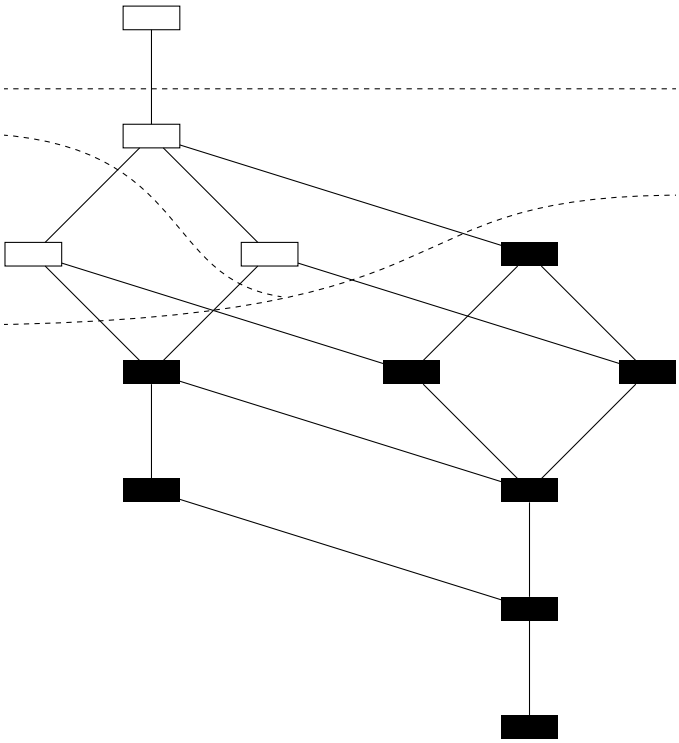
André Hernich

Combining Self-Reducibility with Partial Information Algorithms



Diplomarbeit

Combining Self-Reducibility with Partial Information Algorithms

vorgelegt von

André Hernich

an der Fakultät für Elektrotechnik und Informatik,
Fachgebiet Theoretische Informatik – Algorithmik und Logik
der Technischen Universität Berlin

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

Januar 2005

Erklärung

Die selbstständige und eigenhändige Anfertigung versichere ich an Eides statt.

Berlin, den Januar 2005

Unterschrift

Abstract

A language L is self-reducible if for every word w, the question "$w \in L$?" can be reduced in polynomial time to questions "$q \in L$?" such that every word q is smaller than w. For example, most NP-complete languages are self-reducible.

A partial information algorithm for a language L computes in polynomial time partial information about the membership of its input words in L. Such algorithms are classified depending on the type of partial information they compute. In the literature, languages with many interesting types of partial information algorithms have been studied extensively, for example p-selective, strongly membership comparable, p-cheatable and easily countable languages.

Buhrman, van Helden, and Torenvliet showed that the languages in P can be characterized as self-reducible p-selective languages. We show that this also holds for languages with other types of partial information algorithms. For example, this holds for easily 2-countable languages and languages which are strongly 2-membership comparable as well as its complement. On the other hand, we discuss whether there are self-reducible languages that are not in P and have partial information algorithms.

Zusammenfassung

Eine Sprache L ist selbstreduzierbar, wenn die Frage „$w \in L$?" für jedes Wort w in Polynomialzeit auf Fragen der Art „$q \in L$?" reduziert werden kann, so dass jedes Wort q kleiner als w ist. Beispielsweise sind viele NP-vollständige Sprachen selbstreduzierbar.

Ein Teilinformationsalgorithmus für eine Sprache L berechnet für gegebene Worte in Polynomialzeit Teilinformation darüber, welche dieser Worte in L liegen und welche nicht. Solche Algorithmen werden danach klassifiziert, welche Art von Teilinformation sie berechnen. Aus der Literatur sind bereits Sprachen mit vielen interessanten Arten von Teilinformationsalgorithmen bekannt, wie zum Beispiel die so genannten p-selektiven, strongly membership comparable, p-cheatable und leicht zählbaren Sprachen.

Buhrman, van Helden und Torenvliet zeigten, dass die Sprachen in P dadurch charakterisiert sind, dass sie selbstreduzierbar und p-selektiv sind. Wir zeigen, dass das auch für Sprachen mit anderen Arten von Teilinformationsalgorithmen gilt. Zum Beispiel gilt dies für alle leicht 2-zählbaren Sprachen und Sprachen L, für die L und das Komplement strongly 2-membership comparable ist. Auf der anderen Seite diskutieren wir, ob es selbstreduzierbare Sprachen mit Teilinformationsalgorithmen gibt, die nicht in P liegen.

Danksagung

Ich möchte meinen Betreuern Prof. Dirk Siefkes und Arfst Nickelsen danken. Zum Einen, weil sie diese Arbeit erst ermöglicht haben und zum Anderen wegen der vielen nützlichen Hinweise und Ratschläge. Meine Begeisterung für Teilinformationsklassen habe ich vor allem Arfst Nickelsen und Till Tantau zu verdanken. Vielen Dank auch an die Gruppe TAL und die Mitglieder der Komplexitätstheorie-AG. Darüberhinaus bin ich Alan L. Selman and Frank Stephan für Hinweise und Ratschläge dankbar, die mir sehr beim Schreiben dieser Arbeit geholfen haben.

Contents

Chapter 1

Introduction

Computational complexity theory is concerned among other things with classifying languages with respect to the resources needed to decide them. Languages decidable by deterministic polynomial-time Turing machines make up the complexity class P. Many interesting languages are not known to be in P, but in the class NP of languages decidable by non-deterministic Turing machines in polynomial time. The question whether NP equals P is one of the most important open questions in complexity theory.

The hardest languages in NP are called NP-complete. It is interesting that if an arbitrary NP-complete language is in P, then NP equals P. Many of these languages are *self-reducible* as pointed out by Selman in [Sel82a]. Intuitively, this means that for a word w one can find in polynomial time smaller words such that the question of whether w belongs to the language can be reduced to the question of which of these smaller words belong to the language. There are a lot of self-reducible languages in NP that are not known to be NP-complete, and self-reducible languages that are even not known to be in NP. This makes self-reducibility an interesting concept in its own.

Also, Selman introduced in [Sel82a] the p-selective languages. For such a language L there exists an algorithm that selects in polynomial time from every two given words one word that is in L if at least one of these words is in L. He showed that p-selective languages can be arbitrary complex. This raises the question of whether there can be p-selective languages that are, for example, NP-complete. For the case P $\neq$ NP, Buhrman, van Helden, and Torenvliet [BvHT93] gave a negative answer by showing that every self-reducible p-selective language is contained in P. As a by-product they obtained a characterization of the languages in P as self-reducible p-selective languages.

For a p-selective language L we can compute for each pair of words *partial information* about the membership of these words in L, in particular that the selected word is in L if the other word is in L. Languages which admit computation of other types of partial information are known from the literature. For example, Hoene and Nick-

elsen considered in [HN93] easily m-countable languages L for which we can exclude in polynomial time one possibility for the number of m given words in L. Further examples are p-cheatable [Bei87], easily approximable [BKS95a], and strongly m-membership comparable languages [Köb95].

In general, a *partial information algorithm* for a language L computes on input of a sequence of words in polynomial time some type of partial information about the membership of these words in L. *Partial information classes* contain languages with specific types of partial information algorithms. A framework to study partial information classes in a uniform way has been introduced by Nickelsen [Nic01] who extended the work done in [Bei87], [BKS95b], and [BGK95].

In this thesis, we try to give answers to the following type of question: *Given a partial information class $\mathcal{C}$, is every self-reducible language in $\mathcal{C}$ contained in* P*?* This question is motivated by the result of Buhrman, van Helden, and Torenvliet mentioned above which answers this question for the p-selective languages in a positive way. Several other results appeared already in the literature. For example, in [ABG00, GJY93] it is shown that every self-reducible p-cheatable language is in P. Another result from [BKS95a] shows that all self-reducible easily approximable languages are contained in P. Although we mainly focus on general self-reducibility, we also consider restricted types of self-reducibility such as truth-table and disjunctively self-reducibility.

1.1 Contributions of the Thesis

In [BKS95a] it is asked whether every self-reducible easily m-countable language belongs to P. We answer this question for the case that m equals 2. So, we get a characterization of the languages in P as self-reducible easily 2-countable languages.

On the way toward a solution for the case $m > 2$, it has been proved in [BKS95a] that Turing reductions to easily m-countable languages can be converted into truth-table reductions[1]. We extend this to languages in the partial information class P[NONSEL$_m$]. This class includes the easily m-countable languages, where the inclusion is proper for all $m > 2$. A corollary from the proof is that the question of whether all self-reducible languages in P[NONSEL$_m$] belong to P reduces to the question of whether all truth-table self-reducible languages in this class belong to P. Another corollary which additionally uses a result from [BKS95a] is that all disjunctively self-reducible languages in P[NONSEL$_m$] belong to P.

[1]Reducibilities are introduced in Section 2.3. Self-reducibilities are special reducibilities.

On the other hand, we extend the result from Buhrman, van Helden, and Torenvliet mentioned above to a partial information class that properly includes the p-selective languages: the class of all languages which are strongly 2-membership comparable as well as its complement. We do not know whether the condition for the complement is necessary. Indeed, we argue that self-reducible strongly 2-membership comparable languages might be candidates for self-reducible languages not in P. However, we show that they are at least contained in the complexity class UP.

1.2 Structure of the Thesis

Chapter 2 gives an overview over basic concepts and results in complexity theory. It introduces necessary complexity classes and the notions of reducibility and self-reducibility. We also mention variations such as truth-table and disjunctive (self-) reducibility as well as known results about the complexity of self-reducible languages.

Chapter 3 is an introduction into the theory of partial information classes as presented in [Nic01]. We provide the definitions of pools and families to model pieces and types, respectively, of partial information. Based on these notions we then introduce partial information classes. Section 3.2 discusses some important families. The remaining sections deal with the inclusion structure of partial information classes. For this purpose, we introduce normal forms of families and present known results about their properties. In these sections, we also introduce further families which are defined based on notions related to normal forms.

Chapter 4 deals with closures of partial information classes under various reducibilities. For partial information classes $\mathcal{C}$ and reducibilities, we ask whether $\mathcal{C}$ is closed under this reducibility or whether we can somehow reduce the complexity of reductions to languages in $\mathcal{C}$. Since self-reducibilities are special reducibilities, these results often yield results for self-reducibilities. The first two sections review known results regarding basic reducibilities. In the last section, we show that Turing reductions to languages in $\mathrm{P}[\mathrm{NONSEL}_m]$ are not more powerful than truth-table reductions.

In Chapter 5, we combine self-reducibility and partial information, and show that self-reducible languages in specific partial information classes are in P. First, we ask in Section 5.1, for which partial information classes $\mathcal{C}$ it holds that self-reducible languages in $\mathcal{C}$ are already truth-table self-reducible. This allows us to concentrate in the remaining sections on truth-table self-reducibilities, which are much more convenient. In Section 5.2, we then present partial information classes $\mathcal{C}$ for which self-reducible languages in $\mathcal{C}$

are contained in P. Finally, in Section 5.3, we discuss whether there can be self-reducible partial information languages that are not in P.

Chapter 6 concludes and gives some open questions.

1.3 Basic Definitions and Notation

The natural numbers including zero are denoted by $\mathbb{N}$, the natural numbers without zero by $\mathbb{N}^+$. For each $n \in \mathbb{N}^+$ we write $[n]$ as an abbreviation for $\{1, 2, \ldots, n\}$ and $\mathbb{B}$ for the set of the binary digits 0 and 1.

The cardinality of a finite set A is denoted by $|A|$, where the empty set $\emptyset$ has cardinality zero. We write $A \subseteq B$ if A is a subset of B, and $A \subsetneq B$ if the inclusion is proper. Union, intersection, difference, and the Cartesian product are denoted by $\cup$, $\cap$, $\setminus$ and $\times$, respectively. We define $A^n := A \times A \times \ldots \times A$, where the set A occurs n times in the product on the right side. If $\preceq$ is a partial ordering, we write $a \prec b$ for $a \preceq b$ with $a \neq b$, and $a \succeq b$ ($a \succ b$) for $b \preceq a$ ($b \prec a$).

We use $\Sigma := \{0, 1\}$ as the *standard alphabet*. The set of all words over Σ is denoted by Σ^*. For $n \in \mathbb{N}$ we denote the set of all words $w \in \Sigma^*$ of length n by Σ^n, and the set of all words $w \in \Sigma^*$ of length at most n by $\Sigma^{\leq n}$. The *length of a word* is denoted by $|w|$. The *empty word* is denoted by λ. For two words w_1 and w_2 we write $w_1 w_2$ for the *concatenation* of these words. The function $\langle \cdot, \cdot \rangle \colon \Sigma^* \times \Sigma^* \to \Sigma^*$ is a standard *pairing function* which we extend to more than two words by $\langle w_1, w_2, \ldots, w_n \rangle := \langle \ldots \langle \langle w_1, w_2 \rangle, w_3 \rangle, \ldots, w_n \rangle$. For functions f which take arguments from Σ^* we often write $f(w_1, w_2, \ldots, w_n)$ as an abbreviation for $f(\langle w_1, w_2, \ldots, w_n \rangle)$.

A subset L of Σ^* is called a *language* (over Σ). Its *complement* is given by $\overline{L} := \Sigma^* \setminus L$. The *characteristic function* of L is a function $\chi_L \colon \Sigma^* \to \mathbb{B}$ defined by

$$\chi_L(w) := \begin{cases} 1 & \text{if } w \in L \\ 0 & \text{else.} \end{cases}$$

We extend it to arbitrary many words by defining for all $n \in \mathbb{N}^+$,

$$\chi_L(w_1, w_2, \ldots, w_n) := \chi_L(w_1)\chi_L(w_2) \ldots \chi_L(w_n).$$

Sometimes we fix the number of words to a constant $n \in \mathbb{N}^+$ in which case we write χ_L^n. The *cardinality function* of a language L counts how many words given as arguments to

the function are in L and is defined for all $n \in \mathbb{N}^+$ by

$$\#_L(w_1, w_2, \ldots, w_n) := |\{w_1, w_2, \ldots, w_n\} \cap L| \,.$$

Again, we write $\#_L^n$ if n is fixed.

Words over $\mathbb{B}$ are referred to as *bitstrings* when we want to emphasize that they are considered as possible values of characteristic functions. For a bitstring $b \in \mathbb{B}^n$ and a number $i \in [n]$ we denote by $b[i]$ the symbol at the i-th position of b, or the *projection* of b onto the i-th position. We extend this to sequences $i_1, i_2, \ldots, i_k$ of numbers by $b[i_1, i_2, \ldots, i_k] := b[i_1]b[i_2]\ldots b[i_k]$, and to subsets B of $\mathbb{B}^n$ by $B[i_1, i_2, \ldots, i_k] := \{b[i_1, i_2, \ldots, i_k] \mid b \in B\}$. For $c \in \mathbb{B}$, the number of positions i with $b[i] = c$ is counted by $\#_c(b)$.

Chapter 2

Preliminaries from Complexity Theory

The results presented in this thesis are based on several results from computational complexity theory. This chapter gathers definitions and results from this field which will form the basis of the following chapters. Most of the topics are covered in more detail, for instance, in the books of Hopcroft, Motwani and Ullman [HMU01], Papadimitriou [Pap94], Du and Ko [DK00], and Balcázar, Díaz and Gabarró [BDG88]. They also give excellent introductions into this field.

2.1 Turing Machines and Complexity Classes

This thesis deals mainly with algorithms. To formalize algorithms and to analyze their running time and space usage we use the *Turing machine* model. We consider *deterministic* Turing machines and *non-deterministic* Turing machines.

For a Turing machine M, let $L(M)$ be the *language accepted by* M. We say that M is *polynomially time-bounded* (or a *polynomial-time Turing machine*) if there exists a polynomial p such that M stops on every input w after at most $p(|w|)$ steps. The complexity class P is the set of all languages accepted by deterministic polynomial-time Turing machines. The corresponding complexity class for non-deterministic polynomial-time Turing machines is NP. The class co-NP is the set of the complements of languages in NP. A non-deterministic Turing machine is *unambiguous* if it has on every input at most one accepting computation. Languages accepted by unambiguous non-deterministic polynomial-time Turing machines are exactly the languages in UP. We denote by FP the set of all (total) functions that are computed by deterministic polynomial-time Turing machines.

We also consider *polynomially space-bounded* Turing machines. For such machines there exists a polynomial p such that they visit on every input w at most $p(|w|)$ tape cells. Languages that are accepted by deterministic polynomially space-bounded Turing machines form the class PSPACE.

2.2 Relativized Computations

In many cases, we are interested in the complexity of a language L under the hypothesis that another language K can be solved efficiently. We then pretend to know for all possible words whether they are in K or not and analyze the complexity of L with this additional knowledge. For this purpose, the Turing machine model can be extended so that questions about the membership of words in a given language may be asked during a computation and the correct answers are provided by a so called *oracle* in the next step. Such machines are therefore known in the literature as *oracle Turing machines*.

Definition 2.2.1 (oracle Turing machine). An *oracle Turing machine* M is a Turing machine with a special tape, the *query tape*, and two special states: the *query state* $q_?$, and the *answer state* q_a. The computation of M on input w with an arbitrary language L as *oracle* proceeds like in an ordinary Turing machine, except for the case that M reaches the query state. Then, the word q currently being on the query tape is replaced by $\chi_L(q)$ and the computation of M continues in the answer state. We say that M *queries a word* q. The *language accepted by* M *with oracle* L is denoted by $L(M, L)$.

For arbitrary languages L, the complexity classes P^L and NP^L are defined as the corresponding classes without the superscript, except that the Turing machines are oracle Turing machines that have access to the oracle L. For example, the class P^L consists of all languages decidable by deterministic oracle Turing machines with oracle L. Sometimes we say that a statement A is true *relative* to some language L. We then mean that A is true if all machines have access to L. For example, there are languages L and K such that $\mathrm{P} = \mathrm{NP}$ holds relative to L, and $\mathrm{P} \neq \mathrm{NP}$ holds relative to K [Pap94, p. 340]. This means that $\mathrm{P}^L = \mathrm{NP}^L$ and $\mathrm{P}^K \neq \mathrm{NP}^K$.

We will mostly be concerned with deterministic oracle Turing machines. Therefore, all oracle Turing machines are deterministic in the remaining part of this thesis, unless otherwise stated.

In general, the sequence of queries posed by an oracle Turing machine M on an input w depends on the language plugged in as oracle. The *query tree* of M on an input w

contains for each possible language L a path such that the sequence of labels along this path corresponds to the sequence of queries posed by M on input w with oracle L.

Definition 2.2.2 (query tree). Let M be a polynomial-time oracle Turing machine. The *query tree* of M on an input w, for short $\mathrm{QT}_M(w)$, is a rooted binary tree which satisfies the following conditions.

1. Every interior node is labeled with exactly one word, and each edge is labeled with 0 or 1.

2. If M poses on input w with an oracle L exactly the sequence $q_1, q_2, \ldots, q_k$ of queries, then there exists a path of length k from the root to a leaf such that for all $i \in [k]$, the i-th node on this path is labeled with q_i, and the i-th edge on this path is labeled with $\chi_L(q_i)$. This path is called the *path determined by* L.

3. Every path from the root to a leaf is determined by some language.

Example 2.2.3. We consider an oracle Turing machine M that queries on an input w a word q_1. If q_1 is in the oracle language, then another word, q_2, is queried. The machine accepts w if and only if both words are in the oracle language. Figure 2.1 shows $\mathrm{QT}_M(w)$. The node labeled q_1 is the root node. The bold marked path is the

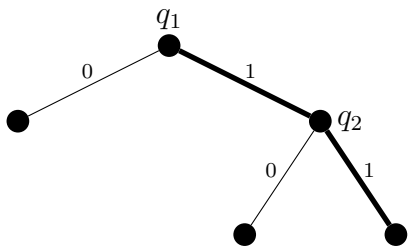


Figure 2.1: The query tree of M on input w.

path determined by any language which contains q_1 and q_2, whereas the path which starts at the root and visits the edges labeled 1 and 0 in this order is determined by any language which contains q_1, but not q_2. △

Observe that for every polynomial-time oracle Turing machine M and every word w, the depth of $\mathrm{QT}_M(w)$ is bounded by a polynomial in $|w|$. This is because every path from the root to a leaf is determined by some language L, i.e. the sequence of labels

along this path corresponds to the sequence of words queried by M on input w with oracle L. Since M is polynomially time-bounded, say by some polynomial p, the length of such a sequence is at most $p(|w|)$.

For the next section, we need a special type of oracle Turing machines, called *non-adaptive oracle Turing machines*. Oracle Turing machines as in Definition 2.2.1 are called adaptive, because in general each query depends on the answers to the queries made before. This is not allowed for non-adaptive oracle Turing machines. Formally, a non-adaptive oracle Turing machine is allowed to enter the query state only once, but on the other hand it may ask a sequence of words in parallel (or *non-adaptively*). To do this, it writes this sequence onto the query tape, for instance "$\langle w_1, w_2, \ldots, w_n \rangle$". The answers are provided by replacing this sequence with the bitstring $\chi_L(w_1, w_2, \ldots, w_n)$, where L is the oracle.

2.3 Reducibilities

A reducibility is a tool to compare the computational complexity of two languages. In general, it is a binary relation on languages which is reflexive and transitive. We say that a language L is *reducible* to another language K with respect to a reducibility $\leq$ (for short, L is $\leq$-reducible to K) if $L \leq K$. Reducibilities are usually defined in terms of oracle Turing machines which decide the reduced language by using the other language as oracle. The type of a reducibility is determined by the time or space the machine may consume, the way it is allowed to access the oracle, the number of queries it is allowed to make, and the way it may evaluate the answers. We only consider *polynomial-time reducibilities* where the oracle Turing machines are polynomially time-bounded. The prefix "polynomial-time" is omitted in the following.

2.3.1 Many-One and Turing Reducibility

The most restrictive reducibility is the many-one reducibility. In a many-one reduction the oracle Turing machine is allowed to make at most one query and it must accept its input if the answer is positive, and it must reject if it is negative. Essentially, the machine transforms an instance w_1 of one problem, L, into an instance w_2 of another problem, K, such that w_1 is in L if and only if w_2 is in K. If there was an efficient algorithm to solve K, then we could easily construct an efficient algorithm to solve L. Therefore L can be seen as "at most as hard" as K.

Definition 2.3.1 (many-one reducibility). A language L is *many-one reducible* to a language K, denoted by $L \leq_{\mathsf{m}}^{\mathsf{p}} K$, if there is a function $f \in \mathrm{FP}$ such that for all words w it holds that $w \in L$ if and only if $f(w) \in K$.

Many-one reducibility is just a special case of Turing reducibility. In a Turing reduction the oracle Turing machine is allowed to make as many queries as possible, where a query may depend on the preceding ones, and they can be evaluated arbitrarily. It is formally defined as follows.

Definition 2.3.2 (Turing reducibility). A language L is *Turing reducible* to a language K, denoted by $L \leq_{\mathsf{T}}^{\mathsf{p}} K$, if there is a polynomial-time oracle Turing machine M with $L(M, K) = L$.

There are many other reducibilities between many-one and Turing reducibility. Truth-table reducibility and its variations as well as positive reducibilities are the most interesting ones there.

2.3.2 Truth-Table, Positive, and Bounded Reducibilities

Truth-table reducibility is a special case of Turing reducibility. Words are queried in parallel instead of sequentially. Based on the mechanism to evaluate the oracle's answers, we distinguish several variations of truth-table reducibility. The most important ones are given in the following definition.

Definition 2.3.3 (truth-table reducibilities).

- A language L is *truth-table reducible* to a language K, denoted by $L \leq_{\mathsf{tt}}^{\mathsf{p}} K$, if there is a non-adaptive polynomial-time oracle Turing machine M with $L(M, K) = L$.

- A language L is *disjunctively reducible* to a language K, denoted by $L \leq_{\mathsf{dtt}}^{\mathsf{p}} K$, if L is truth-table reducible to K via an oracle Turing machine M such that for every word w, if M queries on input w at least one word then $w \in L$ if and only if at least one of the queried words is in K.

- A language L is *conjunctively reducible* to a language K, denoted by $L \leq_{\mathsf{ctt}}^{\mathsf{p}} K$, if L is truth-table reducible to K via an oracle Turing machine M such that for every word w, if M queries on input w at least one word then $w \in L$ if and only if all queried words are in K.

The following equivalent definition of truth-table reducibility appears in [DK00]: A language L is defined to be truth-table reducible to a language K if there exist two functions $f, g \in \mathrm{FP}$ which satisfy the following conditions. On the one hand, the value of f on an input w is a tuple of arbitrary many words. On the other hand, g corresponds for every w to a boolean function $g_w \colon \mathbb{B}^k \to \mathbb{B}$, where k is the number of words in $f(w)$, such that $g_w(\chi_K(f(w))) = \chi_L(w)$.

Disjunctive and conjunctive reducibilities are special cases of positive truth-table reducibility which are itself special truth-table self-reducibilities. In general, positive reducibilities are defined in terms of positive oracle Turing machines as follows.

Definition 2.3.4 (positive reducibilities). An oracle Turing machine M is a *positive oracle Turing machine* if for every two languages L and K with $L \subseteq K$ it holds that $L(M, L) \subseteq L(M, K)$. A language L is *positive Turing reducible* to a language K, denoted by $L \leq^{\mathsf{p}}_{\mathsf{pos}} K$, if L is Turing reducible to K via a positive oracle Turing machine. A language L is *positive truth-table reducible* to a language K, denoted by $L \leq^{\mathsf{p}}_{\mathsf{ptt}} K$, if L is truth-table reducible to K via a positive oracle Turing machine.

Finally, every reducibility $\leq^{\mathsf{p}}_r$ can be restricted in the number of queries that the corresponding oracle Turing machines are allowed to make. We write $L \leq^{\mathsf{p}}_{k\text{–}r} K$ to denote that L is $\leq^{\mathsf{p}}_r$-reducible to K via an oracle Turing machine that makes on every input at most k queries. We should note that for $k > 1$, the reducibility $\leq^{\mathsf{p}}_{k\text{–}\mathsf{tt}}$ is in general not transitive.

2.3.3 Hardness, Completeness, and Reduction Closures

For a reducibility $\leq$, a language L to which all languages of a given complexity class $\mathcal{C}$ are $\leq$-reducible is called $\leq$-*hard for* $\mathcal{C}$. In case that $\leq$ is the Turing reducibility we write that L is $\mathcal{C}$-hard. If L is $\leq$-hard for $\mathcal{C}$ and belongs itself to $\mathcal{C}$, then L is called $\leq$-*complete for* $\mathcal{C}$. For the case that $\leq$ is the many-one reducibility we write that L is $\mathcal{C}$-complete.

Cook showed that the satisfiability problem for boolean expressions that are built from boolean variables, and the boolean connectives $\wedge$, $\vee$, and $\neg$ is NP-complete [Coo71]. The problem is to find out whether there exists a truth assignment which satisfies a given boolean expression or not. The corresponding language is

$$\mathrm{SAT} := \{\varphi \mid \varphi \text{ is a satisfiable boolean expression}\}\,[1].$$

[1] We assume that all objects such as strings, expressions, or graphs that occur in the definition of languages are encoded appropriately as words over Σ^*.

Since then, many other languages have been shown to be complete for certain complexity classes. One of them is the satisfiability problem for quantified boolean expressions which is PSPACE-complete. Such expressions are build from boolean variables, the boolean connectives $\wedge$, $\vee$, and $\neg$, and the quantifiers $\exists$ and $\forall$. The corresponding language is

$$\text{QBF-SAT} := \{\varphi \mid \varphi \text{ is a satisfiable quantified boolean expression}\}.$$

The closure of a complexity class $\mathcal{C}$ under a reducibility $\leq$ (for short, $\leq$-closure of $\mathcal{C}$) is the set of all languages that are $\leq$-reducible to some language in $\mathcal{C}$. The class $\mathcal{C}$ is *closed under $\leq$-reducibility* if the $\leq$-closure of $\mathcal{C}$ equals $\mathcal{C}$. Examples of complexity classes that are closed under many-one reducibility are P, NP, co-NP, and PSPACE. Furthermore, P and PSPACE are closed under Turing reducibility. This is not known for NP. However, Selman [Sel82b] showed that NP is closed under positive Turing reducibility.

Complexity classes $\mathcal{C}$ that are closed under some reducibility $\leq$ are interesting, because if there is a language in $\mathcal{C}$ that is $\leq$-hard for a complexity class $\mathcal{D}$, then $\mathcal{D} \subseteq \mathcal{C}$. For instance, if the language SAT belongs to the class P then we have NP $\subseteq$ P. Since P $\subseteq$ NP it would follow that P = NP. However, at this time it is not known whether the latter statement is true or false. The associated problem is known in the literature as the so called P *versus* NP *problem*.

2.4 Self-Reducibilities

This section deals with a special case of reducibility, called *self-reducibility*. In contrast to reducibilities, self-reducibilities are not binary relations on languages, but can be considered as *properties* of languages that describe some kind of internal structure. Informally, a language is self-reducible if it is reducible to itself and the machine which carries out the reduction queries on every input w only words that are smaller than w. For example, most NP-complete languages including SAT are self-reducible as well as the PSPACE-complete language QBF-SAT (see Examples 2.4.3 and 2.4.4). This may be the main motivation to study self-reducible languages.

2.4.1 Introduction to Self-Reducibility

Informally, a language L is *self-reducible* if it is reducible to itself. However, we restrict the oracle Turing machines which carry out the reductions such that on every input w only words are queried which are *shorter* than w. This condition prevents the machines

from querying its own input to decide whether it is in L or not. According to Beigel [Bei87], this was how self-reducibility was defined originally by Schnorr in [Sch76].

In this thesis we will work with a generalization employed by Meyer and Paterson [MP79]. The queries must be smaller than the input word with respect to some *polynomially related partial ordering* on Σ^*. The following definition of such orderings is essentially the same one as in their work. One must note that the name polynomially related has been used also by Ko [Ko83] for partial orderings on Σ^* which must be in addition to the requirements in Definition 2.4.1 polynomially decidable.

Definition 2.4.1 (polynomially related). A partial ordering $\preceq$ on Σ^* is *polynomially well-founded and length-related* (for short, *polynomially related*) if there is a polynomial p such that

- for all words w_1 and w_2 it holds that $w_1 \preceq w_2$ implies $|w_1| \leq p(|w_2|)$, and
- for every strictly $\preceq$-decreasing chain $w_1 \succ w_2 \succ \ldots \succ w_n$ we have $n \leq p(|w_1|)$.

An example of a polynomially related partial ordering on Σ^* is the one based on word length used in Schnorr's definition of self-reducibility. Originally, self-reducibility has been defined using Turing reducibility. Other variations of self-reducibility appear in the literature. We give a definition of self-reducibility based on reducibilities introduced in Section 2.3.

Definition 2.4.2 (self-reducibility). Let $\leq_r^{\mathsf{p}}$ be any reducibility defined in Section 2.3. A language L is *r-self-reducible* if there exists a polynomially related partial ordering $\preceq$ such that

- L is $\leq_r^{\mathsf{p}}$-reducible to itself via an oracle Turing machine M, and
- M queries on input of a word w only words q with $q \prec w$.

We write that L is r-self-reducible via M (and $\preceq$). The machine M is called a *self-reduction machine* (for L).

Whenever we write *self-reducibility*, we mean T-self-reducibility. Instead of tt-self-reducibility we also say *truth-table self-reducibility*. Similarly, ptt-, dtt-, and ctt-self-reducibility are called *positive*, *disjunctive*, and *conjunctive self-reducibility*, respectively.

From the definition given above, it follows immediately that every 1-tt-self-reducible language is truth-table self-reducible. This implies that it is Turing reducible. However,

it is not positive self-reducible. The other way round, every positive (truth-table) self-reducible language is (truth-table) self-reducible. Every disjunctively or conjunctively self-reducible language is positive truth-table self-reducible. Conjunctively self-reducible languages are the complements of disjunctively self-reducible languages.

2.4.2 Examples of Self-Reducible Languages

Before we proceed we want to look at some examples of typical self-reducible languages. We start by considering the NP-complete language SAT. The property of SAT to be disjunctively self-reducible has been exploited in many works in complexity theory. See, for instance, the chapter "The Self-Reducibility Technique" in [HO02].

Example 2.4.3 (SAT is dtt-self-reducible). The language SAT has been mentioned in Section 2.3. Words of this language are satisfiable boolean expressions which are built in the usual way from boolean variables, and the boolean connectives $\wedge$, $\vee$ and $\neg$.

A possible self-reduction machine M for SAT works on every input w as follows. It checks whether w encodes a boolean expression φ, and if not, it rejects w. Otherwise, a truth assignment which satisfies φ must assign either the value true or false to the first variable v in φ. Let $\varphi_{v:=t}$ denote the formula obtained from φ when setting v to the truth value t and simplifying. Then,

$$\varphi \in \text{SAT} \text{ if and only if } \varphi_{v:=\text{true}} \in \text{SAT} \text{ or } \varphi_{v:=\text{false}} \in \text{SAT}.$$

The machine M determines membership of $\varphi_{v:=\text{true}}$ and $\varphi_{v:=\text{false}}$ in SAT by querying the oracle or by direct evaluation of the resulting formulas. It accepts φ if and only if at least one of these formulas is in SAT. This can be done in polynomial time.

It remains to define an appropriate polynomially related partial ordering. For this, we assume without loss of generality that formulas are encoded such that $\varphi_{v:=\text{true}}$ and $\varphi_{v:=\text{false}}$ yield formulas with shorter encodings. Then, each word queried by M is shorter than its input. It follows that SAT is dtt-self-reducible. $\triangle$

Similarly, QBF-SAT is positive truth-table self-reducible by nearly the same machine as in Example 2.4.3.

Example 2.4.4 (QBF-SAT is ptt-self-reducible). The language QBF-SAT has also been mentioned in Section 2.3. Words in this language encode satisfiable quantified boolean expressions built from boolean variables, the boolean connectives $\wedge$, $\vee$, $\neg$, and the quantifiers $\exists$ and $\forall$.

A possible self-reduction machine M for QBF-SAT works very similar to the machine from Example 2.4.3 for SAT. If w is an input which represents a quantified boolean expression φ, then one of the following two cases occurs. In case that there is no quantifier in φ, M acts like the machine for SAT mentioned above. The second case is that φ equals $Qv.\psi$ for some quantifier Q, a variable v, and a quantified boolean expression ψ. We assume without loss of generality that v is not quantified again in ψ. If $Q = \exists$ then it holds that $\varphi \in$ QBF-SAT if and only if $\psi_{v:=\text{true}} \in$ QBF-SAT or $\psi_{v:=\text{false}} \in$ QBF-SAT, and if $Q = \forall$ then $\varphi \in$ QBF-SAT if and only if $\psi_{v:=\text{true}} \in$ QBF-SAT and $\psi_{v:=\text{false}} \in$ QBF-SAT. Again, M checks these conditions by querying the oracle or by direct evaluation.

As polynomially related partial ordering $\preceq$ we use the one based on length. Then, QBF-SAT is ptt-self-reducible via M and $\preceq$. △

The fact that SAT is dtt-self-reducible can be exhibited to show that other NP-complete languages are dtt-self-reducible. Berman and Hartmanis [BH77] observed that all NP-complete problems they knew are p-isomorphic to SAT. This means that for every such language L there exists a polynomial-time computable bijective function $f\colon \Sigma^* \to \Sigma^*$ whose inverse f^{-1} is also polynomial-time computable and which has the property that a word w is in L if and only if $f(w)$ is in SAT. So, a self-reduction machine M for L can transform an input word w into a formula $\varphi := f(w)$, compute the two queries φ_1, φ_2 posed by the self-reduction machine for SAT from Example 2.4.3, query the two words $f^{-1}(\varphi_1)$ and $f^{-1}(\varphi_2)$, and accept w if and only if at least one of these words is in L. If we define the polynomially related partial ordering $\preceq$ by $w_1 \preceq w_2$ if and only if $f(w_1) \preceq_{\text{SAT}} f(w_2)$, where $\preceq_{\text{SAT}}$ is the partial ordering defined in Example 2.4.3, then L is dtt-self-reducible via M and $\preceq$.

In general, whenever a language L is p-isomorphic to an r-self-reducible language, then L is itself r-self-reducible by the same argument as above. This way, the language which corresponds to the graph isomorphism problem (GI) – which is not believed to be in P nor that it is NP-complete – can be shown to be dtt-self-reducible.

Example 2.4.5 (GI is dtt-self-reducible). A graph is a tuple (V, E) such that V is a finite set whose elements are called nodes, and E is a set of two-element subsets of V whose elements are called edges. Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there exists a bijective mapping f from V_1 to V_2 such that for every two nodes $v, w \in V_1$ it holds that $\{v, w\} \in E_1$ if and only if $\{f(v), f(w)\} \in E_2$. We say that f defines an isomorphism between G_1 and G_2. The graph isomorphism problem (GI) asks whether two given graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic.

We consider the following variation of GI. We name it GI′. For two given graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, and a bijective mapping f from $W_1 \subseteq V_1$ to $W_2 \subseteq V_2$ we are asked whether there exists a bijective mapping f' from V_1 to V_2 such that f' defines an isomorphism between G_1 and G_2 and f' extends f, i.e. for all $v \in W_1$ we have $f'(v) = f(v)$. This problem can be seen p-isomorphic to GI.

Furthermore, in [MP79] it is shown that GI′ is dtt-self-reducible. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs, and f be a bijective mapping from $W_1 \subseteq V_1$ to $W_2 \subseteq V_2$. If $|V_1| = |V_2|$ and $W_1 = V_1$, then f defines an isomorphism between G_1 and G_2 if and only if for each node $v, w \in V_1$ it holds that $\{v, w\} \in E_1$ if and only if $\{f(v), f(w)\} \in E_2$. On the other hand, if $|V_1| \neq |V_2|$, then no extension of f defines an isomorphism between G_1 and G_2. Both conditions can be checked by an oracle Turing machine M in polynomial time without querying the oracle.

Suppose that $|V_1| = |V_2|$ and $W_1 \subsetneq V_1$. Then M picks an arbitrary node $v \in V_1 \setminus W_1$, and defines for all nodes $w \in V_2 \setminus W_2$ a bijective mapping f_w from $W_1 \cup \{v\}$ to $W_2 \cup \{w\}$ by

$$f_w(x) := \begin{cases} f(x) & \text{if } x \in W_1 \\ w & \text{if } x = v. \end{cases}$$

Clearly, f can be extended to a mapping that defines an isomorphism between the two graphs if and only if f_w can be extended to such a mapping for some $w \in V_2 \setminus W_2$. The machine M determines whether the latter condition holds by querying the oracle.

If we define the polynomially related partial ordering $\preceq$ by $\langle G_1, G_2, f\rangle \preceq \langle G_1', G_2', f'\rangle$ if and only if $G_1 = G_1'$, $G_2 = G_2'$, and f is an extension of f', then GI′ is dtt-self-reducible via M and $\preceq$. △

2.4.3 Complexity of Self-Reducible Languages

An interesting question is that of the complexity of self-reducible languages. Are they contained in any of the complexity classes introduced in Section 2.3, or can they be arbitrary complex? An answer has been given by Ko [Ko83]: They are contained in PSPACE, and languages which are self-reducible under special truth-table self-reducibilities are contained in NP, co-NP, or even in P.

Before we give the corresponding theorem, it is convenient to introduce *self-reduction trees*. Such trees are very similar to query trees of deterministic oracle Turing machines. The difference is that not the paths correspond to sequences of queries, but the successors of a node.

Definition 2.4.6 (self-reduction tree). Let M be a self-reduction machine for a language L. The *self-reduction tree* of M on an input w, for short $\mathrm{ST}_M(w)$, is a rooted tree which satisfies the following conditions.

1. Every node is labeled with exactly one word, where the root node is labeled w.
2. If an interior node is labeled w_0, then it has k successor nodes labeled with $w_1, w_2, \ldots, w_k$ if and only if M queries on input w_0 with oracle L exactly k words, and for all $i \in [k]$ it holds that w_i is the i-th queried word.

Example 2.4.7. We consider the self-reduction machine M for SAT from Example 2.4.3 and the boolean expression $\varphi = (v_1 \vee v_2) \wedge v_3$. Figure 2.2 shows a possibility for the self-reduction tree $\mathrm{ST}_M(\varphi)$ of M on input φ when M substitutes always the leftmost variable. We see that the root node is labeled with the input word. Its successor nodes

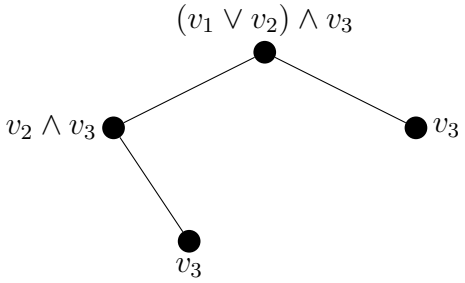


Figure 2.2: A possible self-reduction tree of the oracle Turing machine from Example 2.4.3 on input of the boolean expression $(v_1 \vee v_2) \wedge v_3$.

are labeled with $v_2 \wedge v_3$ and v_3, the two queries of M on input φ. The node labeled $\varphi' = v_2 \wedge v_3$ has only one successor. This is because M queries only ${\varphi'}_{v_2:=\mathrm{true}} = v_3$ on input φ'; ${\varphi'}_{v_2:=\mathrm{false}}$ can be directly evaluated and is false in this case. △

Note that self-reduction trees are also defined for adaptive self-reduction machines. In this case, if a node in this tree is labeled w, then the label of the i-th successor corresponds to the i-th word queried by the machine on input w with the corresponding oracle. In contrast to self-reduction trees for non-adaptive self-reduction machines which can be traversed in a breadth-first fashion, this tree can only be traversed in a depth-first fashion. This is exploited in Theorem 2.4.9 to show that all self-reducible languages are contained in PSPACE.

Finally, let us make the following observation about the depth of self-reduction trees.

Lemma 2.4.8. *For every self-reduction machine M there exists a polynomial p such that for every word w the depth of $\mathrm{ST}_M(w)$ is at most $p(|w|)$.*

Proof. Let M be a self-reduction machine for some language L, and let w be an arbitrary word. We show that each path in $\mathrm{ST}_M(w)$ has a length of at most $p(|w|)$ for some polynomial p. For this, let P be a path from the root to a leaf, and let $w = w_1, w_2, \ldots, w_k$ be the labels of the nodes along P. For every $i \in [k-1]$ it holds that w_{i+1} labels a successor of the node labeled w_i, and by Definition 2.4.6, it follows that w_{i+1} is a word queried by M on input w_i. Since M is a self-reduction machine there exists a polynomially related partial ordering $\preceq$ such that $w_{i+1} \prec w_i$ for every such i. This way we obtain a $\preceq$-decreasing chain $w_1 \succ w_2 \succ \ldots \succ w_k$ which has by Definition 2.4.1 at most a length of $p(|w_1|)$ for some polynomial p. So there are at most $p(|w|)$ many words on each path. □

Now we state Ko's theorem mentioned above and give a proof.

Theorem 2.4.9.

1. *Every self-reducible language is in* PSPACE.
2. *Every dtt-self-reducible language is in* NP.
3. *Every ctt-self-reducible language is in* co-NP.
4. *Every 1-tt-self-reducible language is in* P.

Proof. (1) Let L be a self-reducible language and let M be the corresponding self-reduction machine. To see that L is in PSPACE note that we can determine membership of a word w in L by processing $\mathrm{ST}_M(w)$ in a depth-first fashion as follows. We simulate M on input w, and if it queries a word q, then we recursively determine whether $q \in L$ or not. Finally, we continue the simulation of M on input w until it makes another query, or until it stops in which case we accept w if and only if M accepts it. If d is the depth of $\mathrm{ST}_M(w)$, then we have to maintain at most d simulations of M in parallel each requiring space polynomial in $|w|$. Since d is bounded by a polynomial in $|w|$ by Lemma 2.4.8, we conclude that the whole algorithm just described runs within space polynomial in $|w|$, hence it follows that $L \in$ PSPACE.

(2 and 3) Let L be a dtt-self-reducible language and let M be the corresponding self-reduction machine. Then, it is easy to see that a word w is in L if and only if there is a leaf in $\mathrm{ST}_M(w)$ that is labeled with a word in L. So, a non-deterministic polynomial-time oracle Turing machine M' can guess a path from the root to a leaf (where the label of a successor of a node v is determined by simulating M on v's label and choosing one of the words queried by M) and check whether its label, w', belongs to L or not (again,

by simulating M on input w'). It accepts w if and only if w' is in L. So, it is clear that $L(M') = L$ and we have therefore $L \in$ NP. By symmetry, we also have $L \in$ co-NP, if L is ctt-self-reducible.

(4) Just note that if a language is 1-tt-self-reducible, then the self-reduction tree of the corresponding self-reduction machine M on an input w is in fact a path whose length is bounded by a polynomial in $|w|$. This path can be computed by a deterministic Turing machine (where the label of a node's successor is computed by simulating M on the node's label). Having this path we can determine for each node v from the leaf to the root whether v's label belongs to L or not. Therefore, this language is in P. □

Chapter 3

Partial Information

Let us consider the following problem. For a language L we have to decide which of m given words $w_1, w_2, \ldots, w_m$ belong to L and which do not. If L is in P then this problem can be solved in polynomial time by running for each word the corresponding polynomial-time algorithm. However, many interesting languages such as SAT, QBF-SAT or GI are not known to be in P.

In some cases we might be happy to compute some kind of *partial information* about the membership of these words in L. For instance, a clever *partial information algorithm* might find out that one word is in L if another one is, or that under no circumstances it can be the case that exactly two of these words are in L. Either this information is sufficient to solve the underlying problem or we could try to infer further information from it. For instance, in Chapter 5 we show that if L is self-reducible then partial information algorithms of specific types can be used to decide membership of all given words in L.

When we talk about partial information we have to describe how it is represented. In his PhD-thesis, Nickelsen [Nic01] employs sets of bitstrings for this purpose. A set of bitstrings can be seen as a *pool* of possibilities for the characteristic string of $(w_1, w_2, \ldots, w_m)$ with respect to L. In this framework, a partial information algorithm is a function that computes on input of m words a pool that contains the correct characteristic string. The collection of all pools computed by such an algorithm is called a *family*. Families also describe types of partial information. Finally, for every family $\mathcal{F}$ he defines a *partial information class* $\mathrm{P}[\mathcal{F}]$ that contains all languages with partial information algorithms that output only pools of this family.

The purpose of this chapter is to give an introduction into the theory of partial information and partial information classes as presented in Nickelsen's PhD-thesis [HN93]. All definitions and results in this chapter are taken from his thesis unless otherwise stated. The chapter is structured as follows. In Section 3.1, we introduce pools, fami-

lies, and partial information classes. In Section 3.2, we mention some basic families and look at the corresponding partial information classes. In Section 3.3, normal forms of families are introduced. An important result is that every family has a family in normal form which produces the same class. Also, we mention some families that are defined using notions related to normal forms. Finally, in Section 3.4, we present known results on the inclusion structure of partial information classes. We close this section with a diagram that displays all 2-families in normal form and their inclusion structure.

3.1 Pools, Families, and Classes

In this section we show how to model partial information by pools, and types of partial information by families. We also introduce partial information classes.

Whenever we talk about partial information we mean partial information on the membership of a number of words in some language. For this, recall the problem from the introduction where m words $w_1, w_2, \ldots, w_m$ and a language L are given and it is asked which of these words belong to L and which of these do not. If nothing is known about these words and L, there are a priori 2^m bitstrings that could match the characteristic string $\chi_L(w_1, w_2, \ldots, w_m)$, namely the bitstrings in $\mathbb{B}^m$. Excluding bitstrings that can not be correct characteristic strings gives us some kind of partial information about this string. The set of bitstrings that remain possible make up a *pool* for the correct characteristic string.

Definition 3.1.1 (pool). For $m \in \mathbb{N}^+$, an *m-pool* is a subset of $\mathbb{B}^m$. If $(w_1, w_2, \ldots, w_m)$ is a tuple of words and L is a language then an m-pool is said to be a *pool for $(w_1, w_2, \ldots, w_m)$ and L* if it contains $\chi_L(w_1, w_2, \ldots, w_m)$.

Having introduced pools we want to give a convenient notation for them – the *stacking notation*. Instead of writing the bitstrings in the pool one behind the other, we stack them. For example, the pool $\{00, 01, 11\}$ is written in this notation as $\left\{ \begin{smallmatrix} 00 \\ 01 \\ 11 \end{smallmatrix} \right\}$. In light of this notation it makes sense to talk about the *i-th column* of a pool P. This column refers to the i-th position of each bitstring in P.

As we have seen, partial information for m words and a language L is modeled by m-pools. A certain type of partial information is modeled by a family of pools. Such a family gathers all pools that are necessary and sufficient to describe this type of partial information. For instance, the type "there exists a $k \in [m]$ such that the number of words from $w_1, w_2, \ldots, w_m$ in L is not k" is described by a family which contains only m-pools with bitstrings that do not contain exactly k 1-bits for some $k \in [m]$.

Families are also considered as collections of pools that a specific partial information algorithm is allowed to output. As we will see soon, a partial information algorithm for a language L is required to output only pools which contain the correct characteristic value of the input words with respect to L. Since for every possible bitstring b there is a sequence of words whose characteristic string equals b we additionally require that a family must contain for every such bitstring a pool that contains this bitstring.

Definition 3.1.2 (family). For $m \in \mathbb{N}^+$, an *m-family* $\mathcal{F}$ is a set of m-pools such that for every bitstring $b \in \mathbb{B}^m$ there exists at least one pool in $\mathcal{F}$ that contains b. We also say that $\mathcal{F}$ is an *m-ary family* and that m is the *arity* or *tuple length*.

For each type of partial information modeled by some family $\mathcal{F}$ we can now define a corresponding class of languages which have partial information algorithms over $\mathcal{F}$. We call such a class a *partial information class* and a language in this class a *partial information language*.

Definition 3.1.3 (partial information class). Let $\mathcal{F}$ be an arbitrary m-family for some $m \in \mathbb{N}^+$.

- A language L is in the *partial information class* $\mathrm{P}[\mathcal{F}]$ if there exists an m-ary function $f \in \mathrm{FP}$ such that for all m-tuples $(w_1, w_2, \ldots, w_m)$ of words, $f(w_1, w_2, \ldots, w_m)$ is a pool for $(w_1, w_2, \ldots, w_m)$ and L.
- A language L is in the *partial information class* $\mathrm{P}_{\mathrm{dist}}[\mathcal{F}]$ if there exists an m-ary function $f \in \mathrm{FP}$ such that for all m-tuples $(w_1, w_2, \ldots, w_m)$ of pairwise distinct words, $f(w_1, w_2, \ldots, w_m)$ is a pool for $(w_1, w_2, \ldots, w_m)$ and L.

In both cases, L is called a *partial information language* and the function f a *partial information function* (over $\mathcal{F}$).

In [Nic01], subset complete families are used as a standard form for families. A family $\mathcal{F}$ is *subset complete* if every subset of a pool in $\mathcal{F}$ is already contained in $\mathcal{F}$. Subset complete families are sometimes more convenient when analyzing certain properties of partial information classes. However, sometimes, especially in proofs, it is convenient to have partial information algorithms that only output maximal pools.

Definition 3.1.4 (maximal pool). Let $\mathcal{F}$ be an m-family for some $m \in \mathbb{N}^+$. A *maximal pool* from $\mathcal{F}$ is a pool $P \in \mathcal{F}$ such that there exists no pool $P' \in \mathcal{F}$ with $P \subsetneq P'$.

For every partial information language there exists a partial information function that outputs only maximal pools from the corresponding family. To see this, we consider a language L in $\mathrm{P}[\mathcal{F}]$ for some family $\mathcal{F}$. By Definition 3.1.3 there exists a partial information function f for L that computes partial information of type $\mathcal{F}$. We convert this function into a function f' that outputs only maximal pools as follows: If $P = f(w_1, w_2, \ldots, w_m)$ is no maximal pool then we search for a maximal pool P' in $\mathcal{F}$ that is a superset of P and output this maximal pool instead of P. Clearly, P' is a pool for $(w_1, w_2, \ldots, w_m)$ and L.

3.2 Basic Families

In this section we look at some interesting families and their corresponding partial information classes. All of these families and classes are covered in more detail in Nickelsen's PhD-thesis [Nic01].

We start by considering SIZE-families. Although these families contain only pools that are bounded in their size they are expressive enough to capture many types of partial information studied already in the literature, for example approximability and p-cheatability. We then proceed with SEL-families where the corresponding partial information classes equal the class of p-selective languages. Finally, we look at the families CARD and NONSEL. The first one represents partial information on the cardinality of words in a language whereas the second one is a special superset of CARD.

Throughout this thesis we denote a family by writing its name in capital letters along with an index m that indicates that the family is an m-family. Some families such as the SIZE-families need another parameter k which is written in front of the name followed by a dash, for example k–SIZE_m. This differs slightly from the notation used in [Nic01] and [Tan99].

3.2.1 SIZE-Families

A SIZE-family contains pools that have at most k bitstrings for some constant k given as parameter to the family. So each pool excludes at least $2^m - k$ out of 2^m possibilities for the characteristic string, where m is the arity of the family.

Definition 3.2.1 (k–$\mathbf{SIZE}_m$). For fixed $m, k \in \mathbb{N}^+$ let k–SIZE_m be the m-family consisting of all m-pools of size at most k.

Languages which admit computation of partial information of type k–SIZE$_m$ for some m and k were studied extensively in the literature. Beigel, Kummer, and Stephan introduced them under the notion of $(k, m)_p$-verbose languages [BKS95a, BKS95b], where the parameters k and m are as in Definition 3.2.1.

It is easy to see that the class P[1–SIZE$_m$] equals P for every possible $m \in \mathbb{N}^+$. This is because a polynomial-time partial information algorithm over 1–SIZE$_m$ must output pools which contain exactly the correct characteristic string. On the other hand, if a language is in P then the corresponding polynomial-time algorithm can be used to decide all given words and to output an appropriate pool for them. Pools in $(2^m - 1)$–SIZE$_m$ give the minimum partial information we can get for characteristic strings of length m. Languages that are in P[$(2^m - 1)$–SIZE$_m$] for some $m \in \mathbb{N}^+$ are called *approximable*. Languages that are not approximable are called *p-superterse*.

3.2.2 CHEAT-Families

Another interesting family is CHEAT$_m$ which is defined by m–SIZE$_m$. Languages in CHEAT$_m$ are called *m-cheatable*, and if a language is m-cheatable for some $m \in \mathbb{N}^+$ then it is called *cheatable*. Beigel introduced cheatable languages in his PhD-thesis [Bei87], although not under this definition. Under his definition, a language L is 2^m-cheatable if there exists a language K such that $\chi_L(w_1, w_2, \ldots, w_{2^m})$ can be computed in polynomial-time with at most m queries to K. So we "cheat" while computing this value, which motivates the name cheatable.

There are two maximal pools in CHEAT$_2$ that have special names. The first one is the *equivalence pool* $\text{equ}_2 := \{ \begin{smallmatrix} 00 \\ 11 \end{smallmatrix} \}$. If equ_2 is a pool for a tuple (w_1, w_2) of words and a language L, then the partial information represented by this pool is that w_1 is in L if and only if w_2 is in L. The second one is the *xor-pool* $\text{xor}_2 := \{ \begin{smallmatrix} 01 \\ 10 \end{smallmatrix} \}$. If xor_2 is a pool for (w_1, w_2) and L, then the partial information represented is that w_1 is in L if and only if w_2 is not in L.

3.2.3 SEL-Families

In 1979, Selman [Sel79] introduced p-selective languages. For a p-selective language L there exists a polynomial-time computable function that *selects* on input of two words w and u a word such that if at least one of the words, w and u, is in L then the selected word is in L, too. This function is therefore called a *selector for L*. If this selector outputs u then we have $w \in L$ implies $u \in L$, and if it outputs w we have $u \in L$ implies

$w \in L$. Equivalently the function could output the pool $\{00, 01, 11\}$ in the first case, and the pool $\{00, 10, 11\}$ in the second case. These two pools are the maximal pools of the family SEL_2.

Definition 3.2.2 ($\mathbf{SEL}_2$). Let $\mathrm{sel}_2 := \{00, 01, 11\}$ denote the *selectivity pool*, and let $\text{co-sel}_2 := \{00, 10, 11\}$ denote its negation. The 2-family SEL_2 contains all 2-pools that are subsets of sel_2 or co-sel_2.

As argued above, every p-selective language is in $\mathrm{P}[\mathrm{SEL}_2]$. The converse is also true: Let L be a language in $\mathrm{P}[\mathrm{SEL}_2]$ and f the corresponding partial information function then on input (w, u), if $f(w, u)$ is a subset of sel_2 we output u, and if it is a subset of co-sel_2 we output w[1]. Therefore we call languages in $\mathrm{P}[\mathrm{SEL}_2]$ p-selective and denote the class of all such languages by P–sel. As pointed out by Selman [Sel79], the class P–sel contains arbitrarily complex languages. In particular, it contains non-recursive languages.

Fact 3.2.3. *The class* P–sel *contains non-recursive languages.*

If sel_2 is a pool for a tuple (w_1, w_2) of words and a language L, then $w_1 \in L$ implies $w_2 \in L$. If instead co-sel_2 is a pool for this tuple and L, then $w_1 \in L$. In general, a maximal pool from SEL_2 can be interpreted as a re-ordering of the two words w_1 and w_2 such that if w_i is smaller as w_j with respect to this ordering, then $w_i \in L$ implies $w_j \in L$. This idea can be extended to more than two words: For an m-tuple $(w_1, w_2, \ldots, w_m)$ we want to compute a re-ordering of these words such that if a word w_i is smaller in this ordering than w_j, then $w_i \in L$ implies $w_j \in L$. This type of partial information is represented by ascending chains.

Definition 3.2.4 (ascending chain, $\mathbf{SEL}_m$). A subset $\{b_0, b_1, b_2, \ldots, b_m\}$ of $\mathbb{B}^m$ is called an *ascending chain* if $b_0 = 0^m$, and for all $i \in [m]$ the bitstring b_i is obtained from b_{i-1} by flipping one 0 to 1. For fixed $m \in \mathbb{N}^+$, the family SEL_m contains all m-pools that are subsets of some ascending chain.

As it turns out it holds that for $m > 2$ the classes $\mathrm{P}[\mathrm{SEL}_2]$ and $\mathrm{P}[\mathrm{SEL}_m]$ are equivalent, and so P–sel $= \mathrm{P}[\mathrm{SEL}_m]$ for all $m \geq 2$. This will be more clearer in Section 3.4, when we discuss translations of families to other tuple lengths.

[1] If $f(w, u)$ is a subset of both pools then we can output either w or u.

3.2.4 CARD- and NONSEL-Families

An important type of partial information is that of the number of m given words $w_1, w_2, \ldots, w_m$ in a language L, i.e. partial information on $\#_L^m$. This number is deeply connected with the characteristic string of these words with respect to L, since we have $\#_L^m = \#_1 \circ \chi_L^m$. So, a pool for $(w_1, w_2, \ldots, w_m)$ and L that contains no bitstrings with exactly k 1-bits excludes the possibility that $\#_L(w_1, w_2, \ldots, w_m)$ equals k. The families k–CARD$_m$ contain only pools such that at most k of the values of $\#_L^m$ remain possible, i.e. at least $m - k + 1$ values are excluded.

Definition 3.2.5 (k–CARD$_m$). For fixed $m \in \mathbb{N}^+$ and $k \in [m+1]$ the m-family k–CARD$_m$ contains all m-pools P for which there is a set $N \subseteq \{0, 1, 2, \ldots, m\}$ of size at most k such that for all bitstrings $b \in P$ it holds that $\#_1(b) \in N$.

Languages in $\mathrm{P}_{\mathrm{dist}}[m\text{–CARD}_m]$ are called *easily m-countable*. This name is motivated by the fact that, if $m = 2^k$ for some $k \in \mathbb{N}^+$ then there exists a language K such that $\#_L^m$ can be computed in polynomial time with at most k queries to K (see [Nic01, p. 19]). A language that is easily m-countable for some m is called *easily countable*.

Kummer [Kum92] proved that every easily countable language is recursive. This result was extended by Nickelsen [Nic01, Theorem 4.20] to languages in the larger class P[NONSEL$_m$]. He also noted that NONSEL$_m$ is the largest m-family $\mathcal{F}$ in normal form [2] such that P[$\mathcal{F}$] contains only recursive languages.

Definition 3.2.6 (NONSEL$_m$). For fixed $m \in \mathbb{N}^+$ the m-family NONSEL$_m$ contains all pools that are no supersets of an ascending chain.

The name NONSEL$_m$ stems from the fact that it is the largest m-family in normal form that is no superset of SEL$_m$. The relation of m–CARD$_m$ to NONSEL$_m$ is as follows: For the case that m is equal to 2 both families are equal, but the first family is strictly included in the second one for any m larger than 2. The latter relation holds because for $m \geq 3$ the family NONSEL$_m$ contains the pool $\{0^{m-i}1^i \mid i \neq 1\} \cup \{10^{m-1}\}$ that is no ascending chain and contains for every $k \in \{0, 1, 2, \ldots, m\}$ a bitstring with exactly k 1-bits. So, this pool can not be contained in m–CARD$_m$.

3.3 Normal Forms

In this section we introduce the notion of a family being in normal form. Families in normal form are very convenient in the study of the inclusion structure of partial

[2] We will introduce normal forms in the next section.

information classes. For instance, as stated in the next section, for any two m-families $\mathcal{F}_1$ and $\mathcal{F}_2$ in normal form we have $\mathrm{P}[\mathcal{F}_1] \subseteq \mathrm{P}[\mathcal{F}_2]$ if and only if $\mathcal{F}_1 \subseteq \mathcal{F}_2$, and $\mathrm{P}[\mathcal{F}_1] = \mathrm{P}[\mathcal{F}_2]$ if and only if $\mathcal{F}_1 = \mathcal{F}_2$. Moreover, families in normal form are already sufficient to produce all possible partial information classes.

To define normal forms we first have to define what it means for a family to be closed under permutations, projections and copy operations.

Definition 3.3.1 (permutation, projection, copy operation). Let $\mathcal{F}$ be an arbitrary m-family for some $m \in \mathbb{N}^+$.

1. If σ is a permutation of $(1, 2, \ldots, m)$ and b is a bitstring of length m, then

   $$\sigma(b) := b[\sigma(1), \sigma(2), \ldots, \sigma(m)].$$

   For all m-pools P we define $\sigma(P) := \{\sigma(b) \mid b \in P\}$. The family $\mathcal{F}$ is *closed under permutations* if and only if for all permutations σ of $(1, 2, \ldots, m)$ and for all pools $P \in \mathcal{F}$ we have $\sigma(P) \in \mathcal{F}$.

2. For a bitstring b of length m, an index $i \in [m]$ and a bit $c \in \mathbb{B}$ we define the projection of c on i by

   $$\pi_i^c(b) := b[1] \ldots b[i-1]cb[i+1] \ldots b[m].$$

   For all m-pools P we define $\pi_i^c(P) := \{\pi_i^c(b) \mid b \in P\}$. The family $\mathcal{F}$ is *closed under projections* if and only if for all indices $i \in [m]$, bits $c \in \mathbb{B}$ and pools $P \in \mathcal{F}$ we have $\pi_i^c(P) \in \mathcal{F}$.

3. For a bitstring b of length m and two indices $i, j \in [m]$ we define a copy operation $\rho_{i,j}$ by

   $$\rho_{i,j}(b) = b'_1 b'_2 \ldots b'_m,$$

   where $b'_j = b[i]$, and for all $k \in [m]$ with $k \neq j$ it holds that $b'_k = b[k]$. For all m-pools P we define $\rho_{i,j}(P) := \{\rho_{i,j}(b) \mid b \in P\}$. The family $\mathcal{F}$ is *closed under copy operations* if and only if for all indices $i, j \in [m]$ and pools $P \in \mathcal{F}$ we have $\rho_{i,j}(P) \in \mathcal{F}$.

Now we can define what it means for a family to be in normal form.

Definition 3.3.2 (normal form). A family is in *normal form* if it is subset complete and closed under permutations, projections and copy operations.

Before we proceed, we mention a nice property of families in normal form taken from [Nic01].

Fact 3.3.3. *If $\mathcal{F}$ is in normal form, then* $\mathrm{P}_{\mathrm{dist}}[\mathcal{F}] = \mathrm{P}[\mathcal{F}]$.

All families introduced in the last section are in normal form, except the CARD-families which are in general not in normal form but subset complete. However, the following theorem states that every language that is subset complete has a family in normal form that produces the same partial information class. Moreover, there is exactly one such family. This has been proved by Nickelsen in [Nic01, Theorem 2.27 and Corollary 2.31].

Fact 3.3.4. *For all subset complete families $\mathcal{F}$ there exists exactly one family $\mathcal{F}'$ in normal form such that $\mathcal{F}' \subseteq \mathcal{F}$ and* $\mathrm{P}[\mathcal{F}'] = \mathrm{P}[\mathcal{F}]$.

So, in general we can restrict our studies on partial information classes to classes that are produced by families in normal form. An easy way to define families in normal form is to use generators.

Definition 3.3.5 (generated family). For a sequence of m-pools $P_1, P_2, \ldots, P_n$ let $\mathcal{F} = \langle P_1, P_2, \ldots, P_n \rangle$ denote the smallest m-family in normal form that contains all these pools. The family $\mathcal{F}$ is called the *family generated by* $P_1, P_2, \ldots, P_n$.

The family SEL_2 can now be described by $\langle \mathrm{sel}_2 \rangle$ which can be generalized to the families SEL_m by $\langle \{0^i 1^{m-i} \mid 0 \leq i \leq m\} \rangle$. Furthermore we have 2^m–$\mathrm{SIZE}_m = \langle \mathbb{B}^m \rangle$. Two interesting families which are especially easy to define using generators are BOTTOM_m and TOP_m which represent partial information of the type "at most one word is in the language" and "at least one word is in the language".

Definition 3.3.6 (BOTTOM_m, TOP_m). For fixed $m \in \mathbb{N}^+$, let bottom_m be defined by $\{b \in \mathbb{B}^m \mid \#_1(b) \leq 1\}$, and top_m by $\{b \in \mathbb{B}^m \mid \#_0(b) \leq 1\}$. We define $\mathrm{BOTTOM}_m := \langle \mathrm{bottom}_m \rangle$ and $\mathrm{TOP}_m := \langle \mathrm{top}_m \rangle$.

It is not hard to see that 2–CARD_2 is the union of BOTTOM_2 and TOP_2, and that it can be generated from the two pools bottom_2 and top_2. With the help of BOTTOM- and TOP-families we can even describe the class of strongly m-membership comparable languages and its complement for fixed m. Such languages were defined by Köbler in [Köb95], however in a more general form with m replaced by a special function from $\mathbb{N}$ to $\mathbb{N}^+$.

Definition 3.3.7 (SMC_m, COSMC_m). For fixed $m \in \mathbb{N}$ let SMC_m denote the m-family containing all m-pools that are no supersets of top_m, and let COSMC_m denote the m-family containing all m-pools that are no supersets of bottom_m.

The languages in $\mathrm{P}[\mathrm{SMC}_m]$ are exactly the strongly m-membership comparable languages, those in $\mathrm{P}[\mathrm{COSMC}_m]$ the complements of strongly m-membership comparable languages. We observe that $\mathrm{SMC}_2 = \mathrm{SEL}_2 \cup \mathrm{BOTTOM}_2$ and $\mathrm{COSMC}_2 = \mathrm{SEL}_2 \cup \mathrm{TOP}_2$. The intersection $\mathrm{SMC}_2 \cap \mathrm{COSMC}_2$ of both families forms the family $\mathrm{SEL}_2 \cup \{\mathrm{xor}_2\} = \mathrm{SEL}_2 \cup \mathrm{CHEAT}_2$.

3.4 Inclusion Structure

This last section deals with the inclusion structure of partial information classes. We first consider inclusions between classes produced by families of the same arity. That is, given two m-families $\mathcal{F}_1$ and $\mathcal{F}_2$, we present anwers for the question whether $\mathrm{P}[\mathcal{F}_1]$ is included in $\mathrm{P}[\mathcal{F}_2]$ or whether both classes are equal. Then we will consider inclusions between partial information classes produced by families of different arity. We conclude with a diagram that displays the inclusion structure of all partial information classes produced by 2-families in normal form.

Given two m-families $\mathcal{F}_1$ and $\mathcal{F}_2$ with $\mathcal{F}_1 \subseteq \mathcal{F}_2$ it is easy to see that $\mathrm{P}[\mathcal{F}_1] \subseteq \mathrm{P}[\mathcal{F}_2]$. Furthermore, if $\mathcal{F}_1 = \mathcal{F}_2$, then it follows that $\mathrm{P}[\mathcal{F}_1] = \mathrm{P}[\mathcal{F}_2]$. If both families are in normal form even the converse holds: Nickelsen gives in [Nic01, p. 24f] a nice criterion when we can say that $\mathrm{P}[\mathcal{F}_1]$ is included in $\mathrm{P}[\mathcal{F}_2]$, and when we can say that both classes are equal. We state his result in the following fact.

Fact 3.4.1. *For all m-families $\mathcal{F}_1$ and $\mathcal{F}_2$ in normal form we have:*

- $\mathrm{P}[\mathcal{F}_1] \subseteq \mathrm{P}[\mathcal{F}_2]$ *if and only if* $\mathcal{F}_1 \subseteq \mathcal{F}_2$.
- $\mathrm{P}[\mathcal{F}_1] = \mathrm{P}[\mathcal{F}_2]$ *if and only if* $\mathcal{F}_1 = \mathcal{F}_2$.

Sometimes we want to compare partial information classes that are produced by families of different arity. For instance, if we have two families $\mathcal{F}_1$ and $\mathcal{F}_2$ with different arity we could ask whether $\mathrm{P}[\mathcal{F}_1]$ is included in $\mathrm{P}[\mathcal{F}_2]$ or whether both classes are equal. To solve this problem we could transform one family to a family with the same arity as the other one and then use the criterion given in Fact 3.4.1. In general, we will transform the family with the smaller arity to a family with a larger arity. Such a transformation is called an *upward translation*. For the sake of completeness we also give the definition of downward translation.

Definition 3.4.2 (upward and downward translation). The *upward translation* of an m-family $\mathcal{F}$ to tuple length $n \geq m$, denoted by $\lceil \mathcal{F} \rceil_n$, is an n-family such that for all pools $P \in \lceil \mathcal{F} \rceil_n$ and for all numbers $i_1, i_2, \ldots, i_m$ with $1 \leq i_1 < i_2 < \ldots < i_m \leq n$ it holds that $P[i_1, i_2, \ldots, i_m] \in \mathcal{F}$. The *downward translation* of $\mathcal{F}$ to tuple length $n \in \mathbb{N}^+$ with $n < m$ is defined by $\lceil \mathcal{F} \rceil_n := \{P[1, 2, \ldots, n] \mid P \in \mathcal{F}\}$.

The following theorem which is once more taken from [Nic01, p. 32f] shows that we are not wrong in solving the inclusion problem for families of different arity as described above. It states that upward translations do not change the corresponding partial information classes and preserve normal forms.

Fact 3.4.3. *Let $\mathcal{F}$ be an arbitrary subset complete m-family for some $m \in \mathbb{N}^+$. Then, for all $n \geq m$ it holds that $\mathrm{P}[\mathcal{F}] = \mathrm{P}[\lceil \mathcal{F} \rceil_n]$, and if $\mathcal{F}$ is in normal form then also $\lceil \mathcal{F} \rceil_n$ is in normal form.*

We will now consider upward translations of families already defined in the last two sections. All of the following results can be found in more detail in [Nic01, p. 37ff]. The statement of the first item has also been obtained implicitly by Selman in [Sel79].

Fact 3.4.4. *For all $n \geq 2$ the following statements hold.*

- $\lceil \mathrm{SEL}_2 \rceil_m = \mathrm{SEL}_m$.
- $\lceil \mathrm{BOTTOM}_2 \rceil_m = \mathrm{BOTTOM}_m$.
- $\lceil \mathrm{TOP}_2 \rceil_m = \mathrm{TOP}_m$.
- $\lceil \mathrm{CHEAT}_2 \rceil_m = 2\text{–}\mathrm{SIZE}_m$.

In Section 3.2 we stated that P–sel = P[SEL_m] for every $m \geq 2$. This can now be easily proved by the two preceding theorems: They are equal because P–sel = P[SEL_2] = P[$\lceil \mathrm{SEL}_2 \rceil_m$] = P[$\mathrm{SEL}_m$] for every $m \geq 2$. Similarly we get P[BOTTOM_2] = P[BOTTOM_m] and P[TOP_2] = P[TOP_m] for every $m \geq 2$.

Suppose we know that a language L is in P[$\mathcal{F}$] for some m-family $\mathcal{F}$. An interesting question is whether we can compute for $n \in \mathbb{N}^+$ given words in polynomial time a pool from $\lceil \mathcal{F} \rceil_n$ for these words and L. This question is answered positively in [Nic01, p. 42]. We state the corresponding theorem with an extension to downward translations.

Fact 3.4.5. *Let L be a language in $\mathrm{P}[\mathcal{F}]$ for some m-family $\mathcal{F}$ in normal form. Then there is a function $f \in \mathrm{FP}$ such that for every $n \in \mathbb{N}^+$ and every tuple $(w_1, w_2, \ldots, w_n)$ of words, $f(w_1, w_2, \ldots, w_n)$ is a pool from $\lceil \mathcal{F} \rceil_n$ for $(w_1, w_2, \ldots, w_n)$ and L.*

We finish this section with Figure 3.1 that shows all partial information classes produced by 2-families in normal form and its inclusion structure. Each class is represented by the 2-family that produces it. An arc from a family $\mathcal{F}_1$ to a family $\mathcal{F}_2$ denotes strict inclusion of $\mathcal{F}_1$ in $\mathcal{F}_2$, and due to the results in the last section also strict inclusion of $\mathrm{P}[\mathcal{F}_1]$ in $\mathrm{P}[\mathcal{F}_2]$. The family $\langle\{00, 01\}\rangle$ is usually called MIN_2 and represents the maximum partial information we can get on two words without actually deciding them. It is interesting to note that all classes on the left side contain non-recursive languages whereas those on the right side do not.

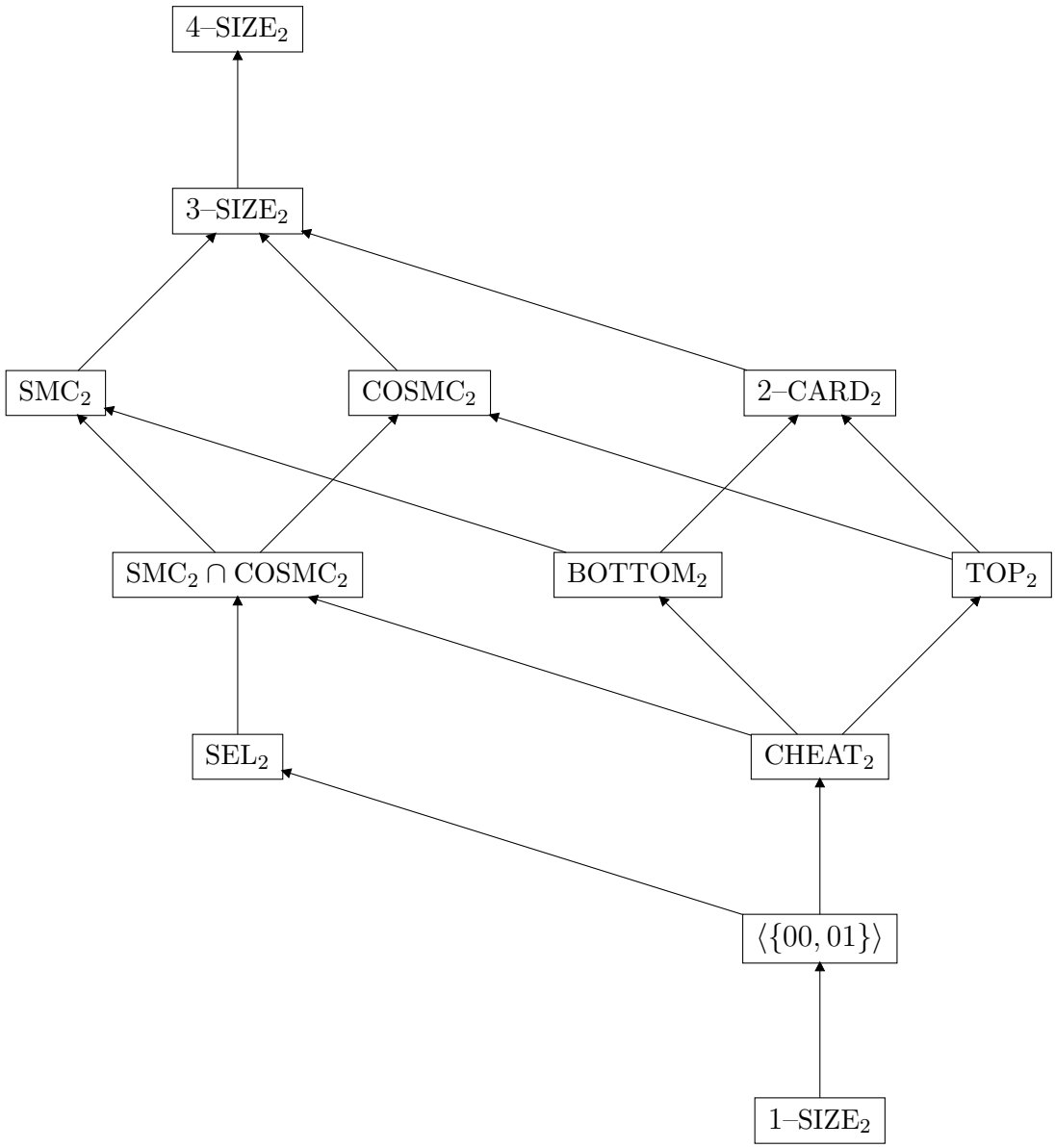


Figure 3.1: Inclusion structure of partial information classes produced by 2-families in normal form. Each class is represented by the corresponding 2-family. If there is an arrow from a family $\mathcal{F}_1$ to another family $\mathcal{F}_2$, then $\mathrm{P}[\mathcal{F}_1]$ is strictly included in $\mathrm{P}[\mathcal{F}_2]$.

Chapter 4

Reduction Closures of Partial Information Classes

To study the effect of partial information on self-reductions, it is reasonable to look at reductions in general first. We will consider in this chapter closures of partial information classes under various reducibilities. Given a family $\mathcal{F}$ and a reducibility $\leq_r^{\mathsf{p}}$, we are especially interested in answers to the following two questions.

1. Is $\mathrm{P}[\mathcal{F}]$ closed under $\leq_r^{\mathsf{p}}$-reducibility?

2. For another reducibility $\leq_{r'}^{\mathsf{p}}$, do the closures of $\mathrm{P}[\mathcal{F}]$ under $\leq_r^{\mathsf{p}}$-reducibility and under $\leq_{r'}^{\mathsf{p}}$-reducibility coincide?

Answers to these questions often yield results for self-reductions. For example, if we know that self-reducible languages in $\mathrm{P}[\mathcal{F}]$ belong to P, and that $\mathrm{P}[\mathcal{F}]$ is closed under $\leq_r^{\mathsf{p}}$-reducibility, then also every self-reducible language $\leq_r^{\mathsf{p}}$-reducible to some language in $\mathrm{P}[\mathcal{F}]$ belongs to P. In case that the $\leq_r^{\mathsf{p}}$-closure of $\mathrm{P}[\mathcal{F}]$ coincides with the $\leq_{r'}^{\mathsf{p}}$-closure, we can sometimes conclude that r-self-reducible languages in $\mathrm{P}[\mathcal{F}]$ are exactly the r'-self-reducible languages in $\mathrm{P}[\mathcal{F}]$.

We start in Section 4.1 by considering closure of partial information classes under basic reducibilities such as many-one, 1-tt and Turing reducibility. Often there are good characterizations of those families $\mathcal{F}$ in normal form for which $\mathrm{P}[\mathcal{F}]$ is closed under one of these reducibilities. For example, Nickelsen [Nic01] showed that all partial information classes produced by families in normal form are closed under many-one reducibility. Although this is not the case for 1-tt and Turing reducibility, he also obtained good characterizations for families in normal form that produce partial information classes closed under 1-tt and Turing reducibility, respectively.

In Section 4.2, we then consider positive reducibilities. An interesting result that has been obtained in [BTvEB93] says that positive Turing reductions to p-selective languages are not more powerful than many-one reductions. Nickelsen [Nic01] showed that a similar result holds for the class $\mathrm{P}[\mathrm{SMC}_2 \cap \mathrm{COSMC}_2]$ when "Turing" is replaced by "truth-table". From these results it follows that both classes are closed under positive Turing and positive truth-table reducibility, respectively. This is surprising because they are even not closed under 1-tt-reducibility.

In Section 4.3, we show that every language that is Turing reducible to a language L in $\mathrm{P}[\mathrm{NONSEL}_m]$ is already truth-table reducible to L. This extends the corresponding result for easily m-countable languages obtained by Beigel, Kummer, and Stephan in [BKS95a]. Moreover, combined with a result of Toda [Tod91], this gives a characterization of the families $\mathcal{F}$ in normal form for which Turing and truth-table closures of $\mathrm{P}[\mathcal{F}]$ coincide.

4.1 Closure under Basic Reducibilities

In this section, we consider closure of partial information classes under many-one, 1-tt, 2-tt, truth-table, and Turing reducibility. For each of these reducibilities, except for truth-table reducibility, there are good characterizations of the families $\mathcal{F}$ in normal form for which $\mathrm{P}[\mathcal{F}]$ is closed under that reducibility. The following results and definitions are taken from [Nic01, Section 3.2].

We start with closure under many-one and 1-tt-reducibility. All partial information classes produced by families in normal form are closed under many-one reducibility. Families $\mathcal{F}$ for which $\mathrm{P}[\mathcal{F}]$ is closed under 1-tt-reducibility are characterized as families in normal form closed under bit-flip.

Definition 4.1.1 (bit-flip, closed under bit-flip). For a bitstring $b \in \mathbb{B}^m$ and an index $i \in [m]$, we define the *bit-flip* at the i-th position of b by

$$\mathrm{flip}_i(b) := b[1] \dots b[i-1](1 - b[i])b[i+1] \dots b[m].$$

We extend this to m-pools P by $\mathrm{flip}_i(P) := \{\mathrm{flip}_i(b) \mid b \in P\}$. An m-family $\mathcal{F}$ is *closed under bit-flip* if and only if for all indices $i \in [m]$ and for all pools $P \in \mathcal{F}$ we have $\mathrm{flip}_i(P) \in \mathcal{F}$.

We summarize the characterizations given above for many-one and 1-tt-reducibility in the next fact.

Fact 4.1.2. *Let $\mathcal{F}$ be a family in normal form. Then:*

- $\mathrm{P}[\mathcal{F}]$ *is closed under many-one reducibility.*
- $\mathrm{P}[\mathcal{F}]$ *is closed under 1-tt-reducibility if and only if* $\mathcal{F}$ *is closed under bit-flip.*

A consequence is that several classes introduced in Chapter 3 are closed under many-one reducibility, but not under 1-tt-reducibility. For example, consider the class $\mathrm{P}[\mathrm{SEL}_2]$. The pool $\mathrm{sel}_2 = \{00, 01, 11\}$ is contained in SEL_2, but $\mathrm{flip}_1(\mathrm{sel}_2) = \{10, 11, 01\} = \mathrm{top}_2$ is not. Therefore, SEL_2 is not closed under bit-flip and hence not under 1-tt-reducibility. On the other hand, $\mathrm{P}[k\text{–}\mathrm{SIZE}_m]$ is closed under 1-tt-reducibility for all $m, k \in \mathbb{N}^+$, because bit-flipping does not increase the size of pools.

For 2-tt and Turing reducibility, there is the following characterization which is a bit surprising, because it states among other things that $\mathrm{P}[\mathcal{F}]$ is closed under 2-tt-reducibility if and only if it is closed under Turing reducibility.

Fact 4.1.3. *Let* $\mathcal{F}$ *be an m-family in normal form with* $\mathcal{F} \neq 2^m\text{–}\mathrm{SIZE}_m$*. Then, the following statements are equivalent.*

1. $\mathrm{P}[\mathcal{F}]$ *is closed under 2-tt-reducibility.*
2. $\mathrm{P}[\mathcal{F}]$ *is closed under Turing reducibility.*
3. $\mathcal{F} = k\text{–}\mathrm{SIZE}_m$ *for some* $k \leq m$*.*

Finally, for truth-table reducibility there is the following result. In the proof of Theorem 2.9 in [BKS95a], Beigel, Kummer, and Stephan constructed languages L and K with $L \leq_{\mathsf{tt}}^{\mathsf{p}} K$ such that L is not approximable and $K \in \mathrm{P}[\mathrm{TOP}_3]$. As noted in [Nic01, Section 3.2], this yields the following fact.

Fact 4.1.4. *Let* $\mathcal{F}$ *be an m-family in normal form with* $\mathcal{F} \neq 2^m\text{–}\mathrm{SIZE}_m$*. Then,* $\mathrm{P}[\mathcal{F}]$ *is not closed under tt-reducibility if* $\mathrm{BOTTOM}_m \subseteq \mathcal{F}$ *or* $\mathrm{TOP}_m \subseteq \mathcal{F}$*.*

4.2 Closure under Positive Reducibilities

As we have seen in the last section, $\mathrm{P}[\mathrm{SEL}_2]$ is closed under many-one (which is equivalent to 1-ptt-reducibility), but not under 1-tt-reducibility (see the discussion below Fact 4.1.2). The same applies to the slightly larger class $\mathrm{P}[\mathrm{SMC}_2 \cap \mathrm{COSMC}_2]$.

Is many-one reducibility the "largest" reducibility these classes are closed under? The question must be answered negatively for $\mathrm{P}[\mathrm{SEL}_2]$, because it is closed under positive Turing reducibility. This has been proved by Buhrman, Torenvliet, and van Emde Boas in [BTvEB93], where they actually obtained the following result.

Fact 4.2.1. *Let L be $\leq^{\mathsf{P}}_{\mathsf{pos}}$-reducible to a p-selective language K via some oracle Turing machine M. Then, L is many-one reducible to K via a function f such that for every word w, $f(w)$ is a word queried by M on input w with some oracle.*

From Fact 4.2.1 and the fact that all families in normal form are closed under many-one reducibility (Fact 4.1.2) we obtain the following corollary.

Corollary 4.2.2. *For all families $\mathcal{F}$ in normal form with $\mathrm{P}[\mathcal{F}] \subseteq \mathrm{P}[\mathrm{SEL}_2]$, the class $\mathrm{P}[\mathcal{F}]$ is closed under positive Turing reducibility.*

For $\mathrm{P}[\mathrm{SMC}_2 \cap \mathrm{COSMC}_2]$, there exists a result similar to Fact 4.2.1. The only difference are the classes itself and that "Turing" is replaced by "truth-table". This has been shown by Nickelsen in [Nic01, p. 66]. As for $\mathrm{P}[\mathrm{SEL}_2]$, it follows that every subclass of $\mathrm{P}[\mathrm{SMC}_2 \cap \mathrm{COSMC}_2]$ produced by a family in normal form is closed under positive truth-table reducibility, because it is closed under many-one reducibility by Fact 4.1.2. Moreover, Nickelsen gave the following characterization of 2-families in normal form that produce partial information classes closed under positive truth-table reducibility.

Fact 4.2.3. *Let $\mathcal{F}$ be a 2-family in normal form with $\mathcal{F} \neq 4\text{–}\mathrm{SIZE}_2$. Then, the following statements are equivalent.*

1. $\mathrm{P}[\mathcal{F}]$ *is closed under ptt-reducibility.*
2. $\mathrm{P}[\mathcal{F}]$ *is closed under 2-ptt-reducibility.*
3. $\mathcal{F} \subseteq \mathrm{SMC}_2 \cap \mathrm{COSMC}_2$.
4. $\mathrm{BOTTOM}_2 \nsubseteq \mathcal{F}$ *and* $\mathrm{TOP}_2 \nsubseteq \mathcal{F}$.

4.3 Converting Turing into Truth-Table Reductions

Beigel, Kummer and Stephan showed in [BKS95a] that every language that is Turing reducible to an easily countable language K is already truth-table reducible to K. In this section, we extend this result to the larger class $\mathrm{P}[\mathrm{NONSEL}_m]$.

Theorem 4.3.1. *If a language is Turing reducible to a language $K \in \mathrm{P}[\mathrm{NONSEL}_m]$, then it is truth-table reducible to K.*

We use essentially the same proof as in [BKS95a]. The phenomenon that results originally proved for easily countable languages also hold for $\mathrm{P}[\mathrm{NONSEL}_m]$ by nearly the same

proof is not new. For example, Kummer's Cardinality Theorem [Kum92] states that if for a language L there is an algorithm that enumerates for every m-tuple $(w_1, w_2, \ldots, w_m)$ of pairwise distinct words a pool from m–CARD$_m$ for $(w_1, w_2, \ldots, w_m)$ and L, then L is recursive. Nickelsen showed that m–CARD$_m$ can be replaced by NONSEL$_m$ [Nic01, p. 78] by using nearly the same proof. In Section 5.2.3, we obtain as a corollary from Theorem 4.3.1 and another result from [BKS95a] that disjunctively self-reducible languages in P[NONSEL$_m$] already belong to P. Alternatively, we could use the proof of the corresponding result for easily countable languages obtained by Hoene and Nickelsen in [HN93].

The idea for the proof of Theorem 4.3.1 is as follows. Let L be Turing reducible to a language $K \in \mathrm{P}[\mathrm{NONSEL}_m]$ via some oracle Turing machine M, and let w be an arbitrary word. We show how to compute in polynomial time a subtree T of $\mathrm{QT}_M(w)$ such that the number of nodes of T is polynomial in $|w|$. This tree contains the path determined by K, i.e. the path whose labels correspond to words queried by M on input w with oracle K. So, we can query all words in T in parallel and obtain information about the membership of each word of T in K, and therefore for each word queried by M on input w with oracle K. This allows us to convert the Turing reduction into a truth-table reduction.

Binary trees are represented in [BKS95a] as languages over $\mathbb{B}$. Based on this representation, they define embeddings and the rank of binary trees. We introduce these notions in Section 4.3.1. Binary trees with bounded rank have the nice property that the number of nodes is bounded by a polynomial in their depth. In Section 4.3.2, we introduce the width of a pool to be the largest rank of any labeled binary tree that is consistent with that pool. Query trees are labeled binary trees, so this notion allows us to bound the rank of any subtree of a query tree that is consistent with a given pool. Finally, in Section 4.3.3, we prove Theorem 4.3.1.

4.3.1 Binary Trees as Languages, Embeddings, and Rank

Let B be a rooted binary tree with the set V of nodes. We call a node $v_1 \in V$ a *descendant* of a node $v_2 \in V$ if v_2 lies on the path from the root to v_1. If v_1 is a descendant of v_2, then v_1 is called a *successor* of v_2 if the path from the root to v_1 is exactly one node longer than the path from the root to v_2. Since B is a binary tree, there are at most two successors for every node in V. Without loss of generality we assume that the edges between a node v and its successors are labeled with elements from $\mathbb{B}$ such that the label of the edge between v and one successor is different from the

label of the edge between v and the other successor, if there is any.

A node $v \in V$ can now be represented as a word b over $\mathbb{B}$ as follows. Let P be the path from the root to v and m its length. Then, b is a word over $\mathbb{B}$ of length m such that the i-th symbol in b corresponds to the label of the i-th edge on P. The language T consisting of binary representations of nodes in V as described above is then a representation of B. For example, the tree shown in Figure 4.1 can be represented by the language $T = \{\lambda, 0, 1, 01, 10, 11, 100, 101\}$.

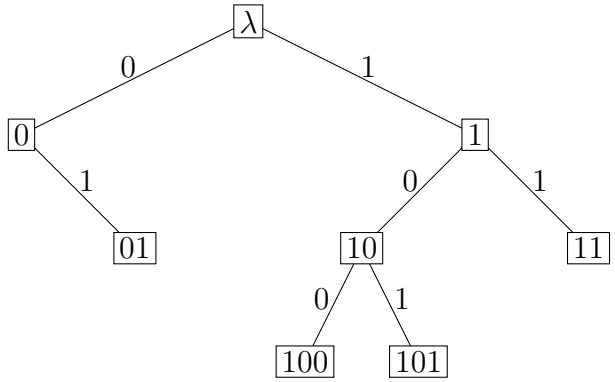


Figure 4.1: A binary tree B. The labels at the edges induce a representation of B as a language T over $\mathbb{B}$. For example, the node reachable from the root node by walking the edges labeled 1, then 0, and finally 1, is represented by the word 101. The language T is given by $\{\lambda, 0, 1, 01, 10, 11, 100, 101\}$.

In the remaining part of this section, by writing that T is a binary tree we mean that T is a language over $\mathbb{B}$ that represents the corresponding binary tree. We call the elements of a binary tree *nodes*. The *edges* are given implicitly by the prefix relation. This means, there is an edge between two nodes $v_1, v_2 \in T$ if one of these nodes, say v_i, is a prefix of the other one, v_j, and v_i is exactly one symbol shorter than v_j. Then, it is easy to see that a node v_1 is a *descendant* of a node v_2 if and only if v_2 is a prefix of v_1, and if v_1 is a descendant of v_2 then v_1 is a *successor* of v_2 if and only if v_2 is exactly one symbol shorter than v_1. The *root node* is represented by the empty word. A *leaf* is a node which is no prefix of another node in T, and an *interior node* is a node which is a prefix of at least one other node in T.

A special binary tree is the *full binary tree* of depth d, defined by $\mathrm{B}_d := \mathbb{B}^{\leq d}$. An embedding ε of B_d into a binary tree T is a mapping from B_d into T with the following property. For every interior node $v \in \mathrm{B}_d$, it maps the "left" successor $v0$ to a node in the "left" subtree of $\varepsilon(v)$, and the "right" successor $v1$ to a node in the "right" subtree.

More formally, it is defined as follows.

Definition 4.3.2 (embedding, embeddable). An *embedding* of B_d into a binary tree T is a mapping $\varepsilon\colon \mathrm{B}_d \to T$ such that for all interior nodes $v \in \mathrm{B}_d$ it holds that $\varepsilon(v0)$ is a descendant of $\varepsilon(v)0$ and $\varepsilon(v1)$ is a descendant of $\varepsilon(v)1$. We say that B_d is *embeddable* into T if there exists an embedding of B_d into T.

Example 4.3.3. Figure 4.2 shows an embedding ε of B_2 into the binary tree $T = \{\lambda, 0, 1, 01, 10, 11, 010, 011, 100\}$. An arc from a node $v_1 \in \mathrm{B}_2$ to a node $v_2 \in T$ means

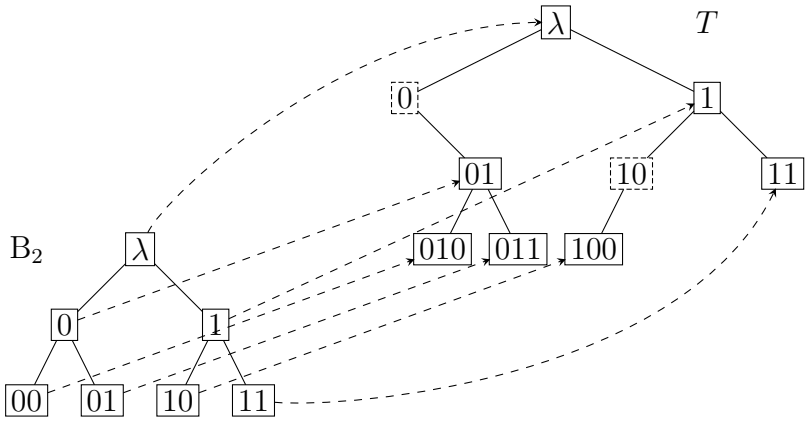


Figure 4.2: An embedding of B_2 into the tree $T = \{\lambda, 0, 1, 01, 10, 11, 010, 011, 100\}$.

that v_1 is mapped to v_2. For example, the root node of B_2 is mapped to the root node of T, the node 0 is mapped to the node 01, and the node 1 is mapped to the node 1. We observe that $\varepsilon(0) = 01$ is a descendant of $\varepsilon(\lambda)0 = 0$ and $\varepsilon(1) = 1$ is a descendant of $\varepsilon(\lambda)1 = 1$. The condition given in Definition 4.3.2 is thus satisfied for the root node of B_2. Moreover, it is satisfied for all interior nodes of B_2. △

Definition 4.3.4 (rank). The *rank* of a binary tree T, written $\mathrm{rk}(T)$, is the maximum d such that B_d is embeddable into T.

Example 4.3.5. Figure 4.3 shows binary trees with ranks 0, 1 and 2. The first tree from the left is a path and has rank zero since it has no node with two successors which would allow an embedding of B_1. The other way round, a binary tree with rank zero is a path since otherwise there is a node v with two successors $v0$ and $v1$ such that B_1 is embeddable into the subtree consisting of v, $v0$ and $v1$; hence the rank is one which contradicts the fact that it is zero. It is also easy to see that the second tree has rank

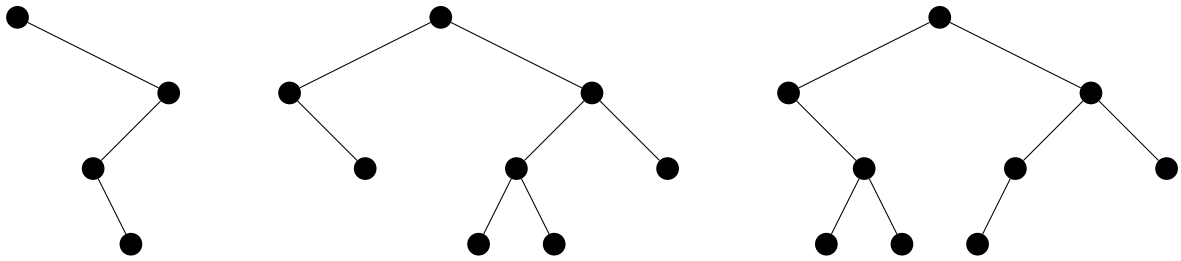


Figure 4.3: Binary trees with rank 0, 1 and 2 (from left to right).

one. Note that the third tree is the tree T shown in Figure 4.2. As it is shown there, B_2 is embeddable into T. However, there is no way to embed B_3 into it. It follows that the rank of T is two. △

As shown in [BKS95a], binary trees with bounded rank have the nice property that the number of nodes is bounded by a polynomial in its depth. We state this result in the following fact.

Fact 4.3.6. *A binary tree of depth d and rank less than r has at most $p_r(d) := \sum_{i=0}^{r-1} \binom{d+1}{i+1}$ many nodes. In particular, p_r is a polynomial in d.*

4.3.2 Bounding the Rank in Terms of Pools

We now introduce the notion of the width of a pool P. It is an upper bound on the rank of any *labeled* binary tree that is consistent with P. First, let us define labeled binary trees and what it means to be consistent with P.

Definition 4.3.7 (labeling, labeled binary trees). Let $\mathcal{L}$ be an arbitrary non-empty set. A *labeling* of a binary tree T with labels from $\mathcal{L}$ (for short, an *$\mathcal{L}$-labeling* of T) is a mapping from the interior nodes of T to $\mathcal{L}$. A binary tree together with an $\mathcal{L}$-labeling is called an *$\mathcal{L}$-labeled binary tree*. For $m \in \mathbb{N}^+$, we write *m-labeling* and *m-labeled binary tree* instead of $[m]$-labeling and $[m]$-labeled binary tree, respectively.

The query tree of a polynomial-time oracle Turing machine M on an input w is a Σ^*-labeled binary tree. We make the convention, that the representation as a language over $\mathbb{B}$ is induced by the labeling of the edges as described in Definition 2.2.2. That is, if M queries on input w with oracle L the sequence $q_1, q_2, \ldots, q_k$ of words, then the path determined by L is the path from the root to the leaf $\chi_L(q_1, q_2, \ldots, q_k)$. For example,

consider the machine M, the word w, and the query tree from Example 2.2.3. We have $\mathrm{QT}_M(w) = \{\lambda, 0, 1, 10, 11\}$ as visualized in Figure 4.4. The corresponding Σ^*-labeling φ

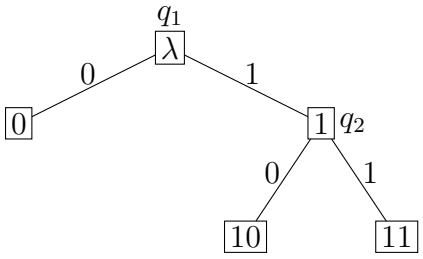


Figure 4.4: The query tree $\mathrm{QT}_M(w) = \{\lambda, 0, 1, 10, 11\}$ from Example 2.2.3. The representation as language over $\mathbb{B}$ is induced by the corresponding edge labeling.

is defined by $\varphi(\lambda) = q_1$ and $\varphi(1) = q_2$. Then, the path determined by any language L containing q_1 and q_2 leads to the leaf $\chi_L(q_1, q_2) = 11$, and the path determined by any language L' containing q_1, but not q_2, leads to the leaf $\chi_{L'}(q_1, q_2) = 10$.

Let M be a polynomial-time oracle Turing machine and w a word. Let $w_1, w_2, \ldots, w_m$ be an enumeration of all words that label some node in $\mathrm{QT}_M(w)$. Suppose we have a pool P for $(w_1, w_2, \ldots, w_m)$ and a language L. Since every path from the root to a leaf v is determined by some language K, we have $v = \chi_K(w_{i_1}, w_{i_2}, \ldots, w_{i_k})$, where $w_{i_1}, w_{i_2}, \ldots, w_{i_k}$ are the labels along this path. We call this path *consistent* with P if $v \in P[i_1, i_2, \ldots, i_k]$. A tree is called consistent with P if every path from the root to a leaf is consistent with P. Therefore, a maximal subtree of $\mathrm{QT}_M(w)$ that is consistent with P contains the path determined by L.

It is convenient to define the notion of consistency with a pool only for m-labeled trees. The idea is to interpret labels as columns in the pool. Then, it is easy to see that every Σ^*-labeled binary tree T can be converted into an m-labeled binary tree T' (where m is the number of words in T) such that T' has the same topology as T and that T' is consistent with a pool P if and only if T is consistent with P: We simply replace a word labeling some node by the index of the corresponding column of P.

Definition 4.3.8 (consistent). Let P be an m-pool. An m-labeling φ of a binary tree T is *consistent with* P if for every leaf $v \in T$ there exists a bitstring $b \in P$ such that for all $i \in [|v|]$ we have $b\big[\varphi\big(v[1, 2, \ldots, i-1]\big)\big] = v[i]$[1]. We say that b certifies consistency of φ with P with respect to v. An m-labeled binary tree is consistent with P if its labeling is consistent with P.

[1]For this section, let $b[1, \ldots, 0]$ defined by λ for any bitstring b.

Example 4.3.9. Let q_1 and q_2 be the two words in the query tree T from Figure 4.4. We consider the following 2-labeling φ of T: $\varphi(\lambda) = 1$ and $\varphi(1) = 2$. Then, φ is consistent with co-sel$_2$, but not with sel$_2$. However, the tree obtained by removing the node 10 and its incident edge from T is consistent with co-sel$_2$ and sel$_2$. △

Our aim is to bound the rank of m-labeled binary trees that are consistent with a pool P in terms of P. This can be done by using the notion of the width of a pool that gives this bound as we shall show.

Definition 4.3.10 (width). The *width of an m-pool P*, written $\mathrm{wd}(P)$, is the maximum d such that there is an m-labeling of B_d that is consistent with P.

Lemma 4.3.11. *If P is an m-pool with $\mathrm{wd}(P) \leq w$, then for every m-labeled binary tree T that is consistent with P we have $\mathrm{rk}(T) \leq w$.*

Proof. Let P be an m-pool with $\mathrm{wd}(P) \leq w$. Let T be a binary tree and φ be an m-labeling of T that is consistent with P. Assume that $\mathrm{rk}(T) > w$, i.e. that there exists an embedding ε of B_{w+1} into T. We show that the m-labeling ψ of B_{w+1} that is defined by $\psi(v) = \varphi(\varepsilon(v))$ for all interior nodes $v \in \mathrm{B}_{w+1}$ is consistent with P. Thus, $\mathrm{wd}(P) > w$ which contradicts $\mathrm{wd}(P) \leq w$.

Let v be an arbitrary leaf of B_{w+1} and $u := \varepsilon(v)$ its embedding. By Definition 4.3.2 we have $\varepsilon(\lambda)v[1] \sqsubseteq \varepsilon(v[1])v[2] \sqsubseteq \ldots \sqsubseteq \varepsilon(v[1,2,\ldots,|v|-1])v[|v|] \sqsubseteq u$. Hence, for every $i \in [|v|]$ there exists a number $j_i \in [|u|]$ such that $\varepsilon(v[1,2,\ldots,i-1])v[i] = u[1,2,\ldots,j_i]$. Since φ is consistent with P it follows that there exists a bitstring $b \in P$ such that for all $i \in [|v|]$ it holds that

$$\begin{aligned} b\big[\psi\big(v[1,2,\ldots,i-1]\big)\big] &= b\big[\varphi\big(\varepsilon(v[1,2,\ldots,i-1])\big)\big] \\ &= b\big[\varphi\big(u[1,2,\ldots,j_i-1]\big)\big] \\ &= u[j_i] \\ &= v[i] \end{aligned}$$

So, b certifies consistency of ψ with P with respect to v. It follows that ψ is consistent with P, which completes the proof. □

The next lemma is useful in the remaining part of this section and gives an upper bound on the width of pools in terms of their tuple length.

Lemma 4.3.12. *If P is an m-pool, then $\mathrm{wd}(P) \leq m$.*

Proof. Suppose for a contradiction that for some $d > m$ there exists an m-labeling φ of B_d that is consistent with P. Let v be an arbitrary leaf of B_d. Since the path from the root to v contains $d > m$ interior nodes there must be two numbers i and j with $1 \leq i < j \leq d$ such that $v[1, 2, \ldots, i-1]$ and $v[1, 2, \ldots, j-1]$ are assigned the same label by φ. Let v' be a leaf (not necessarily distinct to v) of B_d such that $v'[1, 2, \ldots, j-1] = v[1, 2, \ldots, j-1]$ and $v'[i] \neq v'[j]$. Then, $v'[1, 2, \ldots, i-1]$ and $v'[1, 2, \ldots, j-1]$ are still assigned the same label by φ, but no bitstring b in P satisfies $b\big[\varphi\big(v'[1, 2, \ldots, i-1]\big)\big] = v'[i]$ and $b\big[\varphi\big(v'[1, 2, \ldots, j-1]\big)\big] = v'[j]$. So, φ can not be consistent with P. □

4.3.3 Proof of the Theorem

We now prove Theorem 4.3.1. First, we show that if the width of pools in $\lceil\mathcal{F}\rceil_n$ for some family $\mathcal{F}$ in normal form is bounded by some constant for all $n \in \mathbb{N}^+$, then Turing and truth-table closures of $\mathrm{P}[\mathcal{F}]$ coincide. This result has been obtained by Beigel, Kummer, and Stephan in [BKS95a] (see their Lemma 6.7). Finally, we show that the width of all pools in $\lceil\mathrm{NONSEL}_m\rceil_n$ is bounded by $4^m - 2$ for all n.

Lemma 4.3.13. *Let $\mathcal{F}$ be an m-family in normal form such that for some constant c, for all $n \in \mathbb{N}^+$, and for all pools $P \in \lceil\mathcal{F}\rceil_n$ it holds that* $\mathrm{wd}(P) < c$. *Then, for any language L and any language $K \in \mathrm{P}[\mathcal{F}]$, if $L \leq_{\mathsf{T}}^{\mathsf{p}} K$ then $L \leq_{\mathsf{tt}}^{\mathsf{p}} K$.*

Proof. Let $L \leq_{\mathsf{T}}^{\mathsf{p}} K$ via a deterministic polynomial-time oracle Turing machine M. Furthermore, let K be in $\mathrm{P}[\mathcal{F}]$ via a function $f \in \mathrm{FP}$ that computes for every $n \in \mathbb{N}^+$ and for every n-tuple of words partial information of type $\lceil\mathcal{F}\rceil_n$. Such a function exists by Fact 3.4.5.

We show how to compute on an input w in polynomial time a subtree T of $\mathrm{QT}_M(w)$ that contains the path determined by K, that is, the path which contains all words queried by M on input w with oracle K. This allows us to decide whether w is in L or not by asking the oracle K about all of the polynomially many words in T in a non-adaptive way and by simulating M on input w with oracle K properly. This can be made by a deterministic polynomial-time oracle Turing machine, hence $L \leq_{\mathsf{tt}}^{\mathsf{p}} K$.

It remains to describe how to compute T in polynomial time. This is done inductively as follows. At the beginning, let T_0 be a binary tree which consists only of the root node labeled with the first word queried by M on input w. If there is no such word, then the root node is left unlabeled. Now suppose that for some $s \in \mathbb{N}^+$ we have already constructed T_{s-1}. If all leaves in the $(s-1)$-th level of T_{s-1} are unlabeled, then we are done and we define $T := T_{s-1}$.

Otherwise we extend in step s the tree T_{s-1} to a tree T_s as follows. Let $w_1, w_2, \ldots, w_k$ be an enumeration of all words that label some node of T_{s-1} and $P = f(w_1, w_2, \ldots, w_k)$ a pool for these words and K. We consider from left to right all leaves in the $(s-1)$-th level of T_{s-1} that are labeled with a word. Let v be the current leaf. For every $i \in [s]$ let w_{j_i} be the label of the i-th node on the path from the root to v. There can be two potential successors of v in T_s: $v_0 := v0$ and $v_1 := v1$. For each of these successors v_i we do the following. If $v_i \in P[j_1, j_2, \ldots, j_s]$ then we add v_i as a node to T_s and label it with the $(s+1)$-th word queried by M on input w with an oracle X satisfying $\chi_X(w_{j_1}, w_{j_2}, \ldots, w_{j_s}) = v_i$. If there is no such $(s+1)$-th query, then v_i is left unlabeled. This finishes the construction of T_s. Note that if v_i is not contained in $P[j_1, j_2, \ldots, j_s]$, then the path from the root to v_i can not be an initial segment of the path determined by K since P is a pool for $(w_1, w_2, \ldots, w_k)$ and K, and therefore $\chi_K(w_{j_1}, w_{j_2}, \ldots, w_{j_s}) \in P[j_1, j_2, \ldots, j_s]$.

How many nodes have to be processed in step s, i.e. how many leaves are there at most in the $(s-1)$-th level of T_{s-1}? If $s = 1$, there is only one such leaf. To answer this question for the case of $s > 1$, we consider the tree T'_{s-1} that consists only of paths in T_{s-1} from the root to leaves in level $s-1$. This tree is consistent with the pool P computed in step $s-1$ for the words $(w_1, w_2, \ldots, w_k)$ and K: If we consider a leaf v such that the words on the path from the root to v are $w_{j_1}, w_{j_2}, \ldots, w_{j_{s-1}}$, then $v \in P[j_1, j_2, \ldots, j_{s-1}]$ because v has been added to T_{s-1} in step $s-1$. This means that the k-labeling φ of T'_{s-1} which is defined by $\varphi(v) := i$ if and only if the label of node v is w_i is consistent with P. From Lemma 4.3.11 and the fact that $\mathrm{wd}(P) < c$ we derive that $\mathrm{rk}(T'_{s-1})$ is smaller than c, and from Fact 4.3.6 we know that there exists a polynomial p such that T'_{s-1} has at most $p(s-1)$ many nodes. So, the number of leaves in the $(s-1)$-th level of T'_{s-1} which equals the number of leaves in the $(s-1)$-th level of T_{s-1} is bounded above by $p(s-1)$.

The above observation implies that for each $s > 0$, the tree T_s can be computed in time polynomial in the depth of T_{s-1}. However, as mentioned in Section 2.2, since M is polynomially time-bounded the depth of $\mathrm{QT}_M(w)$ is bounded by $q(|w|)$ for some polynomial q, and because T_s is a subtree of $\mathrm{QT}_M(w)$ we have that the depth of T_s is also bounded by $q(|w|)$. So, for every s the tree T_s can be computed in time polynomial in $|w|$. Since s is at most as large as the depth of $\mathrm{QT}_M(w)$ we conclude that T can be computed in time polynomial in $|w|$, and T contains the path determined by K, because no initial segment of this path has been removed at any time. □

We now show that NONSEL_m satisfies the hypothesis of Lemma 4.3.13. For this,

we state two useful lemmas from [Kum92]. The first one deals with monochromatic embeddings of full binary trees into 2-colored full binary trees of double depth. Here, a *2-coloring of a binary tree* is a mapping from the nodes of this tree to some two element set whose elements are called colors.

Lemma 4.3.14. *For every 2-coloring c of B_{2d} there exists an embedding ε of B_d into B_{2d} such that for every two nodes v and u of B_d it holds that $c(\varepsilon(v)) = c(\varepsilon(u))$.*

The next lemma is actually a modification of Lemma 3 from [Kum92] from trees to pools. It plays an important role in the proof of Lemma 6.6 in [BKS95a], but is not proved there. Instead, they point out that it can be obtained by modification of Lemma 3 from [Kum92]. We prove it here.

Lemma 4.3.15. *For every m-pool P with $\mathrm{wd}(P) \geq 4^n - 2$, there exist n numbers $i_1, i_2, \ldots, i_n$ with $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ such that $P[i_1, i_2, \ldots, i_n]$ contains an ascending chain as a subset.*

Proof. Let P be an m-pool with $\mathrm{wd}(P) \geq 4^n - 2$. By Definition 4.3.10 there exists an m-labeling of $\mathrm{B}_{\mathrm{wd}(P)}$ which is consistent with P. Choose a subtree under the root of $\mathrm{B}_{\mathrm{wd}(P)}$. The labels in this subtree form an m-labeling of $\mathrm{B}_{\mathrm{wd}(P)-1}$ which is consistent with P, too. By iterating this process $\mathrm{wd}(P) - 4^n + 2$ times we obtain an m-labeling φ of B_{4^n-2} which is consistent with P.

The numbers are constructed in $2n-1$ steps. In step $s \in [2n-1]$ we choose a number $i_s \in [m]$ and a bitstring $b_s \in P$ such that both are distinct to the numbers, or bitstrings respectively, which have been chosen before. The number i_s is interpreted as the index of a column of P. At the end of this proof we define a subset of P which consists of n of the $2n-1$ bitstrings and one from the remaining bitstrings in P, and argue that the projection of this set onto n of the $2n-1$ columns forms an ascending chain.

To make the proof easier to follow we additionally define m-pools P_s (which contain bitstrings that can be chosen in the $(s+1)$-th step), sets I_s (which contain column indices that can be chosen in the $(s+1)$-th step), labelings φ_s, and numbers $c_s \in \mathbb{B}$, which satisfy the following conditions.

- For all $s \in [2n-1]$ it holds that $P_s \subseteq P_{s-1}$, and for all bitstrings $b \in P_s$ we have $b[i_s] = 1 - c_s$. We define $P_0 := P$.
- For all $s \in [2n-1]$ it holds that $I_s \subseteq I_{s-1}$, and for all $i \in I_s$ we have $b_s[i] = c_s$. We define $I_0 := [m]$.

- Let $h_0, h_1, \ldots, h_{2n-1}$ be a sequence of natural numbers with $h_0 := 4^n - 2$, $h_s := 2(h_{s+1}+1)$ for $1 \leq s \leq 2n-2$, and $h_{2n-1} := 0$. For all $s \in \{0, 1, \ldots, 2n-1\}$, φ_s is an I_s-labeling of B_{h_s} which is consistent with P_s. We define $\varphi_0 := \varphi$ which clearly satisfies this condition.

Suppose we finished the first $s-1$ steps ($1 \leq s \leq 2n-1$). In the s-th step we are given sets P_{s-1}, I_{s-1} and a labeling φ_{s-1} which satisfy the conditions above. Let b_s be a bitstring in P_{s-1} which certifies consistency of φ_{s-1} with P_{s-1} with respect to an arbitrary leaf of $\mathrm{B}_{h_{s-1}}$. This bitstring induces a 2-coloring c of $\mathrm{B}_{h_{s-1}}$ with colors from $\mathbb{B}$ as follows: Every leaf v is colored arbitrarily, for instance, $c(v) := 0$, and every interior node v is colored by $c(v) := b_s[\varphi_{s-1}(v)]$. By Lemma 4.3.14, there exists an embedding ε of $\mathrm{B}_{\frac{1}{2}h_{s-1}} = \mathrm{B}_{h_s+1}$ into $\mathrm{B}_{h_{s-1}}$ such that for every two nodes v_1 and v_2 of B_{h_s+1} we have $c(\varepsilon(v_1)) = c(\varepsilon(v_2))$. Let i_s be the label of the embedded root and c_s its color, i.e.

$$i_s := \varphi_{s-1}(\varepsilon(\lambda)) \quad \text{and} \quad c_s := c(\varepsilon(\lambda)) = b_s[\varphi_{s-1}(\varepsilon(\lambda))] = b_s[i_s]. \tag{4.1}$$

Note that $\varepsilon(\lambda)$ is an interior node of $\mathrm{B}_{h_{s-1}}$, because otherwise ε would not be an embedding by Definition 4.3.2 since $\varepsilon(\lambda)0 \sqsubseteq \varepsilon(0)$ and $\varepsilon(\lambda)1 \sqsubseteq \varepsilon(1)$ can not be satisfied. So, c_s is defined. Note also that for all interior nodes $v \in \mathrm{B}_{h_s+1}$ it holds that

$$b_s[\varphi_{s-1}(\varepsilon(v))] = c(\varepsilon(v)) = c(\varepsilon(\lambda)) = c_s. \tag{4.2}$$

Now we define the sets P_s, I_s, and the labeling φ_s. The set P_s is defined by $P_s := \{b \in P_{s-1} \mid b[i_s] = 1 - c_s\}$. From the definition it is immediate that $P_s \subseteq P_{s-1}$ and that for all $b \in P_s$ we have $b[i_s] = 1 - c_s$. Before we define the set P_s we define the labeling φ_s of B_{h_s} and show that it is consistent with P_s. The labeling φ_s is defined for all interior nodes v of B_{h_s} by

$$\varphi_s(v) := \varphi_{s-1}(\varepsilon((1-c_s)v)).$$

Clearly, φ_s is a labeling of B_{h_s}. It remains to show that φ_s is consistent with P_s. That is, we show that for every leaf v of B_{h_s} there exists a bitstring $b \in P_s$ such that $b[\varphi_s(v[1, 2, \ldots, j-1])] = v[j]$ for all possible indices $j \in [h_s]$. Fix such a leaf v and an index j. Let v' be a leaf of $\mathrm{B}_{h_{s-1}}$ such that the path from the root to this leaf contains the node $\varepsilon((1-c_s)v)$. Because ε is an embedding it follows that $\varepsilon(\lambda)(1-c_s) \sqsubseteq \varepsilon(1-c_s)v[1] \sqsubseteq \varepsilon((1-c_s)v[1])v[2] \sqsubseteq \ldots \sqsubseteq \varepsilon((1-c_s)v) \sqsubseteq v'$. Therefore we find two indices j_1' and j_2' with

$$\varepsilon(\lambda)(1-c_s) = v'[1, 2, \ldots, j_1'] \tag{4.3}$$

and

$$\varepsilon((1-c_s)v[1,2,\ldots,j-1])v[j] = v'[1,2,\ldots,j'_2]. \tag{4.4}$$

Since φ_{s-1} is consistent with P_{s-1} there is a bitstring $b \in P_{s-1}$ such that for all $j' \in [h_{s-1}]$ it holds that $b[\varphi_{s-1}(v'[1,2,\ldots,j'-1])] = v'[j']$. Using equation (4.1) and (4.3) we obtain $b[i_s] = b[\varphi_{s-1}(\varepsilon(\lambda))] = b[\varphi_{s-1}(v'[1,2,\ldots,j'_1-1])] = v'[j'_1] = 1-c_s$ which implies $b \in P_s$. From the definition of φ_s and equation (4.4) it follows that $b[\varphi_s(v[1,2,\ldots,j-1])] = b[\varphi_{s-1}(\varepsilon((1-c_s)v[1,2,\ldots,j-1]))] = b[\varphi_{s-1}(v'[1,2,\ldots,j'_2-1])] = v'[j'_2] = v[j]$. Hence, φ_s is consistent with P_s.

Finally, the set I_s is the set of all labels from φ_s, or to be more precisely, $I_s := \{\varphi_s(u) \mid u \text{ is an interior node of } \mathrm{B}_{h_s}\}$. Therefore, φ_s is an I_s-labeling of B_{h_s} which is consistent with P_s. Since φ_s labels a subtree of the one labeled by φ_{s-1} it follows that $I_s \subseteq I_{s-1}$. Furthermore, if $i \in I_s$ then there exists an interior node v of B_{h_s} such that $b_s[i] = b_s[\varphi_s(v)] = b_s[\varphi_{s-1}(\varepsilon((1-c_s)v))] = c_s$ by equation (4.2). This finishes the s-th step.

Now let $s \in [2n-1]$ be arbitrary. We consider the bitstring b_s. By construction we have $b_s \in P_{s-1} \subseteq \ldots \subseteq P_0$. Since $b[i_j] = 1-c_j$ for all $b \in P_j$ $(1 \leq j \leq 2n-1)$ we obtain $b_s[i_1, i_2, \ldots, i_{s-1}] = (1-c_1)(1-c_2)\ldots(1-c_{s-1})$. On the other hand, each i_j $(s+1 \leq j \leq 2n-1)$ has been defined such that $i_j \in I_{j-1} \subseteq \ldots \subseteq I_s$. Using the fact that $b_s[i] = c_s$ for all $i \in I_s$ we obtain $b_s[i_{s+1}, i_{s+2}, \ldots, i_{2n-1}] = c_s^{2n-s-1}$. Putting this all together, and using equation (4.1), we have

$$b_s[i_1,\ldots,i_{s-1},i_s,i_{s+1},\ldots,i_{2n-1}] = (1-c_1)\ldots(1-c_{s-1})\underbrace{c_s\ldots c_s}_{\substack{2n-s\\ \text{times}}}.$$

By the pigeonhole principle, there exists a number $c \in \mathbb{B}$ and n distinct numbers $j_1, j_2, \ldots, j_n \in [2n-1]$ such that $c_{j_1} = c_{j_2} = \ldots = c_{j_n} = c$. Hence, for all $k \in [n]$ we have

$$b_{j_k}[i_{j_1}, i_{j_2}, \ldots, i_{j_n}] = (1-c_{j_1})\ldots(1-c_{j_{k-1}})\underbrace{c_{j_k}\ldots c_{j_k}}_{\substack{n-k+1\\ \text{times}}} = \underbrace{(1-c)\ldots(1-c)}_{k-1 \text{ times}}\underbrace{c\ldots c}_{\substack{n-k+1\\ \text{times}}}$$

Choose $b_{j_{n+1}}$ out of P_{j_n}. By construction of P_{j_n} it holds that

$$b_{j_{n+1}}[i_{j_1}, i_{j_2}, \ldots, i_{j_n}] = (1-c_{j_1})\ldots(1-c_{j_n}) = \underbrace{(1-c)\ldots(1-c)}_{n \text{ times}}.$$

The set $P' := \{b_{j_1}, b_{j_2}, \ldots, b_{j_{n+1}}\}$ is a subset of P, and by the preceding observations,

$P'[i_{j_1}, i_{j_2}, \ldots, i_{j_n}]$ is an ascending chain. Every permutation of the positions of an ascending chain yields an ascending chain again. Thus, with π being a permutation of $(j_1, j_2, \ldots, j_n)$ with $i_{\pi(j_1)} < i_{\pi(j_2)} < \ldots < i_{\pi(j_n)}$, $P[i_{\pi(j_1)}, i_{\pi(j_2)}, \ldots, i_{\pi(j_n)}]$ contains an ascending chain. □

Let P be a pool in $\lceil \text{NONSEL}_m \rceil_n$ for some $n \in \mathbb{N}^+$. In case $n < m$, Lemma 4.3.12 tells us that $\text{wd}(P) < m \leq 4^m - 2$. If $n \geq m$, then for every sequence of m numbers $i_1, i_2, \ldots, i_m$ with $1 \leq i_1 < i_2 < \ldots < i_m \leq n$ it holds that $P[i_1, i_2, \ldots, i_m]$ is in NONSEL_m (Definition 3.4.2) and therefore contains no ascending chain. The bound $\text{wd}(P) < 4^m - 2$ then follows from Lemma 4.3.15. So, the width of all pools in $\lceil \text{NONSEL}_m \rceil_n$ is bounded by $4^m - 2$ for every $n \in \mathbb{N}^+$. By Lemma 4.3.13, every language that is Turing reducible to some language $K \in \text{P}[\text{NONSEL}_m]$ is truth-table reducible to K. This finishes the proof of Theorem 4.3.1.

It is known [Tod91] that there exists a language L that is Turing reducible to some language in $\text{P}[\text{SEL}_m]$, but not truth-table reducible to any language in $\text{P}[\text{SEL}_m]$. Since NONSEL_m is the largest m-family in normal form that does not include SEL_m, we obtain the following corollary from this fact and Theorem 4.3.1.

Corollary 4.3.16. *Let $\mathcal{F}$ be an m-family in normal form. Then, the following statements are equivalent.*

1. *If a language is Turing reducible to a language $K \in \text{P}[\mathcal{F}]$, then it is truth-table reducible to K.*
2. *$\mathcal{F} \subseteq \text{NONSEL}_m$.*

Chapter 5

Combining Self-Reducibility and Partial Information

In Section 2.4, we introduced self-reducibility and learned that self-reducible languages are contained in PSPACE. Moreover, there are self-reducible languages that are $\leq_{\mathsf{m}}^{\mathsf{p}}$-complete for NP or PSPACE. Therefore, it is unlikely that all self-reducible languages belong, for instance, to P. But does this apply to self-reducible languages with partial information algorithms?

In the first two sections of this chapter, we will show for specific families $\mathcal{F}$ that self-reducible languages in $\mathrm{P}[\mathcal{F}]$ belong to P. This is achieved in two or more steps. In the first step – which is handled in Section 5.1 – we show that Turing self-reducible languages in $\mathrm{P}[\mathcal{F}]$ are already truth-table self-reducible. In the remaining steps – which are handled in the corresponding subsections of Section 5.2 – we then show that this implies membership in P.

In Section 5.3, we finally discuss whether there are or can be self-reducible partial information languages that are not contained in P.

5.1 Converting Turing into Truth-Table Self-Reductions

In this section we try to answer the question, for which families $\mathcal{F}$ it holds that self-reducible languages in $\mathrm{P}[\mathcal{F}]$ are already truth-table self-reducible. This allows us to concentrate in the remaining sections on truth-table self-reductions which are much more convenient than Turing self-reductions.

From Section 4.3 we know already that Turing reductions to languages in $\mathrm{P}[\mathrm{NONSEL}_m]$ can be replaced by truth-table reductions. A closer look at the proof of this fact shows

that only words are queried in the truth-table reduction that are also queried in the Turing reduction. Thus, if the oracle Turing machine which carries out the Turing reduction queries on every input w only words that are smaller than w with respect to some polynomially related partial ordering $\preceq$, then the oracle Turing machine which is responsible for the truth-table reduction queries on input w only words that are smaller than w with respect to $\preceq$, too. This gives the following corollary.

Corollary 5.1.1. *Every self-reducible language in* $\mathrm{P}[\mathrm{NONSEL}_m]$ *is tt-self-reducible.*

From this corollary it follows, for instance, that every self-reducible easily countable language, and every self-reducible p-cheatable language is tt-self-reducible. But there are many partial information classes which are not subsets of $\mathrm{P}[\mathrm{NONSEL}_m]$ for any m. One of them is the class of p-selective languages. However, it was shown by Buhrman, van Helden, and Torenvliet that if a self-reducible language is contained in this class, then it is truth-table self-reducible [BvHT93].

Their result can be generalized to a larger partial information class. For this, recall that for every p-selective language L we can compute on input of $m \geq 2$ words a pool from SEL_m for these words and L. In particular, this pool contains at most $m+1$ bitstrings. The following lemma states that even if we only require that the pool contains at most $2m-1$ bitstrings we can convert self-reducible languages to truth-table self-reducible languages.

Lemma 5.1.2. *Every self-reducible language in* $\mathrm{P}[(2m-1)\text{–}\mathrm{SIZE}_m]$ *is tt-self-reducible.*

Proof. Let L be self-reducible via an oracle Turing machine M and a polynomially related partial ordering $\preceq$. We assume further that L is in $\mathrm{P}[(2m-1)\text{–}\mathrm{SIZE}_m]$ and that f is the corresponding partial information function. We show how a non-adaptive polynomial-time oracle Turing machine M' decides L by asking the oracle L only for words that are smaller than the input with respect to $\preceq$.

On an input w, it processes $\mathrm{QT}_M(w)$ with a breadth-first extend-and-prune algorithm. Pruning means removing a subtree and replacing it by a leaf node labeled "$\chi_A(x) = 1$" or "$\chi_A(x) = 0$". Extending means adding successors to nodes labeled with words. That is, M' adds two successors to a node labeled w', and by simulating M it finds out the corresponding labels of these successors in $\mathrm{QT}_M(w)$. If a successor v is a leaf in $\mathrm{QT}_M(w)$, then M' labels it with "$\chi_L(w) = c$", where $c \in \mathbb{B}$ is chosen to be 1 if and only if M accepts w with any oracle that determines the path to v. This way, M' always keeps a tree T which is up to the labels at the leaves a subtree of $\mathrm{QT}_M(w)$. Additionally, it ensures the following invariants for T.

1. On every path from the root to a leaf in T there are at most $m-2$ nodes having two interior nodes as successors.

2. If a leaf v is labeled "$\chi_L(w) = c$" for some $c \in \mathbb{B}$, then M accepts w for all oracles K which have the following property. Let w_i be the label of the i-th node on the path from the root to v, and a_i the label of the i-th edge. Then, $\{w_i \mid a_i = 1\} \subseteq K$ and $\{w_i \mid a_i = 0\} \subseteq \overline{K}$.

Initially, T is the full query tree up to depth $m-1$. Suppose T has been extended such that there is a path $v_1 v_2 \dots v_k$ on which $m-1$ nodes have two interior nodes as successors. Let $v_{i_1}, v_{i_2}, \dots, v_{i_{m-1}}$ be these nodes in ascending order. For every $j \in [m-1]$, let w_j be the label of v_{i_j}, and for every $j \in [m-2]$, let a_j be the label of the edge from v_{i_j} to v_{i_j+1}. Then, M' computes a pool $P := f(w, w_1, w_2, \dots, w_{m-1})$ and splits it into two pools defined by $P_c := \{b \in P \mid b[1] = c\}$ for $c \in \mathbb{B}$. Observe that $|P_0| + |P_1| = |P| \leq 2m-1$. Hence, there must be a $c \in \mathbb{B}$ with $|P_c| \leq m-1$. Now M' can find an index $j \in [m-1]$ such that $ca_1a_2 \dots a_{j-1}0$ or $ca_1a_2 \dots a_{j-1}1$ is not a prefix of a bitstring in P_c. We consider the first case. The other one is analogous.

If $ca_1a_2 \dots a_{j-1}0$ is not a prefix of a bitstring in P_c, then w_j must be in L if $\chi_L(w) = c$. In other words we have $\chi_L(w) = 1-c$ if w_j is not in L. Let v be the successor of v_{i_j} such that the edge from v_{i_j} to v is labeled 0. Then, M' prunes T by replacing the subtree with v as root by a new leaf node labeled "$\chi_L(w) = 1-c$". This leaf must not be considered further. Observe that when the path determined by L (i.e. the path from the root to a leaf with the following property: if the i-th interior node is labeled w', then the i-th edge is labeled $\chi_L(w')$) ends in this leaf, then we have $\chi_L(w) = 1-c$.

When $\mathrm{QT}_M(w)$ has been processed up to some depth, only a constant number of nodes of this depth with two interior nodes as successors remain in T. Therefore the whole processing can be completed in polynomial time and in the end only polynomially many words are left in T. These are the words that M' now queries in parallel. Finally, if the path determined by L leads to a node labeled "$\chi_L(w) = 1$", then M' accept w, else it reject it.

We conclude that M' is able to determine in polynomial time whether $w \in L$ or not by using only non-adaptive queries to L. The words queried are words also queried by M on input w. So, L is tt-self-reducible via M' and $\preceq$. □

We do not know whether the result of Lemma 5.1.2 is optimal. For all $m > 2$, there are still many m-families that properly include $(2m-1)$–SIZE_m, and whose corresponding partial information classes are therefore not covered by this lemma. For example, the

family SMC_m includes $(2m-1)$–SIZE_m and contains the pool $\mathbb{B}^m \setminus \{1^m\}$. This pool consists of $2^m - 1$ bitstrings, hence for every $m > 2$ the family SMC_m is larger than $(2m-1)$–SIZE_m. Even the family $\mathrm{SMC}_m \cap \mathrm{COSMC}_m$ is larger than $(2m-1)$–SIZE_m for all $m > 2$ since it contains the pool $\mathbb{B}^m \setminus \{1^m, 0^m\}$ which consists of $2^m - 2$ bitstrings.

5.2 Self-Reducible Partial Information Languages that are in P

In this section, we show that self-reducible languages in specific partial information classes belong to P. Since every language in P is trivially self-reducible and contained in every such partial information class, languages in P are characterized as self-reducible languages in those classes. The results presented here are motivated by the following results which appeared in [BvHT93], [ABG00, GJY93], and [BKS95a], respectively.

Fact 5.2.1.

- *A language is self-reducible and p-selective if and only if it is in* P.
- *A language is self-reducible and p-cheatable if and only if it is in* P.
- *A language is self-reducible and easily approximable*[1] *if and only if it is in* P.

The first two results can be strengthened to the following results by using Corollary 4.2.2 and Fact 4.1.3.

Corollary 5.2.2.

- *A language L is self-reducible and $\leq^{\mathsf{p}}_{\mathsf{pos}}$-reducible to a p-selective language if and only if L is in* P.
- *A language L is self-reducible and Turing reducible to a p-cheatable language if and only if L is in* P.

In the next two subsections, we will extend the first result presented in Fact 5.2.1 to the classes $\mathrm{P}[\mathrm{SMC}_2 \cap \mathrm{COSMC}_2]$ and $\mathrm{P}[\mathrm{NONSEL}_2]$. In the third and last one, we will discuss results where disjunctive and conjunctive self-reducibilities are involved.

[1] Easily approximable languages have partial information algorithms that compute for m pairwise distinct input words a pool from k–FREQ_m for some m and $k < \frac{m}{2}$. Every pool in k–FREQ_m contains only bitstrings with hamming distance at most k from a fixed bitstring.

5.2.1 Self-Reducible Languages in P[$\mathrm{SMC}_2 \cap \mathrm{COSMC}_2$] are in P

We show that every self-reducible language in P[$\mathrm{SMC}_2 \cap \mathrm{COSMC}_2$] belongs to P. Since $\mathrm{SMC}_2 \cap \mathrm{COSMC}_2$ forms a subset of 3–SIZE_2, it suffices to show by Lemma 5.1.2 that all truth-table self-reducible languages in P[$\mathrm{SMC}_2 \cap \mathrm{COSMC}_2$] belong to P. Before we prove the corresponding lemma, we state Lemma 3.24 from [Nic01] as a known fact which will be very helpful.

Fact 5.2.3. *Let L be a language in* P[$\mathrm{SMC}_2 \cap \mathrm{COSMC}_2$]. *Then we can compute for every tuple $(w_1, w_2, \ldots, w_k)$ of words in polynomial time a partition of $X = \{w_1, w_2, \ldots, w_k\}$ into disjoint sets of one of the following two types:*

1. *$X = \mathrm{IN} \cup \mathrm{OUT} \cup S_1 \cup \ldots \cup S_l$ with $l \leq k$ such that*
   - *$\mathrm{IN} \subseteq L$ and $\mathrm{OUT} \subseteq \overline{L}$,*
   - *for $1 \leq i \leq l$ we have $S_i \neq \emptyset$, and $S_i \subseteq L$ or $S_i \subseteq \overline{L}$, and*
   - *for $1 \leq i < j \leq l$, $x \in S_i$ and $y \in S_j$ we have $x \in L \implies y \in L$.*
2. *$X = \mathrm{IN} \cup \mathrm{OUT} \cup X_1 \cup X_2$ such that*
   - *$\mathrm{IN} \subseteq L$ and $\mathrm{OUT} \subseteq \overline{L}$,*
   - *for $i \in \{1, 2\}$ we have $X_i \neq \emptyset$, and $X_i \subseteq L$ or $X_i \subseteq \overline{L}$, and*
   - *for $x \in X_1$ and $y \in X_2$ we have $x \in L \iff y \notin L$.*

Lemma 5.2.4. *Every tt-self-reducible language in* P[$\mathrm{SMC}_2 \cap \mathrm{COSMC}_2$] *belongs to* P.

Proof. Suppose that L is tt-self-reducible via M and $\preceq$. Let L be in P[$\mathrm{SMC}_2 \cap \mathrm{COSMC}_2$]. We show how an oracle Turing machine M' decides L with at most one query to L that is smaller than the input word with respect to $\preceq$. This will imply that L is 1-tt-self-reducible and therefore in P by Theorem 2.4.9.

On input w it computes by simulation of M on input w the set Q of words queried by M on input w. If this set is empty, then M' accepts w if and only if M accepts it. Otherwise there is at least one word in Q and M' computes, according to Fact 5.2.3, in polynomial time a partition of $X = Q \cup \{w\}$ of one of the two types.

Case 1. Assume that a partition of the first type has been computed. That is, we obtain pairwise disjoint sets $\mathrm{IN}, \mathrm{OUT}, S_1, \ldots, S_l$ such that X is their union and it holds that

- $\mathrm{IN} \subseteq L$ and $\mathrm{OUT} \subseteq \overline{L}$,

- for $1 \leq i \leq l$ we have $S_i \neq \emptyset$, and $S_i \subseteq L$ or $S_i \subseteq \overline{L}$, and
- for $1 \leq i < j \leq l$, $x \in S_i$ and $y \in S_j$ we have $x \in L \implies y \in L$.

If $w \in \text{IN}$, then it holds that $w \in L$ and M' accepts w. The case for $w \in \text{OUT}$ is analogous except that w is rejected. In any other case let i be the index of the set S_i that contains w. If w is not the only word in S_i then M' chooses a word $x \in S_i$ distinct to w which has, by definition of S_i, the property that $w \in L$ if and only if $x \in L$. So, M' only has to query x in order to determine whether $w \in L$ or not. Now suppose that w is the only word in S_i. The following three cases arise.

1. If $i = 1$, then $w \in L$ implies $x \in L$ for every $x \in S_2$. The only case in which w can be in L is the case that $S_2 \subseteq L$ which implies $S_j \subseteq L$ for all $j \in \{2, \ldots, l\}$. Now M' simulates M on input w for this possibility. If M rejects w, then we know that $w \notin L$. Otherwise we have $w \in L$ if and only if $S_2 \subseteq L$ and we only have to query a word $x \in S_2$ in order to determine membership of w in L.

2. If $i = l$, then $x \in L$ implies $w \in L$ for every $x \in S_{l-1}$. This is very similar to the first case. We only have to query a word $x \in S_{l-1}$ in order to determine membership of w in L.

3. If $1 < i < l$, then $x \in L \implies w \in L \implies y \in L$ for every $x \in S_{i-1}$ and $y \in S_{i+1}$. Three possibilities for inclusion of S_{i-1} and S_{i+1} in L and $\overline{L}$, respectively, remain: (1) $S_{i-1} \subseteq \overline{L}$ and $S_{i+1} \subseteq \overline{L}$, (2) $S_{i-1} \subseteq L$ and $S_{i+1} \subseteq L$, and (3) $S_{i-1} \subseteq \overline{L}$ and $S_{i+1} \subseteq L$. The first two possibilities imply $w \in \overline{L}$ and $w \in L$, respectively. Now M' simulates M on input w for the third possibility. If M accepts w, then $w \in L$ if and only if $S_{i+1} \subseteq L$, and otherwise $w \in L$ if and only if $S_{i-1} \subseteq L$. In each case, membership of w in L can be determined by querying a word $x \in S_{i+1}$ (or S_{i-1}, respectively).

Case 2. Assume that a partition of the second type has been computed. That is, we obtain pairwise disjoint sets $\text{IN}, \text{OUT}, X_1$ and X_2 such that X is their union and it holds that

- $\text{IN} \subseteq L$ and $\text{OUT} \subseteq \overline{L}$,
- for $i \in \{1, 2\}$ we have $X_i \neq \emptyset$, and $X_i \subseteq L$ or $X_i \subseteq \overline{L}$, and
- for $x \in X_1$ and $y \in X_2$ we have $x \in L \iff y \notin L$.

If $w \in \mathrm{IN}$, then we have $w \in L$ and M' accepts w. The case for $w \in \mathrm{OUT}$ is analogous except that w is rejected. Otherwise we may assume without loss of generality that $w \in X_1$ (if $w \in X_2$ we swap X_1 and X_2). Since $X_2 \neq \emptyset$ we can choose a word $x \in X_2$ which has the property that $w \in L$ if and only if $x \notin L$. Hence, M' can determine whether $w \in L$ or not by querying x.

Clearly, M' decides L with oracle L in polynomial time. Note that M' makes at most one query. This is one of the words queried by M on input w. Hence, this query is smaller than w with respect to the polynomially related partial ordering $\preceq$. □

From this lemma we can derive the following two corollaries. The second one is a stronger version of the first one, which additionally uses Fact 4.2.3.

Corollary 5.2.5.

- *A language is self-reducible and in* $\mathrm{P}[\mathrm{SMC}_2 \cap \mathrm{COSMC}_2]$ *if and only if it is in* P.
- *A language L is self-reducible and $\leq^{\mathsf{p}}_{\mathsf{ptt}}$-reducible to a language that is contained in* $\mathrm{P}[\mathrm{SMC}_2 \cap \mathrm{COSMC}_2]$ *if and only if L belongs to* P.

5.2.2 Self-Reducible Languages in P[NONSEL$_2$] are in P

We now show that self-reducible languages in $\mathrm{P}[\mathcal{F}]$ belong to P for all 2-families $\mathcal{F}$ in normal form with $\mathrm{SEL}_2 \not\subseteq \mathcal{F}$. In particular, we show that self-reducible languages in $\mathrm{P}[\mathrm{NONSEL}_2]$ are in P[2]. This partially answers an open question from Beigel, Kummer, and Stephan in [BKS95a]. They ask whether all self-reducible easily m-countable languages – which are contained in NONSEL_m – belong to P.

By Corollary 5.1.1 and also by Lemma 5.1.2, self-reducible languages in $\mathrm{P}[\mathrm{NONSEL}_2]$ are already truth-table self-reducible. We show that every truth-table self-reducible language in $\mathrm{P}[\mathrm{NONSEL}_2]$ belongs to P. This is achieved in two steps. In the first step we show that tt-self-reducible languages in $\mathrm{P}[\mathrm{NONSEL}_2]$ are nor-nand self-reducible, and in the second step we show that this implies already membership in P.

Before we proceed, we define nor-nand self-reducibility. This is a variation of truth-table self-reducibility, where the corresponding boolean functions α_w (see Section 2.3.2) are for all words w either special negated disjunctions or special negated conjunctions.

[2] Although NONSEL_2 equals 2–CARD_2 as mentioned in Section 3.2.4, we prefer to write NONSEL_2 to emphasize that the result obtained here holds for the largest 2-family $\mathcal{F}$ in normal form with $\mathrm{SEL}_2 \not\subseteq \mathcal{F}$.

Definition 5.2.6 (nor-nand self-reducibility). A language L is *nor-nand self-reducible* if it is tt-self-reducible via an oracle Turing machine M with the following property. If M makes on an input w at least one query then either

- at most one query is in L, where $w \notin L$ if and only if one query is in L, or
- at most one query is in $\overline{L}$, where $w \in L$ if and only if one query is in $\overline{L}$.

It is immediate that every nor-nand self-reducible language is truth-table self-reducible, but not positive truth-table self-reducible. We show that non-selective partial information suffices to reduce the complexity of truth-table self-reducible to nor-nand self-reducible languages.

Lemma 5.2.7. *All tt-self-reducible languages in* $\mathrm{P}[\mathrm{NONSEL}_2]$ *are nor-nand self-reducible.*

Proof. Let L be tt-self-reducible via some self-reduction machine M, and let L be in $\mathrm{P}[\mathrm{NONSEL}_2]$ via some partial information function f. Without loss of generality, we assume that f outputs only maximals pools from NONSEL_2. These maximal pools are equ_2, bottom_2, and top_2. We describe how a polynomial-time oracle Turing machine M' decides for every word w whether it belongs to L or not by using L as oracle. This machine will satisfy the conditions given in Definition 5.2.6 and shows therefore that L is nor-nand self-reducible.

Membership of a word w in L is determined by M' in three steps. In the first step, it finds a node v in $\mathrm{ST}_M(w)$ such that the label of v is in L if and only if w is in L, and for all words q labeling some successor of v it holds that $f(w,q) \in \{\mathrm{bottom}_2, \mathrm{top}_2\}$. In the second step, it determines all words labeling some successor of v which have the property that w is in L only if either all or at least one of these words are in $\overline{L}$. These words are then used in the third step to decide whether w is in L or not.

Step 1. The purpose of this step is to find a node v in $\mathrm{ST}_M(w)$ such that the label of this node is in L if and only if w is in L and for every word q labeling some successor of v it holds that $f(w,q) \in \{\mathrm{bottom}_2, \mathrm{top}_2\}$. Let $w' := w$. As long as v has not been found, M' does the following. It computes by simulation of M the list of queries posed by M on input w' (if M makes no query, then M' accepts w if and only if M accepts w'). Let $q_1, q_2, \ldots, q_k$ be this list. If there exists an $i \in [k]$ such that $f(w,q_i) = \mathrm{equ}_2$, then it holds that $w \in L$ if and only if $q_i \in L$, and we repeat the whole procedure with $w' := q_i$. Otherwise, w' is the label of the desired node v and the words $q_1, q_2, \ldots, q_k$ have the property that $f(w,q_i) \in \{\mathrm{bottom}_2, \mathrm{top}_2\}$ for every $i \in [k]$. Since the self-reduction tree

of M is by Lemma 2.4.8 bounded by a polynomial p in $|w|$, M' has to repeat this process at most $p(|w|)$ times.

Step 2. Suppose that $q_1, q_2, \ldots, q_k$ are the labels of the successors of v computed in Step 1. Then, M' computes with the following procedure three sets IN, OUT and X such that

- $\text{IN} \cup \text{OUT} \cup X = \{q_1, q_2, \ldots, q_k\}$,
- $\text{IN} \subseteq L$,
- $\text{OUT} \subseteq \overline{L}$, and
- either $f(w, x) = \text{bottom}_2$ or $f(w, x) = \text{top}_2$ for all $x \in X$.

Initially, it sets $\text{IN} = \text{OUT} = \emptyset$ and $X = \{q_1, q_2, \ldots, q_k\}$. As long as there are two words $x_1, x_2 \in X$ with $f(w, x_1) = \text{bottom}_2$ and $f(w, x_2) = \text{top}_2$, it computes $P = f(x_1, x_2)$. If $P = \text{bottom}_2$, then $x_1 \notin L$ (if x_1 were in L, then w would not be in L, because $f(w, x_1) = \text{bottom}_2$, and therefore x_2 would not be in L which contradicts $f(w, x_2) = \text{top}_2$). So, it moves x_1 from X into OUT. If $P = \text{top}_2$, then $x_2 \in L$ (if x_2 were not in L, then w would be in L, because $f(w, x_2) = \text{top}_2$, and therefore x_1 would be in L which contradicts $f(w, x_1) = \text{bottom}_2$). So, it moves x_2 from X into IN. If $P = \text{equ}_2$, then $w \in L$ if and only if $x_1 \notin L$. In this case, M' stops and $\chi_L(w)$ is determined by querying x_1.

Step 3. Suppose that w' is the label of the node v computed in Step 1, and that the sets IN, OUT, and X form the partition of $\{q_1, q_2, \ldots, q_k\}$ computed in Step 2. We only consider the case that $f(w, x) = \text{bottom}_2$ for all $x \in X$. The case for top_2 is symmetric. If there are two words $x_1, x_2 \in X$ with $f(x_1, x_2) = \text{top}_2$, then w can not be in L (otherwise $x_1, x_2 \notin L$ which contradicts $f(x_1, x_2) = \text{top}_2$). In this case, M' stops and rejects w. Otherwise it computes a partition of X into disjoint sets $X_1, X_2, \ldots, X_k$ such that for all $i \in [k]$ we have $X_i \subseteq L$ or $X_i \subseteq \overline{L}$, and for at most one i we have $X_i \subseteq L$. It makes this inductively: In the $(i+1)$-th step, it picks a word x from the set $X' = X \setminus (X_1 \cup X_2 \cup \ldots \cup X_i)$ and defines $X_{i+1} := \{x\} \cup \{y \in X' \mid f(x, y) = \text{equ}_2\}$. If $X' \setminus X_{i+1} = \emptyset$, then the sets $X_1, X_2, \ldots, X_k$ with $k = i+1$ form the desired partition. Let $x_1, x_2, \ldots, x_k$ be representants from those sets, i.e. $x_i \in X_i$ for every $i \in [k]$. Then, at most one word in $\{w, x_1, x_2, \ldots, x_k\}$ is in L by construction of the X_i and the fact that $f(w, x_i) = \text{bottom}_2$ for all i. The only possibility where w can be in L is when $\{x_1, x_2, \ldots, x_k\}$ is a subset of $\overline{L}$. So, if M rejects w' for the possibility that $\{x_1, x_2, \ldots, x_k\} \subseteq \overline{L}$, then M' rejects w

since it can not be in L. Otherwise, $w \in L$ if and only if all words in $\{x_1, x_2, \ldots, x_k\}$ do not belong to L. In this case, M' queries all words in $\{x_1, x_2, \ldots, x_k\}$ and accepts w if and only if they are not in L.

By construction, the conditions in Definition 5.2.6 are satisfied. If a single word q is queried, then only if $w \in L \iff q \notin L$. More than one word is only queried in Step 3, where the condition is clearly satisfied. □

Having shown that every tt-self-reducible language in P[NONSEL_2] is nor-nand self-reducible, we show that this already implies membership in P.

Lemma 5.2.8. *Every nor-nand self-reducible language in* P[NONSEL_2] *is in* P.

Proof. Let L be nor-nand self-reducible via M and a polynomially related partial ordering $\preceq$. Furthermore, let L be in P[NONSEL_2] via a partial information function f which outputs without loss of generality on every input only maximal pools from NONSEL_2. These maximal pools are equ_2, bottom_2, and top_2. We show how an oracle Turing machine M' decides whether a given input is in L or not by asking the oracle L at most one time for a word that is smaller than the input word with respect to $\preceq$. Thus, L is 1-tt self-reducible via M' and $\preceq$, which implies that L is in P by Theorem 2.4.9.

On input w, it computes $\text{ST}_M(w)$ up to level three and uses f to determine membership of w in L directly or to find a word to ask in order to determine it. More precisely, it simulates M on input w to obtain the set Q of all words queried by M on input w (if this set is empty, it accepts w if and only if M accepts it). To find out which of the cases in Definition 5.2.6 holds, it simulates M once more on input w for the case that all words in Q are in $\overline{L}$. If it accepts w, then the first condition holds and otherwise the second one. We assume that the first condition holds, i.e. that

$$\text{at most one query is in } L, \text{ where } w \in L \text{ if and only if } Q \subseteq \overline{L}. \tag{5.1}$$

The other case is symmetric to this one.

Next, M' considers each word $q \in Q$ and does the following. It computes all words queried by M on input q. Let $q_1, q_2, \ldots, q_k$ be an enumeration of these words. Then, as described above M' finds out which of the conditions in Definition 5.2.6 holds. In both cases, M' is able to determine membership of w or q in L with at most one query to L which is smaller than w with respect to $\preceq$.

Case 1. Assume that the first condition holds, i.e.

$$\text{at most one query is in } L, \text{ where } q \in L \text{ if and only if } q_i \in \overline{L} \text{ for all } i \in [k] \tag{5.2}$$

For every $i \in [k]$, M' computes a pool $P = f(w, q_i)$. If $P = \text{equ}_2$, then $w \in L$ if and only if $q_i \in L$ and it queries q_i in order to determine whether $w \in L$ or not. Since this word has been queried by M on input q, and q has been queried by M on input w, it holds that $q_i \prec w$. If $P = \text{top}_2$, then q must not be in L. Assume that this is not the case, i.e. $q \in L$. Then, $w \notin L$ by Equation (5.1) which implies $q_i \in L$ because $f(w, q_i) = \text{top}_2$. But then, q must not be in L by Equation (5.2) which is a contradiction to $q \in L$ as assumed. In the worst case, we have $P = \text{bottom}_2$ for every $i \in [k]$. In this case, w can not be in L since if $w \in L$ we would have $q_i \notin L$ for all $i \in [k]$ which implies $q \in L$ by Equation (5.2), and this implies $w \notin L$ by Equation (5.1) which is a contradiction.

Case 2. Assume that the second condition holds, i.e.

$$\text{at most one query is in } \overline{L}, \text{ where } w \in L \text{ if and only if } q_i \notin L \text{ for one } i \in [k]. \tag{5.3}$$

For every $i \in [k]$, M' computes a pool $P = f(w, q_i)$. If $P = \text{equ}_2$, then $w \in L$ if and only if $q_i \in L$ and it queries q_i in order to determine whether $w \in L$ or not. Since this word has been queried by M on input q, and q has been queried by M on input w, it holds that $q_i \prec w$. If $P = \text{bottom}_2$, then at most one of both words, w or q_i, can be in L. Assume that w is in L. Then, q_i must not be in L which implies $q \in L$ by Equation (5.3), and this implies once more by Equation (5.1) that $w \notin L$. This is a contradiction, hence if $P = \text{bottom}_2$, then M' must reject w. If $P = \text{top}_2$, then at least one of both words, w or q_i, must be in L. In this case, we have $q_i \in L$ since if q_i is not in L, then w has to be in L, and also q by Equation (5.3). This contradicts Equation (5.1). In the worst case M' computes for all $i \in [k]$ the pool top_2. We have seen that in this case we can decide membership of each word q_i in L. Thus, M' is able to determine membership of w or q in L as described above.

We have seen that M' can decide membership in L for w directly or for all words $q \in Q$ by querying at most one word that is smaller than w with respect to $\preceq$. In the latter case, deciding membership for w is possible as well by simulating M on input w while answering each query $q \in Q$ as determined. It is not hard to see that M' runs in time polynomial in $|w|$. Hence, L is 1-tt-self-reducible via M' and $\preceq$. □

Since every language in P is trivially self-reducible and contained in $\text{P}[\text{NONSEL}_2]$ we obtain from Lemma 5.1.2 and the preceding two lemmas the following corollary.

Corollary 5.2.9. *A language is self-reducible and contained in* $\text{P}[\text{NONSEL}_2]$ *if and only if it is in* P.

We do not know whether self-reducible languages in $\mathrm{P}[\mathrm{NONSEL}_m]$ are contained in P. But we will present a result for disjunctively and conjunctively self-reducible languages in $\mathrm{P}[\mathrm{NONSEL}_m]$ in the next subsection.

5.2.3 Results for Disjunctive and Conjunctive Self-Reducibilities

In Section 5.2.1, we proved that every self-reducible language in $\mathrm{P}[\mathrm{SMC}_2 \cap \mathrm{COSMC}_2]$ belongs to P. It would be nice to know whether this also holds for self-reducible languages in $\mathrm{P}[\mathrm{SMC}_m \cap \mathrm{COSMC}_m]$ for all m. If we replace "self-reducible" by "disjunctively or conjunctively self-reducible", then they are contained in P. This follows from the following two facts. The first one has been proven in [Tan99, Theorem 1.20], whereas the second one is actually a corollary of it because conjunctively self-reducible languages in $\mathrm{P}[\mathrm{SMC}_m]$ are complements of disjunctively self-reducible languages in $\mathrm{P}[\mathrm{COSMC}_m]$.

Fact 5.2.10.

- *A language is dtt-self-reducible and in* $\mathrm{P}[\mathrm{COSMC}_m]$ *if and only if it belongs to* P.
- *A language is ctt-self-reducible and in* $\mathrm{P}[\mathrm{SMC}_m]$ *if and only if it belongs to* P.

Corollary 5.2.11. *A language is disjunctively or conjunctively self-reducible and contained in* $\mathrm{P}[\mathrm{SMC}_m \cap \mathrm{COSMC}_m]$ *if and only if it belongs to* P.

Similarly, it would be nice to generalize the result from Section 5.2.2 to arbitrary tuple lengths and answer the open question from Beigel, Kummer, and Stephan mentioned above. At this time, we know only of a result which uses disjunctive or conjunctive self-reducibility. It is stated in the following corollary of Lemma 4.3.13, Lemma 4.3.15, Theorem 6.9 in [BKS95a], and the fact that $\mathrm{P}[\mathrm{NONSEL}_m]$ is closed under complement (see Section 3.1 in [Nic01]).

Corollary 5.2.12.

- *A language is disjunctively or conjunctively self-reducible and in* $\mathrm{P}[\mathrm{NONSEL}_m]$ *if and only if it belongs to* P.
- *A language L is disjunctively or conjunctively self-reducible and Turing reducible to a language in* $\mathrm{P}[\mathrm{NONSEL}_m]$ *if and only if L belongs to* P.

5.3 Can Self-Reducible Partial Information Languages be not in P?

In the last section we have seen that self-reducible languages in certain partial information classes are in P. Is this the case for all self-reducible languages that belong to some partial information class? If not, we would have separated PSPACE from P since all self-reducible languages are in PSPACE by Theorem 2.4.9. Similarly, if we replace "self-reducible" by "disjunctively self-reducible" we would have separated NP from P. It is assumed that P is a proper subclass of NP. So, it is possible that some partial information class contains a disjunctively self-reducible language which is not in P.

A candidate for such a class could be $\mathrm{P}[\mathrm{SMC}_2]$. We do not know whether all disjunctively self-reducible languages in this class belong to P. However, as the next theorem tells us, they are at least contained in UP. I want to thank Arfst Nickelsen who pointed out that the original proof which included only 2-dtt-self-reducible languages (the self-reduction machine queries at most two words on every input) can be generalized to the present form.

Theorem 5.3.1. *Every dtt-self-reducible language in* $\mathrm{P}[\mathrm{SMC}_2]$ *is contained in* UP.

Proof. Let L be dtt-self-reducible via M and in $\mathrm{P}[\mathrm{SMC}_2]$ via a partial information function f. We are going to construct an unambiguous non-deterministic polynomial-time Turing machine M' that accepts L.

On an input w, M' walks a path in $\mathrm{ST}_M(w)$, and uses f to choose the next node to visit. It accepts w if and only if the last node on this path is in L. Note that $w \in L$ if and only if there exists a path from the root to a leaf such that the leaf is labeled with a word in L. Assume, M' already visited nodes with labels $w_1, w_2, \ldots, w_k$. Then, the label w_{k+1} of the next node to visit is determined as follows. If M poses on input w_k queries $q_1, q_2, \ldots, q_l$, then M' computes $f(q_i, q_j)$ for all i, j with $1 \leq i < j \leq l$. If for such a pair $f(q_i, q_j) = \mathrm{sel}_2$, then q_j is in L if q_i is in L. In this case, M' removes q_i from the list of queries. Let $q_{i_1}, \ldots, q_{i_m}$ be the queries remaining after these removals. If $m = 1$, then M' sets $w_{k+1} := q_{i_1}$. Otherwise we have $f(q_{i_{j_1}}, q_{i_{j_2}}) = \mathrm{bottom}_2$ for all j_1, j_2 with $1 \leq j_1 < j_2 \leq m$. This means, at most one of these queries is in L. Then, M' chooses a $j \in [m]$ nondeterministically and sets $w_{k+1} := q_{i_j}$.

Clearly, M' has on input w at most one accepting computation and is therefore unambiguous. □

As mentioned above, it is unknown whether every disjunctively self-reducible language

in $P[SMC_2]$ belongs to P. However, there exists a language L such that relative to L every disjunctively self-reducible language in $P[SMC_2]$ belongs to P. We can choose L to be any PSPACE-complete language. Then we have $P^L = NP^L$ by Theorem 14.4 in [Pap94] and the claim follows from the fact that relative to L every disjunctively self-reducible language is in NP (by nearly the same proof as for Theorem 2.4.9). On the other hand, in the proof of Theorem 7.1 in [BKS95a], a language K is constructed such that relative to K there exists a disjunctively self-reducible language in $P[SMC_2]$ that does not belong to P. We summarize these results in the following fact.

Fact 5.3.2.

1. *There is a language L such that relative to L every dtt-self-reducible language in* $P[SMC_2]$ *belongs to* P.

2. *There is a language K such that relative to K there exists a dtt-self-reducible language in* $P[SMC_2]$ *that is not in* P.

Like the proof of Theorem 2.4.9, many proofs in complexity theory still work in the presence of arbitrary oracles. We say that they relativize. It is often seen as evident that statements that are true relative to some language and false relative to another one are hard to prove. For those ones, we need non-relativizing proofs since otherwise we would end up in a contradiction. I feel that any constructive proof of the statement "disjunctively self-reducible languages in $P[SMC_2]$ belong to P" would relativize (like all proofs in this thesis relativize) as well as any proof of the opposite statement by diagonalization. So, alternative proof techniques have to be found first to show its truth or falsehood. For more on relativization and its role in complexity theory there is an interesting paper written by Hartmanis et al. [HCC+92].

Chapter 6

Conclusion

This thesis is about self-reducible languages with partial information algorithms. Given a family $\mathcal{F}$, we asked whether all self-reducible languages in $\mathrm{P}[\mathcal{F}]$ belong to P. Positive answers to this question have been given for most 2-families. Answers for some families with tuple lengths greater than two can be found in the literature. However, we obtained no results for such families, except for disjunctively or conjunctively self-reducible languages in partial information classes produced by some of those families. Negative answers to the above question were unlikely to be given since this would imply that P is a proper subclass of PSPACE, NP, or co-NP. Therefore, results in this respect have only been obtained for some relativized world.

In the following sections we summarize the main results of this thesis and present questions that are left open as well as conjectures.

6.1 Summary of Main Results

We showed that self-reducible languages in $\mathrm{P}[\mathrm{NONSEL}_2]$ and $\mathrm{P}[\mathrm{SMC}_2 \cap \mathrm{COSMC}_2]$ characterize the languages in P. The first result is interesting since NONSEL_2 is the largest 2-family in normal form that does not include SEL_2 and therefore completely answers the above question for all those families. It furthermore gives a partial answer to an open question from [BKS95a], which asks whether all self-reducible easily m-countable languages are in P. The second result extends the corresponding result for p-selective languages obtained in [BvHT93].

The proofs were in two steps. First, self-reducible languages in those classes have been shown to be truth-table self-reducible. Then, truth-table self-reducible languages in those classes were broken down to 1-tt-self-reducible languages which implies membership in P. For the first step, we proved two results which are interesting in its own.

The first one – that does not only hold for self-reducibilities, but for reducibilities in general – says that Turing and truth-table reduction closures of languages in P[NONSEL$_m$] coincide. A corollary is that self-reducible languages in P[NONSEL$_m$] are truth-table self-reducible. The second result has been obtained in Section 5.1 and tells us that self-reducible languages in P[$(2m-1)$–SIZE$_m$] are already truth-table self-reducible. For example, p-selective languages are in P[$(2m-1)$–SIZE$_m$] for all m, and for $m = 2$ it contains all partial information classes produced by 2-families distinct to 4–SIZE$_2$. Both results might be very helpful for future research in this direction.

For families with tuple lengths larger than two we obtained no results which answer the question posed at the beginning of this thesis. However, some results concerning p-selective, p-cheatable, and easily approximable languages can be found in the literature (see [BvHT93], [ABG00, GJY93], and [BKS95a], respectively). We were able to obtain as a corollary that disjunctively and conjunctively self-reducible languages in P[NONSEL$_m$] characterize the languages in P. That disjunctively and conjunctively self-reducible languages in P[SMC$_m \cap$ COSMC$_m$] characterize P was known before.

We also considered self-reducible languages in P[SMC$_2$], but had to learn that it is possibly very hard to show that they are in P by the same techniques employed for the other results. We would end up in a contradiction if we use relativizing proofs: Beigel, Kummer, and Stephan constructed a language L such that relative to L there is a disjunctively self-reducible language in P[SMC$_2$] that is not in P, and relative to any PSPACE-complete language K every such language belongs to P. The problem is that a lot of proofs in complexity theory relativize. At least, we were able to show that disjunctively self-reducible languages in P[SMC$_2$] belong to UP.

We finish this section with Figure 6.1 that shows results regarding 2-families obtained in this thesis.

6.2 Open Questions and Conjectures

The question whether self-reducible languages in P[$\mathcal{F}$] are truth-table self-reducible has been fully answered by Corollary 5.1.1 for all m-families $\mathcal{F}$ in normal form with SEL$_m \not\subseteq \mathcal{F}$. Lemma 5.1.2 gives a positive answer for some families $\mathcal{F}$ with SEL$_m \subseteq \mathcal{F} \subseteq (2m-1)$–SIZE$_m$. However, we do not know whether this is optimal, i.e. whether $(2m-1)$–SIZE$_m$ is the largest or the only m-family $\mathcal{F}$ for which self-reducible languages in P[$\mathcal{F}$] are truth-table self-reducible. Therefore, we pose the following question.

Open Question 6.2.1. *Is the result of Lemma 5.1.2 optimal? If not, for which m-*

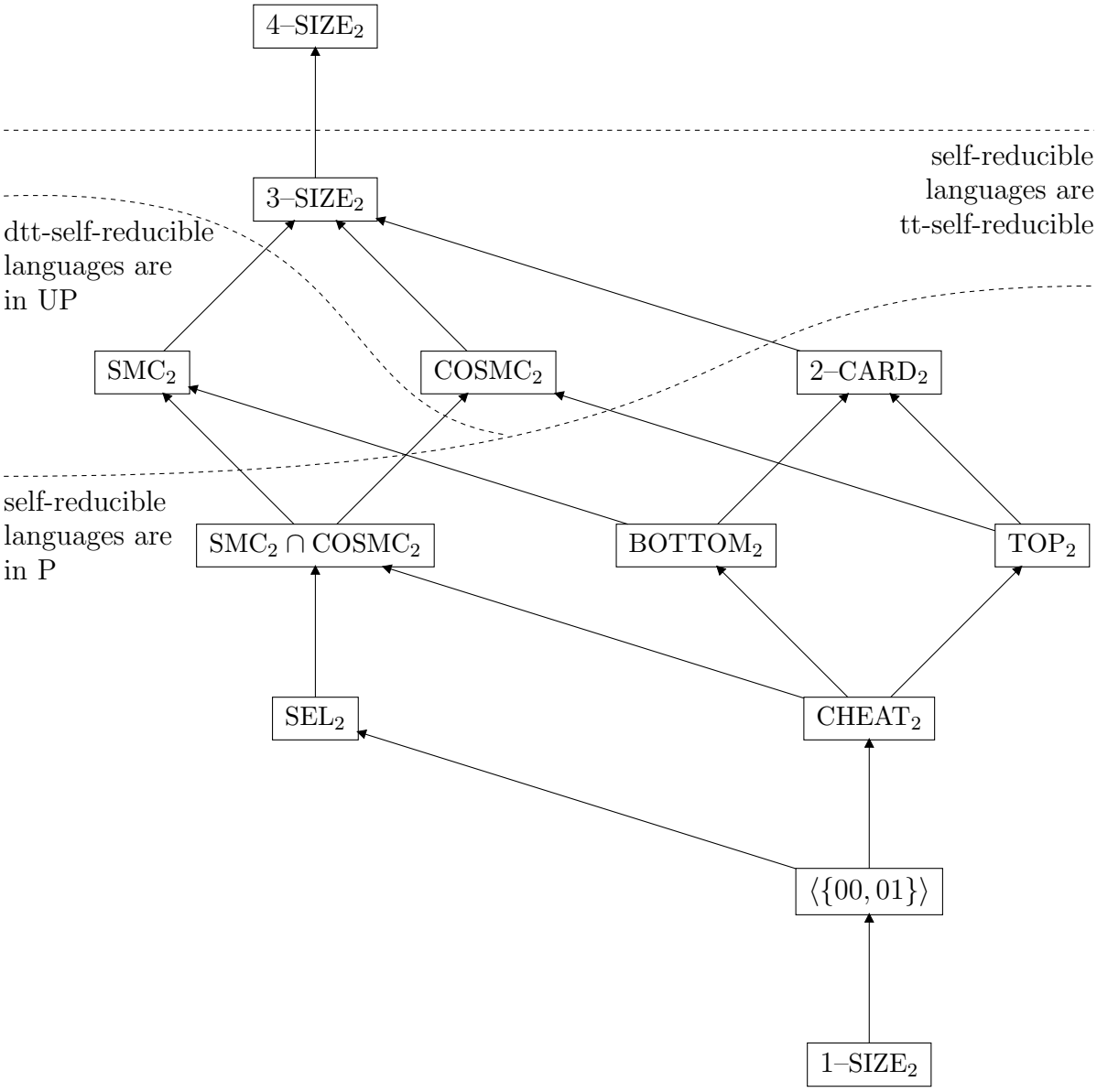


Figure 6.1: This diagram shows all partial information classes produced by 2-families in normal form as in Figure 3.1. Dotted lines separate partial information classes which contain only (disjunctively) self-reducible languages that are in P, UP or are tt-self-reducible, respectively, from partial information classes where this is not known. All classes below a line have the property that is written directly under that line.

families $\mathcal{F}$ *with* $\mathrm{SEL}_m \subseteq \mathcal{F}$ *that are larger than or distinct to* $(2m-1)$–SIZE_m *are self-reducible languages in* $\mathrm{P}[\mathcal{F}]$ *tt-self-reducible?*

In Section 5.2.1, we showed that self-reducible languages in $\mathrm{P}[\mathrm{SMC}_2 \cap \mathrm{COSMC}_2]$ characterize P. What about self-reducible languages in $\mathrm{P}[\mathrm{SMC}_2]$? Showing that such languages belong to P or not might be hard as argued in Section 5.3. As also argued, it might even be hard if we restrict to disjunctively self-reducibility. Nevertheless, I think that it is interesting to further investigate the complexity of such languages.

Open Question 6.2.2. *Does every (disjunctively) self-reducible language in* $\mathrm{P}[\mathrm{SMC}_2]$ *belong to* P*?*

Another question is whether the result from Section 5.2.1 can be generalized to other tuple lengths. That is, do self-reducible languages in $\mathrm{P}[\mathrm{SMC}_m \cap \mathrm{COSMC}_m]$ belong to P? We know already that disjunctively and conjunctively self-reducible languages in $\mathrm{P}[\mathrm{SMC}_m \cap \mathrm{COSMC}_m]$ belong to P. The next goal could be to show that (positive) truth-table self-reducible languages in $\mathrm{P}[\mathrm{SMC}_m \cap \mathrm{COSMC}_m]$ are also in P.

Open Question 6.2.3. *Does every (positive) truth-table self-reducible language that is in* $\mathrm{P}[\mathrm{SMC}_m \cap \mathrm{COSMC}_m]$ *belong to* P*?*

We finish this thesis with a conjecture that I strongly believe to be true.

Conjecture 6.2.4. *Every self-reducible language in* $\mathrm{P}[\mathrm{NONSEL}_m]$ *belongs to* P*.*

It is motivated by the following results obtained in this thesis. Firstly, in Section 5.2.2 we proved that self-reducible languages in $\mathrm{P}[\mathrm{NONSEL}_2]$ belong to P. Secondly, we know for other tuple lengths that disjunctively and conjunctively self-reducible languages in $\mathrm{P}[\mathrm{NONSEL}_m]$ are in P. And finally, we also know that self-reducible languages in $\mathrm{P}[\mathrm{NONSEL}_m]$ are truth-table self-reducible. All what is left to show the truth of this conjecture is to show that truth-table self-reducible languages in $\mathrm{P}[\mathrm{NONSEL}_m]$ are disjunctively or conjunctively self-reducible, or in P.

Bibliography

[ABG00] Amihood Amir, Richard Beigel, and William I. Gasarch. Some connections between bounded query classes and non-uniform complexity (long version). Technical Report TR00-024, Electronic Colloquium on Computational Complexity, www.eccc.uni-trier.de/eccc, 2000.

[BDG88] José Luis Balcázar, Josep Díaz, and Joaquim Gabarró. *Structural Complexity* I. Number 11 in EATCS Monographs on Theoretical Computer Science. Springer, 1988.

[Bei87] Richard Beigel. *Query-limited reducibilities*. PhD thesis, Stanford University, Stanford, USA, 1987.

[BGK95] Richard Beigel, William I. Gasarch, and Efim Kinber. Frequency computation and bounded queries. *Theoretical Computer Science*, 163(1&2):177–192, 1995.

[BH77] L. Berman and Juris Hartmanis. On isomorphism and density of NP and other complete sets. *SIAM Journal on Computing*, 6:305–327, 1977.

[BKS95a] Richard Beigel, Martin Kummer, and Frank Stephan. Approximable sets. *Information and Computation*, 120(2):304–314, 1995.

[BKS95b] Richard Beigel, Martin Kummer, and Frank Stephan. Quantifying the amount of verboseness. *Information and Computation*, 118(1):304–314, 1995.

[BTvEB93] Harry Buhrman, Leen Torenvliet, and Peter van Emde Boas. Twenty questions to a P-selector. *Information Processing Letters*, 4(4):201–204, 1993.

[BvHT93] Harry Buhrman, van Helden, and Leen Torenvliet. P-selective self-reducible sets: A new characterization of P. In *Annual IEEE Conference on Computational Complexity (formerly Annual Conference on Structure in Complexity Theory)*, volume 8, pages 44–51, 1993.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In ACM, editor, *Conference record of third annual ACM Symposium on Theory of Computing: papers presented at the symposium, Shaker Heights, Ohio, May 3, 4, 5, 1971*, pages 151–158, New York, NY, USA, 1971. ACM Press.

[DK00] Ding-Zhu Du and Ker-I Ko. *Theory of Computational Complexity.* Wiley & Sons, 2000.

[GJY93] Judy Goldsmith, Deborah Joseph, and Paul Young. Using self-reducibility to characterize polynomial time. *Information and Computation*, 104:288–308, 1993.

[HCC+92] Juris Hartmanis, Richard Chang, Suresh Chari, Desh Ranjan, and Pankaj Rohatgi. Relativization: A revisionistic retrospective. *BEATCS: Bulletin of the European Association for Theoretical Computer Science*, 47, 1992.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2001.

[HN93] Albrecht Hoene and Arfst Nickelsen. Counting, selecting, and sorting by query–bounded machines. In Patrice Enjalbert, Alain Finkel, and Klaus W. Wagner, editors, *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS '93)*, volume 665 of *LNCS*, pages 196–205, Berlin, Germany, February 1993. Springer.

[HO02] Lane A. Hemaspaandra and Mitsunori Ogihara. *The Complexity Theory Companion.* Springer, 2002.

[Ko83] Ker-I Ko. On self-reducibility and weak P-selectivity. *Journal of Computer and System Sciences*, 26(2):209–221, apr 1983.

[Kum92] Martin Kummer. A proof of Beigel's cardinality conjecture. *The Journal of Symbolic Logic*, 57(2):677–681, 1992.

[Köb95] Johannes Köbler. On the structure of low sets. In *Proceedings of the 10th Annual Structure in Complexity Theory Conference*, pages 246–261, 1995.

[MP79] A. Meyer and M. Paterson. With what frequency are apparently intractable problems difficult? Technical Report MIT/LCS/TM-126, Massachusetts Institute of Technology, 1979.

[Nic01] A. Nickelsen. *Polynomial Time Partial Information Classes.* W&T Verlag, 2001. Dissertation, TU Berlin, 1999, available at www.tal.cs.tu-berlin.de/nickelsen/.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Sch76] C. P. Schnorr. Optimal algorithms for self-reducible problems. In S. Michaelson and R. Milner, editors, *Third International Colloquium on Automata, Languages and Programming*, pages 322–337, University of Edinburgh, 20–23 July 1976. Edinburgh University Press.

[Sel79] Alan L. Selman. P-selective sets, tally languages, and the behaviour of polynomial time reducibilities on NP. *Mathematical Systems Theory*, 13(1):55–65, 1979.

[Sel82a] Alan L. Selman. Analogues of semirecursive sets and effective reducibilities to the study of NP complexity. *Information and Control*, 52(1):36–51, 1982.

[Sel82b] Alan L. Selman. Reductions on NP and p-selective sets. *Theoretical Computer Science*, 19:287–304, 1982.

[Tan99] Till Tantau. Combinatorial Representations of Partial Information Classes and their Truth-Table Closures. Diplomarbeit Informatik, Technische Universität Berlin, available at www.eccc.uni-trier.de/eccc, Germany, 1999.

[Tod91] Seinosuke Toda. On polynomial-time truth-table reducibilities of intractable sets to p-selective sets. *Mathematical Systems Theory*, 24:69–82, 1991.