(\Sigma^*, \preceq)$ *is a well-partial ordering.*

The following theorem provides the announced characterisation of U, the class that precisely corresponds to 1NP in the e-model.

Theorem 7.31 *The following statements are equivalent for any language $L \subseteq \Sigma^*$.*

1. $L \in \mathcal{R}^{\mathrm{pte}}(1)$*, the pte-closure of* $\{1\}$*.*
2. *For all oracles O it holds that* $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L)^O \subseteq \mathrm{1NP}^O$*.*
3. $L \in \mathrm{U}$*, that means both conditions, P1 and P2, fail for L.*
4. *There exist finite sets $A, B \subseteq \Sigma^*$ such that*

$$L = \{w \,\big|\, A \preceq_1 w \text{ and } (\forall v \in B)[v \not\preceq w]\}.^4 \tag{7.1}$$

Proof. **1 $\Leftrightarrow$ 2** : This is an immediate consequence of Theorem 7.9, since for all oracles O, $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(1)^O = \mathrm{1NP}^O$.

2 $\Rightarrow$ 3 : Assume that relative to all oracles, $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L) \subseteq \mathrm{1NP}$. From Lemmas 7.22, 7.26 and Lemmas 7.23, 7.28, we know that if L satisfies P1 or P2, we can construct an oracle O such that $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L)^O \not\subseteq \mathrm{1NP}^O$. This contradicts our assumption. Therefore, L neither satisfies P1 nor P2.

3 $\Rightarrow$ 4 : Let

$$A = \{v \in L \,\big|\, (\forall v' \prec v)[v' \notin L]\}.$$

Observe that A can be seen as the set of minimal words in L. Furthermore, the elements in A are pairwise incomparable with respect to $\preceq$. From Theorem 7.30 and Definition 7.29 it follows that A is finite. Let

$$B = \{w \notin L \,\big|\, (\exists v \in A)[v \preceq w \text{ and } \forall w'[v \preceq w' \prec w \Rightarrow w' \in L]]\}.$$

This set can be thought of as the set of minimal words outside L that have predecessors in A. We claim that B is finite as well: Otherwise, since A is finite, there exists $v \in A$ such that the following subset of B is infinite.

$$B' = \{w \notin L \,\big|\, v \preceq w \text{ and } \forall w'[v \preceq w' \prec w \Rightarrow w' \in L]\}.$$

[4] B can be thought of as the set of forbidden subwords, i.e., events that may not occur in words from L. Contrary, A represents the set of events such that every word in L triggers exactly one such event.

Observe that the elements in B' are pairwise incomparable with respect to $\preceq$. Again, from Theorem 7.30 and Definition 7.29 it follows that B is finite.

We are going to show Equation (7.1). Let $w \in L$. So there exists $v \in A$ such that $v \preceq w$. Assume there exist different $v_1, v_2 \in A$ such that $v_1 \preceq w$ and $v_2 \preceq w$. It follows that v_1, v_2, and w are nonempty. This implies that L satisfies condition P2 which contradicts our assumption. Therefore, there exists exactly one $v \in A$ such that $v \preceq w$. If $v \preceq_k w$ for some $k \geq 2$, then L satisfies condition P2 which again is a contradiction. So $v \preceq_1 w$ and hence $A \preceq_1 w$.

Assume now that there exists $v \in B$ such that $v \preceq w$. By B's definition, there exists $v' \in A$ such that $v' \preceq v$ and for all w', $[v' \preceq w' \prec v \Rightarrow w' \in L]$. In particular, $v' \preceq v \preceq w$ and $v' \in L$, $v \notin L$, and $w \in L$. Hence L satisfies condition P1 which contradicts our assumption. So there does not exist such $v \in B$ and therefore, w belongs to the right-hand side of Equation (7.1). This shows the inclusion $\subseteq$ in Equation (7.1).

Let w be an element of the right-hand side of (7.1). Hence there exists precisely one $v \in A$ such that $v \preceq w$. Assume $w \notin L$ and choose a shortest word $u \notin L$ such that $v \preceq u \preceq w$. It follows that $u \in B$. Together with $u \preceq w$ this implies that w is not an element of the right-hand side of (7.1). This contradiction shows $w \in L$ and finishes the proof of Equation (7.1).

4 $\Rightarrow$ 2 : Let $A = \{u_1, \ldots, u_m\}$ and $B = \{v_1, \ldots, v_n\}$ where $m = |A|$ and $n = |B|$. Let $L' \in \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L)$. So there exists a polynomial-time Turing machine M whose computation paths output symbols from $\Sigma \cup \{\varepsilon\}$ such that $x \in L' \Leftrightarrow \beta_M(x) \in L$. Define a nondeterministic machine N that works as follows on input x. First, N splits into $m + n$ paths $p_1, \ldots, p_m$ and $q_1, \ldots, q_n$. If $u_i = \varepsilon$, then path p_i outputs 1. If $u_i \neq \varepsilon$, then on path p_i the machine nondeterministically guesses an occurrence of u_i (by guessing the positions of u_i's letters) in the leaf string $\beta_M(x)$. If such a guess is successful, then N outputs 1, otherwise it outputs ε. Similarly, on path q_i the machine nondeterministically guesses an occurrence of v_i in $\beta_M(x)$. If such a guess is successful, then N outputs 11 (by producing two neighbouring paths with output 1), otherwise it outputs ε. From (7.1) it follows that

$$x \in L' \Leftrightarrow \beta_M(x) \in L \Leftrightarrow \beta_N(x) = 1.$$

Hence $L' \in \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(1)$ and therefore $\mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(L) \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\varepsilon}(1)$. Finally, observe that our argumentation is relativisable. □

We can now formulate the new gap theorem.

Theorem 7.32 *Let L be a nontrivial language.*

1. *If $L \in \mathrm{U}$, then the e-class of L is contained in* 1NP.
2. *If $L \notin \mathrm{U}$, then the e-class of L contains* UP$\dot{\vee}$coUP *or* UP $\vee$ UP.

Proof. Follows from Theorem 7.31 and the fact that the Lemmas 7.22 and 7.23 are relativisable. □

We summarise properties of U. Accordingly, U is not a (positive) variety of languages [PW97], since it is not closed under union.

Theorem 7.33 *1.* $\mathrm{U} \subseteq \Sigma_1^{\mathrm{FO}} \wedge \Pi_1^{\mathrm{FO}}$ *which is the second level of the Boolean hierarchy over* Σ_1^{FO}*. In particular,* $\mathrm{U} \subseteq \mathrm{SF} \subseteq \mathrm{REG}$*.*

2. U *is closed under intersection.*

3. U *is neither closed under union nor under complement.*

Proof. For the first statement, let $L \in \mathrm{U}$. By Theorem 7.31, there exist finite sets A and B such that

$$L = \{w \mid A \preceq_1 w \text{ and } (\forall v \in B)[v \not\preceq w]\}.$$

The expression $A \preceq_1 w$ is equivalent to

$$(\exists v \in A)[v \preceq w] \text{ and } w \notin C$$

where $C =_{\mathrm{def}} \{w \mid (\exists k \geq 2)[A \preceq_k w]\}$. C is closed upwards with respect to $\preceq$. From Theorem 7.30 it follows that C is finitely generated, i.e., there exists a finite D such that $C = \{w \mid (\exists v \in D)[v \preceq w]\}$. So $A \preceq_1 w$ is equivalent to

$$(\exists v \in A)[v \preceq w] \text{ and } (\forall v \in D)[v \not\preceq w].$$

Hence for $L_1 =_{\mathrm{def}} \{w \mid (\exists v \in A)[v \preceq w]\}$ and $L_2 =_{\mathrm{def}} (\forall v \in B \cup D)[v \not\preceq w]$ it holds that $L = L_1 \cap L_2$.

For the second statement, let $L_1, L_2 \in \mathrm{U}$ and let $L =_{\mathrm{def}} L_1 \cap L_2$. First, we show that $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L) \subseteq \mathrm{1NP}$. For this, let $A \in \mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L)$, i.e., there exists a nondeterministic polynomial time transducer M that produces on every computation path a symbol or the empty word such that

$$x \in A \Leftrightarrow \beta_M(x) \in L.$$

By Theorem 7.31, $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L_1) \subseteq \mathrm{1NP}$ and $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L_2) \subseteq \mathrm{1NP}$. So there exist $A_1, A_2 \in \mathrm{1NP}$ such that $(x \in A_1 \Leftrightarrow \beta_M(x) \in L_1)$ and $(x \in A_2 \Leftrightarrow \beta_M(x) \in L_2)$. Therefore,

$$x \in A_1 \cap A_2 \Leftrightarrow \beta_M(x) \in L_1 \cap L_2 = L.$$

So $A = A_1 \cap A_2$. It follows that $A \in \mathrm{1NP}$, since 1NP is closed under intersection. This shows $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L) \subseteq \mathrm{1NP}$. Since our argumentation is relativisable, it also shows that for all oracles O, $\mathrm{Leaf}_\varepsilon^{\mathrm{p}}(L)^O \subseteq \mathrm{1NP}^O$. So by Theorem 7.31, $L \in \mathrm{U}$.

For the third statement, let $L_1 =_{\mathrm{def}} \{0\}$ and $L_2 =_{\mathrm{def}} \{00\}$. Observe that $L_1, L_2 \in \mathrm{U}$. However, $L_1 \cup L_2 = \{0, 00\} \notin \mathrm{U}$, since it satisfies condition P2 for $k = 2$, $u = v = 0$, and

$w = 00$. So U is not closed under union. If U were closed under complement, then by the closure under intersection and by De Morgan's law it is also closed under union which is not true. □

7.5 Gap Theorems and Perfect Correspondences- an Overview

Table 7.1 summarises the results of this chapter.

$\mathcal{C}$	$\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(\mathcal{C})$	if $B \notin \mathcal{C}$ then $\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(B)$ contains	if $B \in \mathrm{REG} - \mathcal{C}$ then $\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(B)$ contains	if $B \in \mathrm{SF} - \mathcal{C}$ then $\mathrm{Leaf}_{\varepsilon}^{\mathrm{p}}(B)$ contains
$\emptyset$	$\emptyset$	UP or coUP	NP, coNP, or $\mathrm{MOD}_p\mathrm{P}$ for a prime p	NP or coNP
Σ_1^{FO}	NP	coUP	coNP, co1NP, or $\mathrm{MOD}_p\mathrm{P}$ for a prime p	coNP or co1NP
Π_1^{FO}	coNP	UP	NP, 1NP, or $\mathrm{MOD}_p\mathrm{P}$ for a prime p	NP or 1NP
U	1NP	UP $\vee$ UP or UP$\dot{\vee}$coUP	UP $\vee$ UP or UP$\dot{\vee}$coUP	UP $\vee$ UP or UP$\dot{\vee}$coUP
coU	co1NP	coUP $\wedge$ coUP or UP$\dot{\wedge}$coUP	coUP $\wedge$ coUP or UP$\dot{\wedge}$coUP	coUP $\wedge$ coUP or UP$\dot{\wedge}$coUP
Δ_2^{FO}	Δ_2^{P}	–	$\mathrm{AU}\Sigma_2^{\mathrm{P}}$ or $\mathrm{AU}\Pi_2^{\mathrm{P}}$	$\mathrm{AU}\Sigma_2^{\mathrm{P}}$ or $\mathrm{AU}\Pi_2^{\mathrm{P}}$
Σ_2^{FO}	Σ_2^{P}	–	$\mathrm{AU}\Pi_2^{\mathrm{P}}$	$\mathrm{AU}\Pi_2^{\mathrm{P}}$
Π_2^{FO}	Π_2^{P}	–	$\mathrm{AU}\Sigma_2^{\mathrm{P}}$	$\mathrm{AU}\Sigma_2^{\mathrm{P}}$

Table 7.1: Summary of the gap theorems, B is a language different from $\emptyset$ and Σ^*.

The results in the table can be interpreted as gap theorems for ε-leaf-language definability. For instance, the row about Σ_1^{FO} tells us that any e-class either is contained in NP or contains at least coUP. Hence, once an e-class becomes bigger than NP, its complexity jumps to at least NP $\cup$ coUP. Second, there exists evidence that classes in the columns 3–5 are not contained in the corresponding class of column 2. In any case there exist oracles relative to which this non-containment holds. Third, all classes in the first column are decidable, i.e., on input of a finite automaton A we can decide whether the language accepted by A belongs to the class. This permits a decidable and precise classification of e-classes under the assumption that the classes in the 4th column are not contained in the respective class in the 2nd column.

We identified U to be the class of all languages whose e-class is (robustly) contained in 1NP. A language belongs to U if and only if membership of a word can be expressed in terms of a unique occurrence of a substring and in terms of forbidden substrings. This shows that U is a class of regular languages. We proved a decidable characterisation of U, a so-called forbidden-pattern characterisation. It exactly reveals the structure in a finite automaton that is responsible for shifting a language outside U.

We conclude with an overview of perfect correspondences between classes of starfree languages and the polynomial hierarchy.

$\mathcal{C}$	$\mathcal{K}$	Leaf-language model	$\overline{\mathrm{REG}}$
Σ_1^{FO}	NP	e	$\surd$
Π_1^{FO}	coNP	e	$\surd$
U	1NP	e	$\surd$
Δ_2^{FO}	Δ_2^{P}	e	
Σ_2^{FO}	Σ_2^{P}	e	
Π_2^{FO}	Π_2^{P}	e	
$\mathcal{B}_0$	P	u	$\surd$
$\mathcal{B}_{1/2}$	NP	u	
co$\mathcal{B}_{1/2}$	coNP	u	
$\mathcal{B}_1$	BC(NP)	u	
$\mathcal{B}_{3/2}$	Σ_2^{P}	u	

Table 7.2: Summary of perfect correspondences via different leaf language models. A symbol "$\surd$" in the fourth column indicates that the perfect correspondence between $\mathcal{C}$ and $\mathcal{K}$ can be extended to non-regular languages, i.e. for *all* $L \in \overline{\mathcal{C}}$ (this includes all non-regular languages), there exists an oracle O such that $\mathrm{Leaf}^{\mathrm{p}}(L)^O \not\subseteq \mathcal{K}^O$.

Bibliography

[AB96] M. Agrawal and S. Biswas. NP-creative sets: A new class of creative sets in NP. *Mathematical Systems Theory*, 29(5):487–505, 1996.

[AFH87] K. Ambos-Spies, H. Fleischhack, and H. Huwig. Diagonalizations over polynomial time computable sets. *Theoretical Computer Science*, 51:177–204, 1987.

[Agr02] M. Agrawal. Pseudo-random generators and structure of complete degrees. In *IEEE Conference on Computational Complexity*, pages 139–147, 2002.

[Amb84] K. Ambos-Spies. P-mitotic sets. In E. Börger, G. Hasenjäger, and D. Roding, editors, *Logic and Machines*, volume 171 of *Lecture Notes in Computer Science*, pages 1–23. Springer Verlag, 1984.

[Arf91] M. Arfi. Opérations polynomiales et hiérarchies de concaténation. *Theoretical Computer Science*, 91:71–84, 1991.

[BBFG91] R. Beigel, M. Bellare, J. Feigenbaum, and S. Goldwasser. Languages that are easier than their proofs. In *IEEE Symposium on Foundations of Computer Science*, pages 19–28, 1991.

[BCS92] D. P. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104:263–283, 1992.

[BF92] R. Beigel and J. Feigenbaum. On being incoherent without being very hard. *Computational Complexity*, 2:1–17, 1992.

[BG82] A. Blass and Y. Gurevich. On the unique satisfiability problem. *Information and Control*, 82:80–88, 1982.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the P=NP problem. *SIAM Journal on Computing*, 4:431–442, 1975.

[BH77] L. Berman and J. Hartmanis. On isomorphism and density of NP and other complete sets. *SIAM Journal on Computing*, 6:305–322, 1977.

[BHT98] H. Buhrman, A. Hoene, and L. Torenvliet. Splittings, robustness, and structure of complete sets. *SIAM Journal on Computing*, 27:637–653, 1998.

[BK78] J. A. Brzozowski and R. Knast. The dot-depth hierarchy of star-free languages is infinite. *Journal of Computer and System Sciences*, 16:37–55, 1978.

[BKS99] B. Borchert, D. Kuske, and F. Stephan. On existentially first-order definable languages and their relation to NP. *Theoretical Informatics and Applications*, 33:259–269, 1999.

[BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[BLS+05] B. Borchert, K. Lange, F. Stephan, P. Tesson, and D. Thérien. The dot-depth and the polynomial hierarchies correspond on the delta levels. *International Journal of Foundations of Computer Science*, 16(4):625–644, 2005.

[BM95] J. Balcazar and E. Mayordomo. A note on genericty and bi-immunity. In *Proceedings of the Tenth Annual IEEE Conference on Computational Complexity*, pages 193–196, 1995.

[Bor95] B. Borchert. On the acceptance power of regular languages. *Theoretical Computer Science*, 148:207–225, 1995.

[BRS91] R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. In *Proceedings 23rd Symposium on Theory of Computing*, pages 1–9. ACM Press, 1991.

[Brz76] J. A. Brzozowski. Hierarchies of aperiodic languages. *RAIRO Inform. Theor.*, 10:33–49, 1976.

[BSS99] B. Borchert, H. Schmitz, and F. Stephan. Unpublished manuscript, 1999.

[BT96] H. Buhrman and L. Torenvliet. P-selective self-reducible sets: A new characterization of P. *Journal of Computer and System Sciences*, 53:210–217, 1996.

[BV97] D. Boneh and R. Venkatesan. Rounding in lattices and its cryptographic applications. In *SODA*, pages 675–681, 1997.

[BV98] H.-J. Burtschick and H. Vollmer. Lindström quantifiers and leaf language definability. *International Journal of Foundations of Computer Science*, 9:277–294, 1998.

[BWSD77] R. V. Book, C. Wrathall, A. L. Selman, and D. P. Dobkin. Inclusion complete tally languages and the hartmanis-berman conjecture. *Mathematical Systems Theory*, 11:1–8, 1977.

[CB71] R. S. Cohen and J. A. Brzozowski. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5:1–16, 1971.

[CGH+88] J.-Y. Cai, T. Gundermann, J. Hartmanis, L. A. Hemachandra, V. Sewelson, K. W. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17:1232–1252, 1988.

[CH90] J. Y. Cai and L. Hemachandra. On the power of parity polynomial time. *Mathematical Systems Theory*, 23(2):95–106, 1990.

[DF03] R. Downey and L. Fortnow. Uniformly hard languages. *Theoretical Computer Science*, 298:303–315, 2003.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[EHTY92] D. Eppstein, L. A. Hemachandra, J. Tisdall, and B. Yener. Simultaneous strong separations of probabilistic and unambiguous complexity classes. *Mathematical Systems Theory*, 25:23–36, 1992.

[Eil76] S. Eilenberg. *Automata, languages and machines*, volume B. Academic Press, New York, 1976.

[For79] S. Fortune. A note on sparse complete sets. *SIAM Journal on Computing*, 8(3):431–433, 1979.

[FR02] L. Fortnow and J. Rogers. Separability and one-way functions. *Computational Complexity*, 11(3-4):137–157, 2002.

[Fu93] B. Fu. On lower bounds of the closeness between complexity classes. *Mathematical Systems Theory*, 26(2):187–202, 1993.

[Gil77] J. Gill. Computational complexity of probabilistic turing machines. *SIAM Journal on Computing*, 6:675–695, 1977.

[Gla05] C. Glaßer. Polylog-time reductions decrease dot-depth. In *Proceedings 22nd Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.

[GOP+05] C. Glaßer, M. Ogihara, A. Pavan, A. L. Selman, and L. Zhang. Autoreducibility, mitoticity, and immunity. In *Proceedings 30th International Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 387–398. Springer-Verlag, 2005.

[GPSS06] C. Glaßer, A. Pavan, A. L. Selman, and S. Sengupta. Properties of NP-complete sets. *SIAM Journal on Computing*, 36(2), 2006.

[GPSZ05] C. Glaßer, A. Pavan, A. L. Selman, and L. Zhang. Redundancy in complete sets. Technical Report 05-068, Electronic Colloquium on Computational Complexity (ECCC), 2005.

[GPSZ06] C. Glaßer, A. Pavan, A. L. Selman, and L. Zhang. Redundancy in complete sets. In *Proceedings 23nd Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2006.

[GPT08] C. Glaßer, A. Pavan, and S. Travers. False positives and NP-hard sets. 2008. Submitted.

[GS88] J. Grollmann and A. L. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988.

[GSTW07] C. Glaßer, A. L. Selman, S. Travers, and K. W. Wagner. The complexity of unions of disjoint sets. In *Proceedings 24th International Symposium on Theoretical Aspects of Computer Science*, volume 4393 of *Lecture Notes in Computer Science*, pages 248–259. Springer Verlag, 2007.

[GSTZ07] C. Glaßer, A. L. Selman, S. Travers, and L. Zhang. Non-mitotic sets. In *Proceedings 27th Symposium on Foundations of Software Technology and Theoretical Computer Science*, volume 4855 of *Lecture Notes in Computer Science*, pages 146–157. Springer Verlag, 2007.

[GT07] C. Glaßer and S. Travers. Machines that can output empty words. *Theory of Computing Systems*, 2007. To appear.

[GTW06] C. Glaßer, S. Travers, and K. W. Wagner. Perfect correspondences between dot-depth and polynomial-time hierarchy. In *Proceedings Tenth International Conference Developments in Language Theory*, volume 4036 of *Lecture Notes in Computer Science*, pages 408–419. Springer Verlag, 2006.

[GW86] T. Gundermann and G. Wechsung. Nondeterministic Turing machines with modified acceptance. In *Proceedings 12th Symposium on Mathematical Foundations of Computer Science*, volume 233 of *Lecture Notes in Computer Science*, pages 396–404. Springer Verlag, 1986.

[Hig52] G. Higman. Ordering by divisibility in abstract algebras. In *Proc. London Math. Soc.*, volume 3, pages 326–336, 1952.

[HJRW98] L. A. Hemaspaandra, Z. Jiang, J. Rothe, and O. Watanabe. Boolean operations, joins, and the extended low hierarchy. *Theoretical Computer Science*, 205(1-2):317–327, 1998.

[HLS+93] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207, 1993.

[HMU07] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Series in Computer Science. Addison-Wesley, Reading, MA, 2007.

[Hom90] S. Homer. Structural properties of nondeterministic complete sets. In *Structure in Complexity Theory Conference*, pages 3–10, 1990.

[Hom97] S. Homer. Structural properties of complete problems for exponential time. In A. Selman and L. A. Hemaspaandra, editors, *Complexity Theory Retrospective II*, pages 135–153. Springer Verlag, New York, 1997.

[HOW92] L. A. Hemachandra, M. Ogiwara, and O. Watanabe. How hard are sparse sets? In *Structure in Complexity Theory Conference*, pages 222–238, 1992.

[HP06] J. Hitchcock and A. Pavan. Comparing reductions to NP-complete sets. Technical Report TR06-039, Electronic Colloquium on Computational Complexity, 2006.

[HS92] S. Homer and A. Selman. Oracles for structural properties: The isomorphism problem and public-key cryptography. *Journal of Computer and System Sciences*, 44(2):287–301, 1992.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17:935–938, 1988.

[JY85] D. Joseph and P. Young. Some remarks on witness functions for nonpolynomial and noncomplete sets in NP. *Theoretical Computer Science*, 39:225–237, 1985.

[KMR95] S. A. Kurtz, S. R. Mahaney, and J. S. Royer. The isomorphism conjecture fails relative to a random oracle. *Journal of the Association for Computing Machinery*, 42:401–420, 1995.

[Kna83] R. Knast. A semigroup characterization of dot-depth one languages. *RAIRO Inform. Theor.*, 17:321–330, 1983.

[KS97] J. Köbler and U. Schöning. High sets for NP. In *Advances in Algorithms, Languages, and Complexity*, pages 139–156, 1997.

[KSW87] J. Köbler, U. Schöning, and K. W. Wagner. The difference and the truth-table hierarchies for NP. *RAIRO Inform. Théor.*, 21:419–435, 1987.

[Lac67] A. H. Lachlan. The priority method. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 13:1–10, 1967.

[Lad73] R. E. Ladner. Mitotic recursively enumerable sets. *Journal of Symbolic Logic*, 38(2):199–211, 1973.

[LLS75] R. E. Ladner, N. A. Lynch, and A. L. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.

[Lon78] T. J. Long. *On some Polynomial Time Reducibilities.* PhD thesis, Purdue University, Lafayette, Ind., 1978.

[Lov79] L. Lovaáz. On the shannon capacity of graphs. *IEEE Transactions on Information Theory*, 25:1–7, 1979.

[Myh55] J. Myhill. Creative sets. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 1:97–108, 1955.

[NR98] R. Niedermeier and P. Rossmanith. Unambiguous computations and locally definable acceptance types. *Theoretical Computer Science*, 194(1-2):137–161, 1998.

[Ogi91] M. Ogiwara. On P-closeness of polynomial-time hard sets. manuscript, 1991.

[OH93] M. Ogiwara and L. Hemachandra. A complexity theory of feasible closure properties. *Journal of Computer and System Sciences*, 46:295–325, 1993.

[OW91] M. Ogiwara and O. Watanabe. On polynomial-time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal on Computing*, 20(3):471–483, 1991.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

[Pin98] J. E. Pin. Bridges for concatenation hierarchies. In *Proceedings 25th ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 431–442. Springer Verlag, 1998.

[Pos44] E. L. Post. Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, 50:284–316, 1944.

[PP86] D. Perrin and J. E. Pin. First-order logic and star-free sets. *Journal of Computer and System Sciences*, 32:393–406, 1986.

[PS02] A. Pavan and A. L. Selman. Separation of NP-completeness notions. *SIAM Journal on Computing*, 31(3):906–918, 2002.

[PW97] J. E. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory of computing systems*, 30:383–422, 1997.

[Rog67] H. Rogers Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1967.

[Sch83] U. Schöning. A low and a high hierarchy within NP. *Journal of Computer and System Sciences*, 27(1):14–28, 1983.

[Sch86] U. Schöning. Complete sets and closeness to complexity classes. *Mathematical Systems Theory*, 19(1):29–41, 1986.

[Sch01] H. Schmitz. *The Forbidden-Pattern Approach to Concatenation Hierarchies.* PhD thesis, Fakultät für Mathematik und Informatik, Universität Würzburg, 2001.

[Sel79] A. L. Selman. P-selective sets, tally languages, and the behavior of polynomial-time reducibilities on NP. *Mathematical Systems Theory*, 13:55–65, 1979.

[Sel82] A. Selman. Reductions on NP and p-selective sets. *Theoretical Computer Science*, 19:287–304, 1982.

[Sel88] A. L. Selman. Natural self-reducible sets. *SIAM Journal on Computing*, 17(5):989–996, 1988.

[Sho76] J. R. Shoenfield. Degrees of classes of RE sets. *Journal of Symbolic Logic*, 41(3):695–696, 1976.

[ST07] H. Spakowski and R. Tripathi. On the power of unambiguity in alternating machines. *Theory of Computing Systems*, 41(2):291–326, 2007.

[Sto77] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[Str81] H. Straubing. A generalization of the Schützenberger product of finite monoids. *Theoretical Computer Science*, 13:137–150, 1981.

[Str85] H. Straubing. Finite semigroups varieties of the form $V * D$. *J. Pure Appl. Algebra*, 36:53–94, 1985.

[Sze87] R. Szelepcsényi. The method of forcing for nondeterministic automata. *Bull. of the EATCS*, 33:96–100, 1987.

[Thé81] D. Thérien. Classification of finite monoids: the language approach. *Theoretical Computer Science*, 14:195–208, 1981.

[Tho84] W. Thomas. An application of the Ehrenfeucht–Fraïssé game in formal language theory. *Société Mathématique de France, mémoire 16*, 2:11–21, 1984.

[Tra70] B. Trahtenbrot. On autoreducibility. *Dokl. Akad. Nauk SSSR*, 192, 1970. Translation in Soviet Math. Dokl. 11: 814– 817, 1970.

[Tur36] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society, Series 2, 42*, page 230–265, 1936.

[Val76] L. G. Valiant. Relative complexity of checking and evaluation. *Information Processing Letters*, 5:20–23, 1976.

[Ver93] N. K. Vereshchagin. Relativizable and non-relativizable theorems in the polynomial theory of algorithms. *Izvestija Rossijskoj Akademii Nauk*, 57:51–90, 1993. In Russian.

[Vol99] H. Vollmer. Uniform characterizations of complexity classes. *Complexity Theory Column 23, ACM-SIGACT News*, 30(1):17–27, 1999.

[Wag04a] K. W. Wagner. Leaf language classes. In *Proceedings International Conference on Machines, Computations, and Universality*, volume 3354 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.

[Wag04b] K. W. Wagner. New BCSV theorems. Technical Report 337, Inst. für Informatik, Univ. Würzburg, 2004. Available via ftp from *http://www.informatik.uni-wuerzburg.de/reports/tr.html*.

[Wan91] J. Wang. On p-creative sets and p-completely creative sets. *Theoretical Computer Science*, 85(1):1–31, 1991.

[Wra77] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1977.

[WW85] K. W. Wagner and G. Wechsung. On the boolean closure of NP. In *Proceedings International Conference on Fundamentals of Computation Theory*, volume 199 of *Lecture Notes in Computer Science*, pages 485–493. Springer-Verlag, 1985.

[Yao90] A. C.-C. Yao. Coherent functions and program checkers. In *Proceedings of the 22nd ACM Symposium on Theory of Computing*, pages 84–94. ACM Press, 1990.

[Yes83] Y. Yesha. On certain polynomial-time truth-table reducibilities of complete sets to sparse sets. *SIAM Journal on Computing*, 12(3):411–425, 1983.

Index