Classification of Search Problems and Their Definability in Bounded Arithmetic

by

Tsuyoshi Morioka

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

ISSN 1433-8092

Abstract

Classification of Search Problems and Their Definability in Bounded Arithmetic

Tsuyoshi Morioka

Master of Science

Graduate Department of Computer Science

University of Toronto

2001

We present a new framework for the study of search problems and their definability in bounded arithmetic. We identify two notions of complexity of search problems: verification complexity and computational complexity. Notions of exact solvability and exact reducibility are developed, and exact Σ_i^b-definability of search problems in bounded arithmetic is introduced. We specify a new machine model called the oblivious witness-oracle Turing machines.

Based on work of Buss and Krajicek, we present a type-2 search problem ITERATION (ITER) that characterizes the class PLS and the exactly Σ_1^b-definable search problems of the theory T_2^1. We show that the type-2 problems of Beame et. al. are not Turing reducible to ITER. The separations of the corresponding type-2 classes and the unprovability of certain combinatorial principles in a relativized version of T_2^1 are obtained as corollaries.

We also present the first characterization of the exactly Σ_2^b-definable search problems of S_2^1 and T_2^1.

Acknowledgements

It has been an absolute pleasure to do a graduate study under the supervision of Steve Cook, to whom I'd like to express my greatest gratitude. His support and encouragement have been invaluable at all stages of my research, and discussions with him have been always inspiring and delightful.

I would like to thank Toni Pitassi for reading a draft of this thesis and suggsting some improvements.

I am so lucky to have many theory-oriented grad students as colleagues . Talking with them about research is so much fun, and talking with them about other things is even more fun.

I would like to thank my best friend Masami Migiyama for her support throughout the writing of this thesis.

Finally, I would like to thank my parents, Takeshi Morioka and Fumiko Morioka, who have supported me throughout my life. They taught me how to learn, and they have been always encouraging me to pursue my own interests.

Contents

Chapter 1

Introduction

1.1 Background

Computational Complexity is the study of computational problems and the amount of resources required to solve them. Cook's introduction of NP-completeness in 1971 [Coo71] started decades of fruitful research, and since then numerous computational problems have been shown to be complete for, or at least classified into, various complexity classes. In particular, a great number of combinatorial problems that arise naturally in practical settings are shown to be NP-complete.

The interesting aspect of Computational Complexity is that it has been focused mainly on *decision problems*, or the problems of deciding whether the input has a certain property, although most combinatorial problems are more naturally formulated as *search problems*, or the problems of finding an object with a certain property. For example, when we wish to analyze the complexity of $CLIQUE\text{-}Search(G, k)$, the problem of finding a clique of size $\geq k$ in the input graph G, we first transform it to the corresponding decision problem $CLIQUE(G, k)$ of deciding whether G have a clique of size $\geq k$. There was supposed to be little loss in doing so, since most search problems have corresponding decision problems that are computationally equivalent in the sense that if one is feasible, so is the other, or, more formally, they are Turing reducible to each other. Thus, studying the complexity of decision problems should reveal the complexity of the corresponding search problems.

However, it has been realized that search problems have deeper level of structure that are lost when they are transformed to decision problems, and the existence of search problems for which there are no computationally equivalent decision problems has been demonstrated [BCE+98]. Thus, it is necessary to study search problems directly, and several frameworks for this purpose have been proposed. The following is a list of such frameworks on which our work is based.

- The classes $\mathbf{FP}^{\mathbf{NP}}$ and $\mathbf{FP}^{\mathbf{NP}}[O(\log n)]$ [Kre88]
- **FNP**, **FP**, **TFNP**, and subclasses of **TFNP** [JPY88, Pap90, MP91, Pap94b, BCE+98]
- The hierarchy $\Box_i^p$ of functions [Bus86]
- $\mathbf{FP}^{\mathbf{\Sigma}_i^p}[wit, f(n)]$ [BKT93, Kra95]

We should emphasize that the above list is grossly incomplete and there are many other frameworks that we do not discuss in this report, such as [KPS90, Kre92, Sel94, GKR95].

Krentel [Kre88] studied the complexity of computation of optimal values in terms of the number of queries to an NP oracle required to perform it in polynomial time. He came up with the classes $\mathbf{FP}^{\mathbf{NP}}$ and $\mathbf{FP}^{\mathbf{NP}}[O(\log n)]$: $\mathbf{FP}^{\mathbf{NP}}$ is the class of functions that can be computed in deterministic polynomial time with polynomially many queries to an **NP** oracle, and $\mathbf{FP}^{\mathbf{NP}}[O(\log n)]$ is obtained from $\mathbf{FP}^{\mathbf{NP}}$ by restricting the number of NP queries to $O(\log n)$. One of his many results is that that $TSP\text{-}COST(G)$, the problem of computing the cost of an optimal tour of a graph, is complete for $\mathbf{FP}^{\mathbf{NP}}$, while $MAX\text{-}CLIQUE\text{-}SIZE(G)$ is complete for $\mathbf{FP}^{\mathbf{NP}}[O(\log n)]$, with respect to a suitable notion of reducibility. Moreover, he showed that $TSP\text{-}COST \in \mathbf{FP}^{\mathbf{NP}}[O(\log n)]$ implies

P $=$ **NP**; hence, in terms of the number of **NP** queries required, $TSP\text{-}COST(G)$ is strictly harder than $MAX\text{-}CLIQUE(G)$ assuming **P** $\neq$ **NP**, even though the decision counterparts of both search problems are NP-complete.

Krentel's framework is a nice setting for the study of functions, that is, search problems with a unique solution. However, it does not cover search problems with multiple solutions. Meggido and Papadimitriou [MP91] introduced the classes **FNP**, **FP** and **TFNP** that easily capture the multiplicity of solutions.

Let $R(x, y)$ be a polynomially balanced, polynomial-time predicate. Then R gives rise to an *NP search problem* Q: given x, find y such that $R(x, y)$ holds. **FNP** is the class of *NP* search problems [MP91]. For example, the problem $SAT\text{-}Search(\phi)$ of finding a satisfying solution for a propositional formula ϕ is in **FNP**. Note that $SAT\text{-}Search(\phi)$ has no solution if ϕ is unsatisfiable. We say a search problem is *total* if every instance of it has a solution. For example, $SAT\text{-}Search(\phi)$ is not total.

TFNP is the subclass of **FNP** containing total search problems [MP91]. **TFNP** contains many natural combinatorial problems, but it is a semantic class and therefore is not expected to have complete problems [BCE$^+$98]. However, **TFNP** contains several interesting syntactic subclasses such as **PLS** [JPY88, Yan97], **PPP**, **PPA**, **PPAD**, and **PPADS** [Pap90, Pap94b, BCE$^+$98]. Each class is defined based on the combinatorial principle that guarantees the totality of problems in the class. For example, **PLS** is the class of local search problems whose totality follows from the combinatorial principle "every directed acyclic graph has a sink", and **PPP** is the class of problems whose totality is guaranteed by the injective pigeonhole principle "there is no injective mapping from $n + 1$ to n". **TFNP** and its subclasses are interesting because they contain search problems that have no computationally equivalent decision problems [BCE$^+$98].

FP is the subclass of **FNP** containing problems for which there is a polynomial-time algorithm that finds a solution [MP91]. Thus, the problem $HORNSAT\text{-}Search(\phi)$ of finding a satisfying truth assignment for a propositional Horn formula is in **FP**.

Beame, Cook, Edmonds, Impagliazzo, and Pitassi [BCE$^+$98] introduced type-2 total search problems that capture certain combinatorial principles, and they presented an elegant type-2 characterization of the classes **PPA**, **PPAD**, **PPADS**, and **PPP**. By exhibiting the existence or nonexistence of reduction between the type-2 problems, they showed all possible separations of these classes in a generic relativized world. Furthermore, they proved that none of their type-2 problems is Turing equivalent to any decision problem.

In his thesis published in 1986, Buss [Bus86] developed a hierarchy of theories of *bounded arithmetic* and showed its connection to the polynomial hierarchy. In doing so, he also introduced a hierarchy of classes of function as follows: $\square_1^p$ is the class of polynomial-time computable functions, and $\square_{i+1}^p$, $i \geq 1$, is the class of functions computable in polynomial-time with access to a predicate from $\mathbf{\Sigma}_i^p$. He showed that a function f is in $\square_i^p$ if and only if it is Σ_i^b-definable in S_2^i.

More results on the characterization of $\forall\exists\Sigma_i^b$-consequences of bounded arithmetic theories followed [Kra95]. One particularly interesting result is by Buss and Krajicek [BK94]: they showed that the $\forall\exists\Sigma_1^b$-consequences of T_2^1 corresponds to **PLS** in a certain sense, suggesting the existence of other search classes corresponding to $\forall\exists\Sigma_i^b$-consequences of various theories. Chiari and Krajicek present in [CK98] many results in this direction.

Buss, Krajicek, and Takeuti [BKT93] introduced the notion of *witness oracles*, an extension of oracles whose positive answers are accompanied by an object that witnesses the correctness of the answer. They defined classes $\mathbf{FP}^{\mathbf{\Sigma}_i^p}[wit, f(n)]$ of total search problems whose solution can be found in polynomial time using at most $f(n)$ witness queries to a $\mathbf{\Sigma}_i^p$ predicate. Krajicek [Kra95] characterized $\forall\exists\Sigma_2^b$-consequences of S_2^1 and T_2^1 by classes of this form.

Thanks to the success of the above lines of research, much has been known about the search problems and their definability in bounded arithmetic. However, because of inconsistent terminology and implicit assumptions, it is hard to see how the results in one framework relate to the results in another framework.

For example, notice the different usages of the term '**FP**'. In the sense of Krentel [Kre88], **FP** denotes the class of polynomial-time total functions. However, Meggido and Papadimitriou [MP91] defines **FP** to be the class of search problems such that it is easy to recognize solutions and at least one solution can be computed in polynomial time. Their **FP** can be modified to be a class of total search problems, but in its original formulation, it contains nontotal problems. And **FP** in $\mathbf{FP}^{\mathbf{\Sigma}_i^p}[wit]$ of Buss, Krajicek, and Takeuti [BKT93] denotes the class of total search problems such that one of the solutions can be found in polynomial time. The problems in this class can have solutions that are impossible to compute or recognize.

1.2 Our Work

The purpose of our work is to provide a unified framework for classification of search problems and their definability in bounded arithmetic. Our framework incorporates most of the above classes without any inconsistency or confusion, even though some classes are called by different names in our setting.

In Chapter 2, we present the basic definition of search problems, followed by introduction of the the notions of *verification complexity*, *computational complexity*, *exact solvability*, and *exact reducibility*.

Verification complexity of a search problem is the hardness of verifying or recognizing solutions. Magiddo and Papadimitriou's **TFNP** [MP91] is called in our setting **VP** (verifiable in polynomial time), and it is at the bottom of our verifiability hierarchy $\mathbf{VPH} = \bigcup_{i \geq 0} \mathbf{V\Sigma}_i^p$, where $\mathbf{V\Sigma}_i^p$ is the class of total search problems for which the verification of solutions is in $\mathbf{\Sigma}_i^p$ of the polynomial hierarchy **PH** [Sto77, Wra77, Pap94a]. We show that **VPH** collapses if and only if **PH** collapses.

Computational complexity of a search problem is the hardness of finding a solution, that is, the hardness of 'solving' the search problem. We show that the use of an intuitive notion of solvability results in the notion of computational complexity that is not meaningfully related to verification complexity. We introduce a new notion of solvability called *exact solvability.*

Our work is based on a new model of computation called the *oblivious witness-oracle Turing machines.* It is as powerful as the ordinary witness-oracle Turing machines of Buss, Krajicek, and Takeuti [BKT93] with respect to simple solvability. Interestingly, the power of oblivious machines seems to be more restricted with respect to exact solvability, and it allows a more sensitive treatment of computational complexity of search problems.

We develop a collection of search classes of the form $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[oblivious, f(n)]$ that capture the hardness of exactly solving the search problems. We show that our classes contain natural combinatorial problems.

Reducibility between problems is an essential tool in the study of complexity. However, we show that the commonly-used *many-one reducibility* is not the right tool for our purpose. We come up with a stronger notion of *exact many-one reducibility* that corresponds nicely to exact solvability.

Chapter 3 is devoted to the type-2 characterization of various search classes. We present a type-2 total search problem $ITERATION$ and use it to characterize the class $E[\mathbf{PLS}]$, the smallest class closed under exact many-one reduciblity. Using the type-2 problems of [BCE$^+$98], we provide type-2 characterization of **PPA**, **PPAD**, **PPADS**, and **PPP** that is slightly different from that of Beame et. al.

Then we present the most significant of our theorems: we show that any of the type-2 problems of Beame et. al. is not Turing reducible to $ITER$. This theorem has two important consequences. The first is the type-2 separation of $E[\mathbf{PLS}]$ from the other search classes corresponding to the other type-2 problems. The second, which is stated in Chapter 4, is the unprovability of certain combinatorial principles in a relativized theory

$T^1_2(\alpha)$ of bounded arithmetic. The latter suggests a methodology to obtain separations of theories of bounded arithmetic via separations of type-2 search problems.

The chapter ends with the introduction of two new type-2 problems: $MAXIMIZER$ and $L\text{–}MAXIMIZER$. They characterize $\mathbf{EP}^{\mathbf{NP}}[oblivious]$ and $\mathbf{EP}^{\mathbf{NP}}[oblivious, O(\log n)]$, respectively.

Chapter 4 connects our framework of search problems with Buss's theories of bounded arithmetic. After a brief introduction, we present the notion of *exact* Σ^b_i*-definability* of search problems in bounded arithmetic, that corresponds to exact solvability and exact reducibility. This definability notion is from [BKT93], where it is called *strong* Σ^b_i*-definability*. Then, we show the equalities between the classes of the exact Σ^b_1-definable search problems of S^1_2, T^1_2, and variants of S^1_2 and the search classes that we have introduced in Chapter 3.

Lastly, we characterize the classes of exactly Σ^b_2-definable search problems of S^1_2 and T^1_2 by $L\text{–}MAXIMIZER$ and $MAXIMIZER$. This is the first characterization of these classes, although a similar result has been obtained by Chiari and Krajicek [CK98] in a slightly different context.

1.3 Preliminaries

We work with the natural numbers $\mathbb{N}$. For $n \in \mathbb{N}$, $|n|$ denotes the length of binary representation of n; hence $|n| = \lceil \log\,(n+1) \rceil$. For $a \in \mathbb{N}$, $\mathbb{N}^{\leq |a|}$ denotes $\{n \in N : |n| \leq |a|\}$, the set of all numbers that can be represented using $|a|$ bits.

Let A be any unary predicate. A *deterministic oracle Turing machine with access to* A, or simply an *oracle Turing machine*, is a deterministic Turing machine that is allowed to ask *queries* of the form '$A(z)$?', for which the oracle returns $q \in \{0, 1\}$, where $q = 1$ if $A(z)$ holds, and $q = 0$ otherwise. A *nondeterministic oracle Turing machine* is defined similarly.

A deterministic, nondeterministic, or oracle Turing machine M is said to be *polynomial-time* if there exists a constant c such that M on inputs of length n halts within $O(n^c)$ steps.

A predicate $R(\overline{a})$ is said to be polynomial-time computable if it can be computed by a deterministic polynomial-time Turing machine. We say R is *polynomial-time in* A if a polynomial-time oracle Turing machine with access to A decides R.

Let $\mathbf{C}, \mathbf{D}$ be complexity classes. Then $\mathbf{C}^{\mathbf{D}}$ is the class of predicates decidable by a deterministic or nondeterministic oracle Turing machine that runs within the time or space bound of complexity class $\mathbf{C}$ with access to a predicate in $\mathbf{D}$.

Let $R(x, y)$ be a binary predicate on the natural numbers. We say R is *polynomially balanced* if there exists a constant c such that, for all $a, b \in \mathbb{N}$, $R(a, b)$ implies $|b| \leq |a|^c + c$ [Pap94a]. Note that, if R is polynomially balanced, it can be expressed as $R(x, y) \equiv |y| \leq |x|^c + c \wedge R'(x, y)$ for some predicate R'.

Theorem 1.3.1 *[Pap94a] Let $R(x)$ be a predicate. $R \in \mathbf{NP}$ if and only if there is a polynomially balanced, polynomial-time predicate $S(x, y)$ such that $R(x) \iff (\exists y)S(x, y)$.*

The *polynomial hierarchy* is the sequence of classes of predicates defined as follows: First, $\mathbf{\Sigma}_0^p = \mathbf{\Pi}_0^p = \mathbf{\Delta}_0^p = \mathbf{P}$; and for $i \geq 0$,

$$\begin{aligned}
\mathbf{\Sigma}_{i+1}^p &= \mathbf{NP}^{\mathbf{\Sigma}_i^p} \\
\mathbf{\Pi}_{i+1}^p &= \mathbf{coNP}^{\mathbf{\Sigma}_i^p} \\
\mathbf{\Delta}_{i+1}^p &= \mathbf{P}^{\mathbf{\Sigma}_i^p}.
\end{aligned}$$

Note that $\mathbf{\Sigma}_i^p \cup \mathbf{\Pi}_i^p \subseteq \mathbf{\Delta}_{i+1}^p \subseteq \mathbf{\Sigma}_{i+1}^p \cap \Pi_{i+1}^p$, for all $i \geq 0$. Finally, $\mathbf{PH} = \bigcup_{i \geq 0} \mathbf{\Sigma}_i^p$. We say *the polynomial hierarchy collapses at the ith level* if $\mathbf{PH} = \mathbf{\Sigma}_i^p$. See [Sto77, Wra77, Pap94a] for more information on the polynomial hierarchy.

Theorem 1.3.2 *[Wra77, Pap94a] Let $R(x)$ be a predicate, and $i \geq 1$. $R \in \mathbf{\Sigma}_i^p$ if and only if there exists a polynomially balanced predicate $S(x, y) \in \mathbf{\Pi}_{i-1}^p$ such that $R(x) \iff (\exists y)S(x, y)$.*

A function $f : \mathbb{N} \mapsto \mathbb{N}$ is said to be *polynomial-time* if there exists a deterministic polynomial-time Turing machine M such that M on x halts with the binary representation of $f(x)$ in its output tape. We say f is *polynomial-time in A* if an oracle machine with access to A computes the value of f in polynomial-time.

$\square_1^p$ is defined by Buss to be the class of all polynomial-time functions and $\square_{i+1}^p$ is the class of functions polynomial in a $\mathbf{\Sigma}_i^p$ predicate [Bus86, Bus98].

The following definitions are from [Bus86, Kra95, Bus98]. Let

$$L_{BA} = \{0, S, +, \cdot, \lfloor \frac{x}{2} \rfloor, |x|, \#, \leq\}$$

be the *language of bounded arithmetic*, where 0 is a constant, S is the successor function, $|x| = \lceil \log_2 x \rceil$ denotes the binary length of x, and $x\#y = 2^{|x|\cdot|y|}$ is the smash function. Note that for every term $t(a)$ in the language L_{BA}, there exists a constant c such that $|t(x)| \in O(|x|^c)$.

A quantifier is said to be *sharply bounded* if it is of the form $(\exists x \leq |t(\overline{a})|)$ or $(\forall x \leq |t(\overline{a})|)$, where t is a term in the language L_{BA}. A quantifier is said to be *bounded* if it is of the form $(\exists x \leq t(\overline{a}))$ or $(\forall x \leq t(\overline{a}))$.

The sets $\Sigma_0^b = \Pi_0^b$ are the sets of formulas in which all quantifiers are sharply bounded. For $i \geq 1$, the set Σ_i^b is the set containing $\Sigma_{i-1}^b \cup \Pi_{i-1}^b$ and closed under $\wedge$, $\vee$, sharply bounded quantifications and the existential bounded quantification. The set Π_i^b is defined similarly except that it is closed under the universal bounded quantification, instead of the existential bounded quantification.

Let $\underline{\mathbb{N}}$ be the standard model of arithmetic, in which nonlogical symbols of L_{BA} assume their standard interpretation. We say that a formula $\phi(a, b)$ *represents* a predicate $R(a, b)$ if and only if

$$\underline{\mathbb{N}} \models \phi(s_n, s_m) \iff R(n, m),$$

where s_n, s_m are numerals representing $n, m \in \mathbb{N}$, respectively.

Theorem 1.3.3 *[Bus86, Kra95, Bus98] A predicate R is in $\mathbf{\Sigma}_i^p$ if and only if there exists a Σ_i^b formula ϕ that represents R.*

Chapter 2

Search Problems

2.1 Verification Complexity of Search Problems

The purpose of our study is to present a unified framework for the study of complexity of search problems, and thus we begin with the formal definition of search problems. The following is a slight generalization of the definition in [MP91, BCE$^+$98]; a similar definition appears in [GJ79].

Definition 2.1.1 *[BCE$^+$98] Let $R \subseteq \mathbb{N}\times\mathbb{N}$ be any polynomially balanced binary predicate on the natural numbers. Then R defines a* search problem *Q_R which associates with every $x \in \mathbb{N}$ the set $Q_R(x) = \{y : R(x,y)\}$. We say that $Q_R(x)$ is the set of* solutions *for an* instance *x and that R is a* defining predicate *of Q_R.*

Q_R is said to be total *if $|Q_R(x)| > 0$ for all $x \in \mathbb{N}$. We say Q_R is a* function problem *if $|Q_R(x)| \leq 1$ for all $x \in \mathbb{N}$.* □

Note that, since we require a defining predicate R to be polynomially balanced, for every search problem Q_R there exists a constant c such that $Q(x) \subseteq \mathbb{N}^{\leq |x|^c + c}$ for all x.

The subscript of Q_R is omitted when it is not necessary to indicate a defining predicate of the search problem.

Definition 2.1.2 *For any predicate $R(x)$, let f_R be the* characteristic function problem, *defined as*

$$f_R(x) \quad = \quad \{y : (y = 1 \wedge R(x)) \vee (y = 0 \wedge \neg R(x))\}.$$

$\square$

Intuitively, the characteristic function problem f_R of R is the 'search version' of the characteristic function, since $f_R(x) = \{1\}$ if $R(x)$ holds, and $f_R(x) = \{0\}$ otherwise. For example, f_{SAT} is the characteristic function problem of the NP-complete predicate SAT.

The following are examples of natural search problems. Let ϕ be a propositional formula, ψ a Horn formula, τ a truth assignment, G an undirected graph, and C a set of vertices of a graph G. W denotes a weighted graph, and T is a sequence of vertices of W. Under a suitable encoding scheme, these objects can be encoded as natural numbers.

$$\begin{aligned}
SAT\text{-}Search(\phi) &= \{\tau : \tau \text{ satisfies } \phi\} \\
HORNSAT\text{-}Search(\psi) &= \{\tau : \tau \text{ satisfies } \psi\} \\
MAX\text{-}CLIQUE(G) &= \{C : C \text{ is a maximum clique of } G\} \\
MAX\text{-}CLIQUE\text{-}SIZE(G) &= \{n \in \mathbb{N} : n \text{ is the size of the largest clique of } G\} \\
TSP(W) &= \{T : T \text{ is an optimal tour of } W\} \\
TSP\text{-}COST(W) &= \{n \in \mathbb{N} : n \text{ is the cost of the optimal tour of } W\}
\end{aligned}$$

TSP stands for the Traveling Salesperson Problem.

Note that SAT-*Search* and $HORNSAT$-*Search* are the only total problems among the above, and MAX-$CLIQUE$-$SIZE$ and TSP-$COST$ are the only function problems.

Let Q_R be a search problem defined by a predicate R. The *verification problem* for Q_R is the task of deciding whether $b \in Q(a)$ for arbitrary $a, b \in \mathbb{N}$, which can be done by simply evaluating $R(a, b)$. Hence, the complexity of the verification problems for Q is same as the complexity of a defining predicate R.

For example, the verification complexity of *SAT-Search* and *HORNSAT-Search* is **P** while the verification problems for *MAX-CLIQUE* and *TSP* are **coNP**. The verification problem of *MAX-CLIQUE-SIZE* and *TSP-COST* seem harder since they are complete for the class **DP**, which contains $\mathbf{NP} \cup \mathbf{coNP}$ and is contained in $\mathbf{\Sigma}_2^p \cap \mathbf{\Pi}_2^p$. (See [PY84, Pap94a] for more information on the class **DP**).

Note that the verification problem for f_{SAT} can be solved in polynomial time by making one query to *SAT*. Thus, its verification problem is in $\mathbf{P}^{\mathbf{NP}} = \mathbf{\Delta}_2^p$. In general, the verification problem of f_R is in $\mathbf{P}^R$.

The remarks above indicate that search problems can be classified, according to their verification complexity, into the following classes.

Definition 2.1.3 **VP** *is the class of* total *search problems whose verification problems are in* **P**. **VP** *stands for 'Verifiable in Polynomial-time'.*

For $i \geq 0$, $\mathbf{V\Sigma}_i^p$, $\mathbf{V\Pi}_i^p$, *and* $\mathbf{V\Delta}_i^p$ *are the classes of* total *search problems whose verification problems are in* $\mathbf{\Sigma}_i^p$, $\mathbf{\Pi}_i^p$, *and* Δ_i^p, *respectively.*

Finally, we define the verifiability hierarchy ***VPH*** *as* $\mathbf{VPH} = \bigcup_{i \geq 0} \mathbf{V\Sigma}_i^p$. □

Note that **VP** is the class of search problems Q_R such that $R \in \mathbf{P}$ and $(\forall x)(\exists y)R(x,y)$ holds; similarly for $\mathbf{V\Sigma}_i^p$, $\mathbf{V\Pi}_i^p$, and $\mathbf{V\Delta}_i^p$. Hence, *MAX-CLIQUE* is in $\mathbf{V\Pi}_1^p$ and *MAX-CLIQUE-SIZE* is in $\mathbf{V\Delta}_2^p$. However, *SAT-Search* and *HORNSAT-Search* do not belong to any of the classes just defined since they are not total.

The problem f_{SAT} is in $\mathbf{V\Delta}_2^p$. The following is an easy lemma that generalizes this fact.

Lemma 2.1.4 *Let* $i \geq 0$. *If a predicate* R *is in* $\mathbf{\Sigma}_i^p$, *then* $f_R \in \mathbf{V\Delta}_{i+1}^p$.

The following lemma connects the polynomial hierarchy and the verifiability hierarchy.

Lemma 2.1.5 *For every* $i \geq 0$, $\mathbf{V\Sigma}_i^p = \mathbf{V\Sigma}_{i+1}^p$ *if and only if* **PH** *collapses to the ith level.*

Proof. The 'if' part is obvious.

Before showing the 'only if' part, let us introduce a predicate $QSAT_i$ for $i \geq 1$. Each instance x of $QSAT_i$ encodes a propositional formula ϕ with its variables partitioned into i sets $X_1, X_2, \ldots, X_i$. We write $\phi(X_1, X_2, \ldots X_i)$ to indicate the formula and the partition. $QSAT_i(x)$ is true if and only if

$$\exists X_1 \forall X_2 \exists X_3 \ldots Q X_i \phi(X_1, X_2, \ldots, X_i)$$

is true, where Q is $\exists$ if i is odd and $\forall$ if i is even. Note that $QSAT_1$ is equivalent to SAT, which is complete for **NP**; in fact, for all $i \geq 1$, $QSAT_i$ is complete for $\mathbf{\Sigma}_i^p$ [Wra77, Pap94a]. $QSAT_i$ stands for 'Quantified Satisfiability with i alternations of quantifiers'.

Let T_{i+1} be a search problem such that

$$\begin{aligned} T_{i+1}(x) &= \{y : [y = 0] \vee [y = 1 \wedge x \in QSAT_{i+1}]\} \\ &= \begin{cases} \{0,1\} & \text{if } \exists X_1 \forall X_2 \exists X_3 \ldots Q X_{i+1} \phi(X_1, \ldots, X_{i+1}), \\ \{0\} & \text{otherwise.} \end{cases} \end{aligned}$$

It is clear that $T_{i+1} \in \mathbf{V\Sigma}_{i+1}^p$. Assume $\mathbf{V\Sigma}_i^p = \mathbf{V\Sigma}_{i+1}^p$. Then, $T_{i+1} \in \mathbf{V\Sigma}_i^p$, and it follows that $QSAT_{i+1} \in \mathbf{\Sigma}_i^p$, since $x \in QSAT_{i+1} \iff 1 \in T_{i+1}(x)$. The collapse of the polynomial hierarchy follows from the fact that $QSAT_{i+1}$ is complete for $\mathbf{\Sigma}_{i+1}^p$. $\square$

Note that **VP** coincides with the class **TFNP** introduced by Megiddo and Papadimitriou [MP91]. Several important subclasses of **VP** are identified and studied in [JPY88, Pap94b, BCE$^+$98]. We will introduce these classes later.

2.2 Witness Oracles and Computational Complexity

In the preceding section we introduced the notion of *verification complexity* of search problems. We present in this section the notion of *computational complexity* of search problems, or the complexity of 'solving' search problems. In order to do this, we need to specify a model of computation and formalize the notion of 'solving a search problem'. After making necessary definitions, we will show how verification complexity and computational complexity are related to each other.

We introduce the *Turing machines with witness oracle* of [BKT93], which is an extension of the oracle Turing machines.

Definition 2.2.1 *[BKT93] A* Turing machine M with witness oracle A, *also called a* witness-oracle Turing machine M with access to A, *is a deterministic machine with a special query tape and a write-only output tape with the following properties.*

1. *On every input, M writes a binary number in its output tape before halting.*
2. *Oracle A is in $\mathbf{\Sigma}_i^p$ for some $i \geq 1$ and of the form $A(a) \equiv (\exists x)B(a,x)$, where $B \in \mathbf{\Pi}_{i-1}^p$. Note that there exists a constant c such that if $B(a,x)$, then $|x| \leq |a|^c + c$.*
3. *When M asks a* witness query *of the form '$A(a)$?' with a binary number a, the oracle returns a pair (q,w) with $q \in \{0,1\}$ and $w \in \mathbb{N}^{\leq |a|^c+c}$ such that*

   (1) $q = 1$ and $B(a,w)$ if $A(a)$ is true; and

   (2) $q = 0$ and $w = 0$ if $A(a)$ is false.

Let $Output_M(x)$ denote the set of all possible outputs that can be produced by M on x.

A nondeterministic Turing machine with witness oracle A, *also called a* nondeterministic witness-oracle Turing machine with access to A, *is defined similarly.* □

Although witness-oracle Turing machines M are deterministic, nondeterministic behaviour may arise when a query '$(\exists w)B(a,w)$?' has multiple witnesses: the oracle may return any $(1, w^*)$ satisfying $B(a, w^*)$ and the computation of M branches off according to the witness it receives. Since M's next query may depend on w^*, different queries may be asked in different computation paths of M. Thus, valid computations of M on input x can contain different sequences of witness queries and answers. For this reason, witness-oracle machines can be too general for our analysis to work. The following, more restricted model turns out to be useful.

Definition 2.2.2 *Let M be a witness-oracle machine. M is* oblivious *if its ith query depends only on the input x and the sequence $(q_1, q_2, \ldots, q_{i-1})$, where q_j is part of the oracle answer (q_j, w_j) to M's jth query.*

We work with the oblivious machines. Note that an oblivious witness-oracle machine M can still exhibit nondeterminism, but it asks the same sequence of witness queries in every valid computation. This is because every valid sequence $(q_1, w_1), (q_2, w_2), \ldots, (q_i, w_i)$ of answers to M's queries must have the same sequence $(q_1, q_2, \ldots, q_i)$ of yes/no answers.

The following is our first attempt to formalize the notion of when a search problem should be considered solved. This is the notion implicitly used in [BKT93, Kra95].

Definition 2.2.3 *Let Q be a total search problem and M be a Turing machine with witness oracle. We say M* simply solves *Q if, for all x, M on x outputs* some *$y \in Q(x)$ in every valid computation, that is, $Output_M(x) \subseteq Q(x)$.* $\square$

The above definition has intuitive value: when a machine M always outputs a correct solution for Q and never fails, it seems reasonable to say M solves Q. However, since Q may contain solutions which M can never find, the connection between M and Q is not very tight: M solves any Q with $Output_Q(x) \subseteq Q(x)$, and there are infinitely many such Q's.

Furthermore, consider the following example.

Example 2.2.4 *Let U be a search problem defined as $U(x) = \{y : y = 0 \vee [|y| \leq |x| \wedge K(x)]\}$, where K is an undecidable predicate. Even though the verification problem* *for U is undecidable, U can be solved by a trivial machine that outputs* 0 *on every input, and therefore U can be simply solved in constant time.*

The above example shows that the simple solvability of a search problem is not meaningfully related to the verification complexity of the problem.

In order to address the issues raised in the preceding paragraph, we present a stronger notion of solvability of search problems.

Definition 2.2.5 *Let Q be a total search problem and M be a witness-oracle Turing machine. We say that M* exactly solves *Q if $Output_M(x) = Q(x)$ for all x.* □

If a witness-oracle machine M exactly solves Q, then $Q(x)$ can be described as

$$Q(x) = \{y : \text{there exists a valid computation of } M \text{ on } x \text{ with } y \text{ as the output}\}.$$

Moreover, exact solvability implies a stronger connection between machines and search problems in the sense that for every machine M, there exists a unique search problem that is exactly solved by M.

Now we introduce classes of search problems based on their computational complexity.

Definition 2.2.6 **EP** *is the class of total search problems that are* exactly solvable *by a deterministic polynomial time Turing machine.* **EP** *stands for 'Exactly solvable in Polynomial-time'.* **SP** *is defined similarly except that the machines are required to simply solve the problems instead of exactly solving them.* **SP** *stands for 'Solvable in Polynomial-time'.* □

Example 2.2.4 shows that $\mathbf{SP} \not\subseteq \mathbf{VPH}$.

The class **FP** of [JPY88, MP91, Pap94b, BCE$^+$98] can be defined as the subclass of **VP** containing search problems for which a solution can be found in deterministic polynomial-time. Therefore, $\mathbf{FP} = \mathbf{VP} \cap \mathbf{SP}$.

Note that **EP** is a class of function problems, since the output of a deterministic machine is unique. **EP** and $\square_1^p$ coincide in the following sense: $f \in \square_1^p$ if and only if $Q(x) = \{f(x)\} \in \mathbf{EP}$.

EP is extended in the following way.

Definition 2.2.7 *For $i \geq 1$, $\mathbf{EP}^{\mathbf{\Sigma}_i^p}$ is the class of total search problems that can be exactly solved by a polynomial-time oracle Turing machine with access to a $\mathbf{\Sigma}_i^p$ predicate. $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[wit]$ is defined by polynomial-time witness-oracle machines with access to $\mathbf{\Sigma}_i^p$ predicates, and $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[oblivious]$ is defined similarly by oblivious witness-oracle machines. $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[wit, f(n)]$ and $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[wit, O(f(n))]$ are obtained by bounding the number of witness queries by $f(n)$ and $O(f(n))$, respectively, where n is the input length.*

Extensions of **SP** *are defined in a similar way.* □

It should be remarked that Krentel's classes $\mathbf{FP}^{\mathbf{NP}}$ and $\mathbf{FP}^{\mathbf{NP}}[O(\log n)]$ [Kre88, Pap94a] are equivalent to $\mathbf{EP}^{\mathbf{NP}}$ and $\mathbf{EP}^{\mathbf{NP}}[O(\log n)]$, respectively. Furthermore, Krajicek's classes $\mathbf{FP}^{\mathbf{NP}}[wit]$ and $\mathbf{FP}^{\mathbf{NP}}[O(\log n)]$ [Kra95] correspond to $\mathbf{SP}^{\mathbf{NP}}[wit]$ and $\mathbf{SP}^{\mathbf{NP}}[O(\log n)]$, respectively. Notice how the term **FP** is used differently by different researchers. In our framework, the subtle differences between Krentel's classes and Krajicek's classes are made explicit.

The following is a restatement of Krentel's result in our framework.

Lemma 2.2.8 *[Kre88, Pap94a]*

(1) MAX-CLIQUE-SIZE $\in \mathbf{EP}^{\mathbf{NP}}[O(\log n)]$, *and*

(2) TSP-COST $\in \mathbf{EP}^{\mathbf{NP}}$.

Based on Krentel's result above, we can show that two of our classes defined by oblivious queries contain natural combinatorial problems.

Lemma 2.2.9

(3) MAX-CLIQUE $\in \mathbf{EP}^{\mathbf{NP}}[oblivious, O(log\ n)]$

(4) TSP $\in \mathbf{EP}^{\mathbf{NP}}[oblivious]$

Proof of Lemmas 2.2.8 and 2.2.9. Suppose we are given a graph G and asked to compute the maximum size of cliques of G. Since the size of G's largest clique is at most $|V| \leq n$, it can be computed by binary search using $O(\log n)$ queries to an NP predicate

$$R(m) \quad \equiv \quad \text{“ } G \text{ has a clique of size } \geq m\text{”}.$$

Thus, claim (1) holds.

Claim (3) follows from the above argument. Let M be a witness-oracle Turing machine that first executes the above algorithm to compute the largest clique size m^*. Then M makes a witness query '$R(m^*)$?' for which the oracle returns $(1, C)$, where C is a clique of the maximum size m^*. Then, M is an oblivious witness-oracle machine that exactly solves *MAX-CLIQUE*.

Claim (2) is shown similarly to (1). Let x describe a weighted graph G with weights represented in binary. Then the cost of a tour is at most 2^n, where $n = |x|$. Hence, the cost of an optimal TSP tour can be computed by binary search using $O(\log 2^n) = O(n)$ NP queries, and (4) easily follows. □

The following lemma shows one aspect of how the computational complexity and verification complexity are related.

Lemma 2.2.10 *For any $i \geq 1$, $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[wit] \subseteq \mathbf{V\Sigma}_{i+1}^p \subseteq \mathbf{EP}^{\mathbf{\Sigma}_{i+1}^p}[wit, 1]$.*

Proof. The second inclusion is obvious, since if $Q \in \mathbf{V\Sigma}_{i+1}^p$, then $(\exists y) y \in Q(x)$ is a $\mathbf{\Sigma}_{i+1}^p$ predicate, which can be used as a witness oracle.

The first inclusion requires more work. If $Q \in \mathbf{EP}^{\mathbf{\Sigma}_i^p}[wit]$, then there exists a witness-oracle machine M with access to an $\mathbf{\Sigma}_i^p$ predicate $R(a) \equiv (\exists z) B(a, z)$, $B \in \mathbf{\Pi}_{i-1}^p$, such that M exactly solves Q. We construct a nondeterministic oracle machine N with access to a $\mathbf{\Sigma}_i^p$ predicate such that N on (x, y) accepts if and only if $y \in Q(x)$.

Suppose that M on x asks at most $p(|x|)$ witness queries. Given (x, y), $|x| = n$, N rejects if y is too long to be in $Q(x)$. Otherwise, N nondeterministically guesses a sequence $(q_1, w_1), (q_2, w_2), \dots (q_{p(n)}, w_{p(n)})$ of oracle answers. Then N starts simulating M on x using the guessed sequence in the following way. For $i = 1, 2, \dots, p(n)$, when M asks the ith witness query a_i, N suspends the simulation and test whether the following condition is satisfied:

- $q_i = 1$ and $B(a_i, w_i)$ holds, or
- $q_1 = 0$, $w_i = 0$ and $\neg(\exists w)B(a_i, w)$ holds.

Note that this can be done by making one query to a predicate complete for $\mathbf{\Sigma}_i^p$. If the test fails, then N rejects because the guessed pair (q_i, w_i) is incorrect. Note, however, that there is one successful guess for every correct oracle answer for the ith query. Thus, the number of computations of N that successfully complete the simulation is equal to the number of valid computations of M on x. Finally, N accepts y if it is the output of M and rejects otherwise.

It is not hard to see that N accepts (x, y) if and only if $y \in Q(x)$, and that N runs in time polynomial in $|(x, y)|$. Therefore, the verification problem of Q is in $\mathbf{NP}^{\mathbf{\Sigma}_i^p} = \mathbf{\Sigma}_{i+1}^p$, and the claim holds. $\square$

The following statement appears in [BKT93, Kra95].

Lemma 2.2.11 *[BKT93, Kra95] For any $i \geq 1$, if $Q \in \mathbf{SP}^{\mathbf{\Sigma}_i^p}[wit, f(n)]$, then Q is simply solvable by a machine M with access to a predicate in $\mathbf{\Sigma}_i^p$ that asks $f(n)$ oracle queries (not witness queries) followed by one witness query.*

Proof. The proof appears in [Kra95] (see Lemma 6.3.4). It is similar to the proof of Lemma 2.2.13 below. $\square$

Inspired by the machine M in the above lemma, we make the following definition. We call M above a *patient* witness-oracle machine.

Definition 2.2.12 *A* patient *witness-oracle machine M is a special kind of witness-oracle machine which is allowed to ask only one witness query preceded by a sequence of oracle queries (not witness queries).*

For $i \geq 1$, let $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[patient, f(n)]$ be the class of search problems exactly solvable by a polynomial-time patient witness-oracle Turing machine M with access to a predicate $A \in \mathbf{\Sigma}_i^p$. Note that M is allowed to ask $f(n) - 1$ oracle queries (not witness queries) followed by one witness query.

Define

$$\mathbf{EP}^{\mathbf{\Sigma}_i^p}[patient, O(f(n))] = \bigcup_{g(n) \in O(f(n))} \mathbf{EP}^{\mathbf{\Sigma}_i^p}[patient, g(n)].$$

$\mathbf{SP}^{\mathbf{\Sigma}_i^p}[patient, f(n)]$ is defined similarly by replacing 'exactly solvable' with 'simply solvable', and $\mathbf{SP}^{\mathbf{\Sigma}_i^p}[patient, O(f(n))]$ is defined in an analogous way. □

A statement similar to Lemma 2.2.11 holds with respect to **EP** and oblivious machines.

Lemma 2.2.13 *For any $i \geq 1$, $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[oblivious, f(n)] \subseteq \mathbf{EP}^{\mathbf{\Sigma}_i^p}[patient, f(n) + 1]$*

Proof. This is proven similarly to Lemma 2.2.11 as follows. Let $Q \in \mathbf{EP}^{\mathbf{\Sigma}_i^p}[oblivious, f(n)]$ and assume that M is an oblivious witness-oracle machine that exactly solves Q. Fix x and let $n = |x|$. Then, a patient machine M' can compute the sequence $q_1, q_2, \ldots, q_{f(n)}$ of correct yes/no answers to M's witness queries by making $f(n)$ oracle queries of the form

- Is there a valid computation of M on x such that $q_1, \ldots, q_k$ is a correct sequence of yes/no answers to the first k witness queries and $q_{k+1} = 1$?

Then, M' asks a witness query

- Is there a valid computation of M on x such that $q_1, \ldots, q_{f(n)}$ is a correct sequence of yes/no answers to the queries made by M?

Note that every valid computation of M on x has the same sequence $q_1, q_2, \ldots, q_{f(n)}$ of yes/no answers. Hence, the final witness query of M' returns every valid computation of M on x, from which M' can easily extract the output of M on x. Therefore, $Output_M(x) = Output_{M'}(x)$, and M' exactly solves Q. $\square$

Corollary 2.2.14 *For any $i \geq 1$,*
(1) $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[oblivious, O(f(n))] = \mathbf{EP}^{\mathbf{\Sigma}_i^p}[patient, O(f(n))]$, and
(2) $\mathbf{SP}^{\mathbf{\Sigma}_i^p}[wit, O(f(n))] = \mathbf{SP}^{\mathbf{\Sigma}_i^p}[oblivious, O(f(n))] = \mathbf{SP}^{\mathbf{\Sigma}_i^p}[patient, O(f(n))]$.

Proof. We show (1). That $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[patient, O(f(n))] \subseteq \mathbf{EP}^{\mathbf{\Sigma}_i^p}[oblivious, O(f(n))]$ is trivial, since every patient witness-oracle machine is oblivious. The other inclusion follows from Lemma 2.2.13.

For (2), first note that

$$\mathbf{SP}^{\mathbf{\Sigma}_i^p}[patient, O(f(n))] \subseteq \mathbf{SP}^{\mathbf{\Sigma}_i^p}[oblivious, O(f(n))] \subseteq \mathbf{SP}^{\mathbf{\Sigma}_i^p}[wit, O(f(n))]$$

is immediate. The equalities follow from Lemma 2.2.11 which implies

$$\mathbf{SP}^{\mathbf{\Sigma}_i^p}[wit, O(f(n))] \subseteq \mathbf{SP}^{\mathbf{\Sigma}_i^p}[patient, O(f(n))].$$

$\square$

By Corollary 2.2.14, the oblivious witness-oracle machines and the patient witness-oracle machines are equally powerful. Interestingly, they are as powerful as the ordinary witness-oracle machines with respect to simple solvability, but we do not know whether a similar statement holds for exact solvability.

The equality (2) of Corollary 2.2.14 seems to be the reason Buss, Krajicek, and Takeuti [BKT93], and Krajicek [Kra95] study search problems in the context of simple solvability. However, the combination of exact solvability and oblivious witness-oracle machines turns out to be a natural context for the study of search problems.

2.3 Reducibility among Search Problems

Reducibility between problems is an essential tool in the field of Computational Complexity, and we need a reasonable notion of reducibility between search problems. In this section, we present two such notions. The first is many-one reducibility, which is commonly used in various papers such as [BCE$^+$98, JPY88, MP91, Pap94b]. We show that many-one reducibility corresponds in a certain sense to simple solvability, which is not a desirable property. We will then introduce a stronger notion of *exact reducibility*, which corresponds to exact solvability.

Definition 2.3.1 *Let Q_1 and Q_2 be total search problems. Then Q_1 is* many-one reducible to Q_2*, denoted $Q_1 \leq_m Q_2$, if there exist two polynomial-time functions f, g such that*

$$y \in Q_1(x) \textit{ if } (\exists z)[z \in Q_2(f(x)) \wedge y = g(x, z)].$$

$\square$

Many-one reducibility is closely connected to the notion of simple solvability in the following sense. If Q_1 and Q_2 are two total search problems and if $Q_1 \leq_m Q_2$, then there exists a witness-oracle Turing machine M with access to a predicate '$(\exists y) y \in Q_2(a)$' that simply solves Q_1 in polynomial-time as follows. Given x, M first makes a witness query '$(\exists y) Q_2(f(x))$?', which must have at least one witness. The oracle returns $(1, y^*)$ with some $y^* \in Q_2(f(x))$, and M outputs $g(x, y)$, which is guaranteed to be in $Q_1(x)$.

Let us generalize the notion of witness oracle.

Definition 2.3.2 *Let Q be a search problem. Then a Turing machine M with witness oracle Q is a machine that uses $(\exists y) y \in Q(a)$ as a witness oracle.*

The paragraph preceding the above definition shows that, if $Q_1 \leq_m Q_2$, then there exits a witness-oracle Turing machine M that simply solves Q_1 by making one witness query to Q_2.

Since we would like a framework based on exact solvability rather than simple solvability, we define a new notion of reducibility, *exact many-one reducibility.*

Definition 2.3.3 *Let Q_1 and Q_2 be total search problems. Then Q_1 is* exactly many-one reducible to Q_2*, denoted $Q_1 \leq_{em} Q_2$, if there exist two polynomial-time functions f, g such that*

$$y \in Q_1(x) \iff (\exists z)[z \in Q_2(f(x)) \wedge y = g(x, z)].$$

$\square$

Note that $\leq_{em}$ is transitive. We write *exact reducibility* when 'exact many-one reducibility' is intended. It is easy to see that, if Q_1 and Q_2 are total search problems and $Q_1 \leq_{em} Q_2$, there is a polynomial-time witness-oracle Turing machine M that *exactly solves* Q by making one witness query to Q_2. Thus, $\leq_{em}$ is related to the notion of exact solvability, as opposed to simple solvability.

Let $\mathbf{S}$ be a class of search problems. We say S is *closed under many-one reducibility (exact reducibility)* if $R \in \mathbf{S}$ and $Q_1 \leq_m Q_2$ ($Q_1 \leq_{em} Q_2$) implies $Q_1 \in \mathbf{S}$. Moreover, we define $C[\mathbf{S}]$ as the smallest class containing $\mathbf{S}$ and closed under many-one reducibility, i.e.,

$$C[\mathbf{S}] \;=\; \{Q : Q \leq_m Q' \text{ for some } Q' \in \mathbf{S}\}.$$

Similarly, $E[\mathbf{S}]$ is defined as

$$E[\mathbf{S}] \;=\; \{Q : Q \leq_{em} Q' \text{ for some } Q' \in \mathbf{S}\}.$$

We say Q_1 is *Turing reducible* to Q_2, and write $Q_1 \leq_T Q_2$, if there exists a polynomial-time Turing machine M that simply solves Q_1 by making multiple witness queries to Q_2. If Q_1 is exactly solvable by such M, then we say Q_1 is *exactly Turing reducible to* Q_2 and write $Q_1 \leq_{eT} Q_2$. Note that $Q_1 \leq_{em} Q_2$ implies $Q_1 \leq_m Q_2$, which implies $Q_1 \leq_T Q_2$.

Let $\mathbf{S}$ be a class of search problems. Then $C_T[\mathbf{S}]$ and $E_T[\mathbf{S}]$ are the smallest classes containing $\mathbf{S}$ and closed under $\leq_T$ and $\leq_{eT}$, respectively.

The following two lemmas show that notions of exact solvability and exact many-one reducibility constitute a nice framework for the study of verification complexity and computational complexity of search problems.

Lemma 2.3.4

(1) $\mathbf{EP}$ *and* $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[wit, f(n)]$ *for any* $i \geq 1$ *and any* $f(n)$ *are closed under* $\leq_{em}$.

(2) For every $i \geq 1$, $\mathbf{V\Sigma}_i^p$ *is closed under* $\leq_{em}$.

Proof. (1) We argue for the case of $\mathbf{EP}$. Let $Q_1 \leq_{em} Q_2$ and $Q_2 \in \mathbf{EP}$. By the discussion following Definition 2.3.3, there exists a polynomial-time witness oracle Turing machine M that exactly solves Q_1 by making one witness query '$(\exists y) y \in Q_2(f(x))$?'. Since $Q_2 \in \mathbf{EP}$, the query to Q_2 can be simulated in deterministic polynomial time, and therefore $Q_1 \in \mathbf{EP}$.

A similar arguments works for $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[wit, f(n)]$. The only difference is that a witness query is simulated by using $f(n)$ witness queries to a $\mathbf{\Sigma}_i^p$ predicate.

(2) Let $Q \leq_{em} Q'$ and $Q' \in \mathbf{V\Sigma}_i^p$, $i \geq 1$. Then Q can be expressed as

$$Q(x) = \{y : (\exists z) \underbrace{[y = g(x,z) \wedge z \in Q'(f(x))]}_{(*)}\},$$

where $f, g \in \square_1^p$ are functions that are asserted to exist in Definition 2.3.3. Since '$z \in Q'(f(x))$' is a $\mathbf{\Sigma}_i^p$ predicate and '$y = g(x,y)$' is polynomial-time decidable, it follows that (*) is $\mathbf{\Sigma}_i^p$ and therefore '$(\exists z)(*)$' is also $\mathbf{\Sigma}_i^p$. Thus, $Q \in \mathbf{V\Sigma}_i^p$. $\square$

However, $\mathbf{EP}$ and its extensions of Definition 2.2.7 are not closed under $\leq_m$; the search problem U in Example 2.2.4 is not in $\mathbf{EP}$, but it is many-one reducible to a trivial problem in $\mathbf{EP}$. The same argument shows that the subclasses of the verifiability hierarchy $\mathbf{VPH}$ are not closed under $\leq_m$.

The following lemma shows that the classes $\mathbf{V\Pi}_i^p$, $i \geq 0$, are not as robust as $\mathbf{V\Sigma}_i^p$. In particular, it states that $\mathbf{VP}$ is closed under $\leq_{em}$ if and only if $\mathbf{P} = \mathbf{NP}$.

Lemma 2.3.5 *For every $i \geq 0$, $\mathbf{V\Pi}_i^p$ is closed under $\leq_{em}$ if and only if the polynomial hierarchy collapses at the ith level.*

Proof. Fix $i \geq 0$. We show the 'if' direction first. Let Q_1 be a problem such that $Q_1 \leq_{em} Q_2$ for $Q_2 \in \mathbf{V\Pi}_i^p$. Then Q can be expressed as

$$Q_1(x) = \{y : (\exists z) \underbrace{[y = g(x,z) \wedge z \in Q_2(f(x))]}_{(*)}\},$$

where $f, g \in \square_1^p$ are the reduction. The $(*)$ part is $\mathbf{\Pi}_i^p$, and by Theorem 1.3.2, this defining predicate of Q_1 is $\mathbf{\Sigma}_{i+1}^p$. If the polynomial hierarchy collapses at the $i$the level (i.e. $\mathbf{PH} = \mathbf{\Sigma}_i^p$), then $\mathbf{\Pi}_i^p = \mathbf{PH}$ [SP98] and hence $Q_1 \in \mathbf{V\Pi}_i^p$.

For the 'only if' direction, the actual claim is the following: if $\mathbf{V\Pi}_i^p$ is closed under $\leq_{em}$, then $\mathbf{\Sigma}_{i+1}^p \subseteq \mathbf{\Pi}_i^p$, which implies the collapse of the polynomial hierarchy at the ith level.

Recall that $QSAT_i$ is a predicate complete for $\mathbf{\Sigma}_i^p$ [Wra77, Pap94a]; we introduced it in the proof of of Lemma 2.1.5. Let S_{i+1} be a total search problem whose instance is the same as that of $QSAT_{i+1}$, i.e., instance x encodes a propositional formula $\phi(X_1, \ldots, X_{i+1})$, where X_j's are a partition of the variables. Let λ be an arbitrary number that does not encode a partial truth assignment to X_1. The set of solutions is defined as

$$\begin{aligned} S_{i+1}(x) \quad &= \{X_1^* : \ \text{Either } \forall X_2 \exists X_3 \ldots Q X_{i+1} \phi(X_1^*, X_2, \ldots, X_{i+1}) \\ &\qquad\qquad \text{or } X_1^* = \lambda\}, \end{aligned}$$

where Q is $\exists$ if $i+1$ is odd and $\forall$ otherwise. Note that S_{i+1} is total, since the number λ is always a solution. Furthermore, $S \in \mathbf{V\Pi}_i^p$.

Let g be a mapping such that for all x,

$$g(x,z) = \begin{cases} 1 & \text{if } z \text{ encodes a truth assignement to } X_1, \\ 0 & \text{otherwise.} \end{cases}$$

Then, let T_{i+1} be a search problem such that $T_{i+1} \leq_{em} S_{i+1}$, expressed as

$$T_{i+1}(x) = \{y : (\exists z)[z \in S_{i+1}(x) \wedge y = g(x,z)\}.$$

From this definition, it follows that

$$T_{i+1}(x) = \begin{cases} \{0,1\} & \text{if } \exists X_1 \forall X_2 \exists X_3 \ldots QX_{i+1}\phi(X_1, \ldots, X_{i+1}), \\ \{0\} & \text{otherwise.} \end{cases}$$

The rest of the proof is identical to the final step of the proof of Lemma 2.1.5. If $\mathbf{V\Pi}_i^p$ is closed under $\leq_{em}$, then $T_{i+1} \in \mathbf{V\Pi}_i^p$ and thus $QSAT_{i+1} \in \mathbf{\Pi}_i^p$, because $x \in QSAT_{i+1} \iff 1 \in T_{i+1}(x)$. The collapse of the polynomial hierarchy follows from the fact that $QSAT_{i+1}$ is complete for $\mathbf{\Sigma}_{i+1}^p$ [Wra77, Pap94a]. □

Krentel [Kre88] showed that his classes $\mathbf{FP^{NP}}$ and $\mathbf{FP^{NP}}[O(\log n)]$ have natural complete problems. We can directly translate his results into our framework as follows.

Theorem 2.3.6 *[Kre88, Pap94a] With respect to* $\leq_{em}$*,*
(1) $MAX\text{-}CLIQUE\text{-}SIZE$ *is complete for* $\mathbf{EP^{NP}}[O(\log n)]$*; and*
(2) $TSP\text{-}COST$ *is complete for* $\mathbf{EP^{NP}}$*.*

Proof Idea. Krentel defines a search problem $MAX\text{-}OUTPUT$ as follows.

Input $(N, 1^n)$, where N is a nondeterministic polynomial-time Turing machine such that in every valid computation, N on 1^n halts with a binary number of length $O(n)$ as its output.

Output A maximum number that is output by N on 1^n.

He first proves that $MAX\text{-}OUTPUT$ is complete for $\mathbf{EP^{NP}}$. Then, he shows several problems are complete for the class, including $TSP\text{-}COST$.

Let $MAX\text{-}OUTPUT[O(\log n)]$ be defined similarly to $MAX\text{-}OUTPUT$ except that N outputs a binary number of length $O(\log n)$. This problem can be shown to be complete for $\mathbf{EP^{NP}}[O(\log n)]$, and it is used in the proof showing that $MAX\text{-}CLIQUE\text{-}SIZE$ is complete for the class. □

Based on the above result of Krentel, we believe the following assertions hold: with respect to $\leq_{em}$,

- $MAX\text{-}CLIQUE$ is complete for $\mathbf{EP}^{\mathbf{NP}}[oblivious, O(\log n]$; and
- TSP is complete for $\mathbf{EP}^{\mathbf{NP}}[oblivious]$.

2.4 Subclasses of VP

Johnson, Papadimitriou, and Yannakakis [JPY88] defined a *polynomial local search* (PLS) problem as an optimization problem that can be formulated as a local search problem. The following definition of PLS problems is by Buss and Krajicek [BK94, Bus98].

Definition 2.4.1 *[BK94, Bus98] Let Q be a search problem. We say Q is a PLS problem if the following conditions are met.*

1. *There exists a set $F_Q(x)$ of* candidates *and a constant c such that for every instance x, $0 \in F_Q(x)$ and $Q(x) \subseteq F_Q(x) \subseteq \mathbb{N}^{\leq |x|^c + c}$. The binary predicate '$v \in F_Q(x)$' is required to be polynomial-time.*

2. *There exist polynomial-time functions $C_Q, N_Q : \mathbb{N} \mapsto \mathbb{N}$ such that for every instance x, $N_Q(x,v) \neq v$ implies $C_Q(x,v) < C_Q(x, N_Q(x,v))$. N_Q is an abstraction of a heuristic used to improve the cost of the current candidate. When $v \in F_Q(x) \wedge N_Q(x,v) = v$, v is said to be* locally optimal.

3. *$Q(x)$ can be expressed as the set of candidates that are locally optimal, i.e.,*

$$Q(x) = \{v : v \in F_Q(x) \wedge N_Q(x,v) = v\}.$$

PLS *is the class of all search problems satisfying the above conditions.* □

Let Q be a PLS problem and x be an instance of Q. The instance x, together with F_Q, C_Q and N_Q, implicitly define an exponentially large directed graph $G = (V, E)$, where $V = F_Q(x)$ and $(u, v) \in E$ if $N(x, u) = v \wedge v \neq u$. By the definition of PLS problems, G is acyclic and $Q(x)$ is the set of sinks of G. Since every directed acyclic graph has a sink, Q is total. Sinks of G can be recognized in polynomial time, and therefore **PLS** $\subseteq$ **VP**.

PLS contains optimization problems that arise naturally in practical settings, some of which are complete for the class; see [Yan97] for more information on **PLS**.

The relation between **PLS** and the $\forall\exists\Sigma_1^b$-consequences of the theory T_2^1 of bounded arithmetic is shown in [BK94]; see also [Kra95, CK98]. We will discuss and extend this result in Chapter 4.

It is obvious that **PLS** is not closed under $\leq_m$; see Example 2.2.4. However, the question of whether it **PLS** is closed under $\leq_{em}$ is related to the question of whether every PLS problem is simply solvable in polynomial time.

Lemma 2.4.2 *If* $E[$**PLS**$] \subseteq$ **VP**, *then* **PLS** $\subseteq$ **SP**.

Proof. Let $Q \in$ **PLS**. Assume that there exists a problem $Q^* \in E[$**PLS**$]$ and

$$Q^*(x) = \{z : (\exists y)[y \in Q(x) \wedge z \text{ is a prefix of } y]\},$$

where z is said to be a prefix of y if it consists of the most significant $|z|$ bits of y. In other words, if $y \in Q(x)$, then $Q^*(x)$ contains all possible prefixes of y.

If $E[$**PLS**$] \subseteq$ **VP**, then the verification problem for Q^* is polynomial-time. If this is the case, we can construct a solution for $Q(x)$ in a bit-by-bit manner as follows. First, check if $0 \in Q^*(x)$. If the answer is yes, then there exists a solution $y \in Q(x)$ whose first bit is 0, so set $b_1 := 0$. If the answer is no, then set $b_1 := 1$. Next, check if $b_1 0 \in Q^*(x)$ and set b_2 accordingly. By repeating this polynomially many times, we can find every bit of a solution for $Q(x)$. Thus, $Q \in$ **SP** follows.

It remains to show that such $Q^* \in E[$**PLS**$]$ exists. It is done by constructing another

PLS problem Q' so that $Q^* \leq_{em} Q'$. We define Q' as

$$Q'(x) = \{y : (\exists y_1, y_2)(y = y_1 1 y_2) \wedge |y_1| \leq c\log |x| \wedge y_2 \in Q(x)\},$$

where '$y = 1y_1 1 y_2$' means that the binary string y is a concatenation of y_1, '1', and y_2, and c is a constant such that $Q(x) \subseteq \mathbb{N}^{\leq |x|^c+c}$. Intuitively, if $y \in Q(x)$ then $Q'(x)$ contains $|x|^c + c$ solutions of the form $i1y$, where $i \in \mathbb{N}^{\leq |x|^c+c}$.

It is not hard to show that Q' is also a PLS problem. Recall that F_Q, C_Q and N_Q define a dag G such that $Q(x)$ is the set of sinks of G. Then we can easily construct $F_{Q'}, C_{Q'}$ and $N_{Q'}$ that define a dag G' consisting of $|x|^c + c$ copies of G with vertices named appropriately so that $Q'(x)$ is the set of sinks of G'.

Now we can define Q^* as

$$Q^*(x) = \{z : (\exists y)[y = y_1 1 y_2 \wedge y \in Q'(x) \wedge |z| = y_1 \wedge z \text{ is a prefix of } y_2]\}.$$

Note that a prefix y_1 of $y \in Q'(x)$ indicates the length of z, a prefix of $y_2 \in Q(x)$.

From this definition, it is clear that $Q^* \leq_{em} Q'$. Moreover, $Q^*(x)$ is the set of all prefixes of every $y \in Q(x)$. □

Corollary 2.4.3 *If* **PLS** *is closed under* $\leq_{em}$*, then* **PLS** $\subseteq$ **SP**.

Proof. If **PLS** is closed under $\leq_{em}$, then $E[\mathbf{PLS}] \subseteq \mathbf{PLS} \subseteq \mathbf{VP}$. □

Let $Q \in$ **PLS**. Recall that instance x and F_Q, N_Q, and C_Q implicitly specifies an exponentially large directed acyclic graph (dag) G, and that a *PLS* problem can be thought of as the problem of finding a sink of G. The existence of a solution is guaranteed by a combinatorial principle "every dag has a sink". The classes **PPA**, **PPAD**, **PPADS**, and **PPP** are defined similarly as problems of finding in an exponentially large graph a node with a certain property [Pap90, Pap94b]. The following combinatorial principles guarantee the totality of search problems in each class: the parity principle "*every graph of degree two or less has an even number of leaves*" (**PPA**), the parity principle for directed graphs (**PPAD** and **PPADS**), and the injective pigeonhole principle "*there is no injective mapping from* $n+1$ *to* n" (**PPP**). These classes contain many natural problems of practical interest, some of which are complete. Type-2 characterization of these classes appears in [BCE$^+$98], and it is a topic of the next chapter.

Chapter 3

Type-2 Characterization

3.1 Basic Definitions

So far we have been working with type-1 predicates, or predicates whose arguments are numbers. A predicate is said to be *type-2* when its arguments are either numbers or *functions* on numbers [Tow90, CIY97, BCE$^+$98].

Let $P(\alpha, x, y)$ be a type-2 predicate that takes a function argument α and number arguments x and y. We generalize the notion of polynomially balanced predicate and say P is *polynomially balanced* if there exists a constant c such that if $P(\alpha, x, y)$ then $|y| \leq |x|^c + c$. We work only with polynomially balanced type-2 predicates.

The complexity of a type-2 predicate P is measured as the amount of resource (time or space) required to decide $P(\alpha, x, y)$ in a world in which $\alpha(c)$ for any c can be computed at unit cost. More specifically, a Turing machine M that decides $P(\alpha, x, y)$ is allowed to access α as an oracle: M on (x, y) can ask queries of the form '$\alpha(c) =?$' and receive the value of $\alpha(c)$ at unit cost. P is said to be *polynomial-time* if M runs in time polynomial in $|(x, y)|$.

Definition 3.1.1 *[BCE$^+$98] Every 3-ary, polynomially balanced type-2 predicate P defines a* type-2 search problem $Q_P(\alpha, x) = \{y : P(\alpha, x, y)\}$. *A pair (α, x) is an* instance *of Q_P, and $Q_P(\alpha, x) = \{y : P(\alpha, x, y)\}$ is the set of* solutions *for the instance. P is a* defining predicate *of Q_P.*

A type-2 search problem Q_P is said to be total *if $|Q_P(\alpha, x)| > 0$ for every (α, x), and it is a* function *if $|Q_P(x)| \leq 1$ for every (α, x).* □

The subscript of Q_P is omitted when it is not necessary to indicate a defining predicate of a search problem.

As in the type-1 case, evaluating $P(\alpha, x, y)$ for arbitrary (α, x, y) is the *verification problem* for Q_P. The *verification complexity* of Q is the complexity of its verification problem.

Definition 3.1.2 $\mathbf{V^2P}$ *is the class of type-2 total search problems whose verification problems are polynomial-time computable. For $i \geq 0$, $\mathbf{V^2\Sigma}_i^p$ and $\mathbf{V^2\Pi}_i^p$ are the classes of type-2 total search problems whose verification problems are in type-2 $\mathbf{\Sigma}_i^p$ and type-2 $\mathbf{\Pi}_i^p$, respectively.* □

See [Tow90, CIY97] for more information on type-2 predicates and the type-2 polynomial-time hierarchy.

The notion of many-one reducibility is generalized to type-2 setting as follows.

Definition 3.1.3 *[CIY97, BCE$^+$98]*

Let Q_1 and Q_2 be type-2 total search problems. We say Q_1 is many-one reducible to Q_2, *denoted $Q_1 \leq_m Q_2$, if there exist three type-2 polynomial-time functions F, G and H such that*

$$y \in Q_1(\alpha, x) \text{ if } (\exists z)[z \in Q_2(F[\alpha, x], G(\alpha, x)) \wedge y = H(\alpha, x, z)],$$

where $F[\alpha, x] = \lambda z.F(\alpha, x, z)$. □

Exact many-one reducibility between type-2 search problems is defined as follows.

Definition 3.1.4 *Let Q_1 and Q_2 be type-2 total search problems. Then Q_1 is* exactly many-one reducible to Q_2*, denoted $Q_1 \leq_{em} Q_2$, if there exist three type-2 polynomial-time functions F, G and H such that*

$$y \in Q_1(\alpha, x) \iff (\exists z)[z \in Q_2(F[\alpha, x], G(\alpha, x)) \wedge y = H(\alpha, x, z)],$$

where $F[\alpha, x] = \lambda z.F(\alpha, x, z)$. □

When Q_1 is type-1, the above definitions are applied by treating Q_1 as a type-2 problem with no function argument. More specifically, Q_1 is exactly many-one reducible to Q_2 if there are three type-1 functions f, g, and h such that

$$Q_1(x) = \{y : (\exists z)[z \in Q_2(f[x], g(x)) \wedge y = h(x, z)]\},$$

where $f[x] = \lambda z.f(x, z)$.

Let Q be a type-2 total search problem. Then

$$\begin{aligned} C(Q) &= \{R : R \text{ is type-1 and } R \leq_m Q\}, \text{and} \\ E(Q) &= \{R : R \text{ is type-1 and } R \leq_{em} Q\}. \end{aligned}$$

It is straightforward to generalize Turing reducibility to the type-2 setting. When Q_1 and Q_2 are type-2 total search problems, we say Q_1 is *Turing reducible* to Q_2, denoted $Q_1 \leq_T Q_2$, if there exists a polynomial-time Turing machine M with witness oracle Q_2 that, on (α, x), outputs some $y \in Q_1(\alpha, x)$. For each query to Q_2, M must provide an instance (β, z) of Q_2, where β is polynomial-time computable with access to α.

We say Q_1 is *exactly Turing reducible* to Q_2 if there exists a witness-oracle Turing machine M that exactly solves Q_1 using Q_2 as a witness oracle.

3.2 Type-2 Problems and Search Classes

We begin with the definition of a new type-2 problem *ITERATION* (*ITER*). It is inspired by the *iteration problems* of [CK98]. Since both **PLS** [BK94] and the iteration problems [CK98] characterize the $\forall\exists\Sigma_1^b$-consequences of T_2^1 in a certain sense, they have been known to be equivalent. However, their equivalence has been stated in an indirect way and therefore it has not been clear how problems in one class relate to the problems in the other.

We show a more precise correspondence by demonstrating $E[\mathbf{PLS}] = E(ITER)$, that is, a search problem is exactly reducible to a PLS problem if and only if it is exactly reducible to $ITER$. Moreover, our result is obtained directly without relying on the results in bounded arithmetic.

Definition 3.2.1 *ITERATION (ITER) is a type-2 problem specified as follows. Let (α, x) be an instance, where $\alpha : \mathbb{N} \mapsto \mathbb{N}$ is any function and x is a natural number. $V = \mathbb{N}^{\leq |x|}$ is the* search space. *A function $\alpha^* : V \mapsto V$ is such that*

$$\alpha^*(v) = \begin{cases} \alpha(v) & \textit{if } \alpha(v) \in V \textit{ and } \alpha(v) > v \\ v & \textit{otherwise} \end{cases}$$

for all $v \in V$. Note that $\alpha^*(v) \geq v$ for every $v \in V$. Then the set of solutions for an instance (α, x) is

$$ITER(\alpha, x) = \{v \in V : [v = 0 \wedge \alpha^*(0) = 0] \vee [v < \alpha^*(v) \wedge \alpha^*(v) = \alpha^*(\alpha^*(v))]\}.$$

$\square$

The totality of $ITER$ follows from the iteration principle which states that if f satisfies the conditions

1. $0 < f(0)$

2. $(\forall x < a)[f(x) = a \vee f(x) < f(f(x))]$, and

3. $(\forall x < a)f(x) \leq a$,

then there exists a $b < a$ such that $f(b) = a$ [BK94].

We can also interpret $ITER$ as the problem of finding vertices with a certain property in an exponentially large directed graph. Let (α, x) be an instance of $ITER$, and let $V = \mathbb{N}^{\leq |x|}$ as in the above definition. If we define E as the set of pairs $(u, v) \in V \times V$ such that $\alpha^*(u) = v$, then $G = (V, E)$ is a directed graph with no cycles of length ≥ 2. We say that v is a *loop* if $\alpha^*(v) = v$, and we say that v is the *successor* of u if $\alpha^*(u) = v$, $u \neq v$. Then $Q(x)$ is the set containing (i) 0 if it is a loop, and (ii) every v whose successor is a loop. It is not hard to see that the totality of $ITER$ follows from the combinatorial principle "every dag has a sink".

The fact that $ITER(\alpha, x)$ can be expressed as the set of sinks and possibly 0 suggests a strong connection between **PLS** and $ITER$. This connection is formalized in the following statements.

Lemma 3.2.2 **PLS** $\subseteq E(ITER)$.

Proof. Let $Q \in$ **PLS**, and let F_Q, N_Q, C_Q be the predicate and functions that specify a directed acyclic graph (dag) G such that $Q(x)$ is the set of sinks of G. Our goal is to show that $Q \leq_{em} ITER$ by constructing functions f, g, and h such that

$$y \in Q(x) \iff (\exists z)[z \in ITER(f[x], g(x)) \wedge y = h(x, z)],$$

where $f[x] = \lambda z.f(x, z)$, and $f[x], g, h \in \square_1^p$. Informally, $f[x]$ and g specify a directed graph so that h is a mapping from the set of sinks onto $Q(x)$.

The idea is to construct f, g, h so that they define a dag G' similar to G such that $Q(x)$ can be extracted from the set of sinks of G'. This seems easy, except that we have to ensure that if $(u, v) \in E(G')$ then $u \geq v$. Note that there is no such restriction in G. We construct G' in the following way so that this condition is satisfied.

Recall that there exists a constant c and a set $F_Q(x)$ such that $Q(x) \subseteq F_Q(x) \subseteq \mathbb{N}^{\leq |x|^c + c}$ and $0 \in F_Q(x)$ for every x. Since C_Q is a polynomial-time function, there is a polynomial $p(n)$ such that for all $v \in F_Q(x)$, $|C_Q(x, v)| \leq p(|x|)$. Assume without loss of generality that $C_Q(x, v) > 0$ for all $v \in F_Q(x)$.

Fix an instance x of Q, and define $U = \mathbb{N}^{\leq p(|x|)+|x|^c+c}$, the set of numbers of binary length at most $p(|x|)+|x|^c+c$. We assume that every $u \in U$ is described by $p(|x|)+|x|^c+c$ bits, possibly with leading 0's. Then, every $u \in U$ is the concatenation of u_1 and u_2, where u_1 is a $p(|x|)$-bit string and u_2 $|x|^c + c$-bit. We say u is *correct* if $u_1 = C_Q(u_2)$, that is, if the prefix of u correctly describes the cost of the candidate corresponding to u's suffix.

Define a function $f[x] : U \mapsto U$ with parameter x as

$$f[x](u) = \begin{cases} (u_1+1)u_2 & \text{if } u_2 \in F_Q(x) \wedge u_1 < C_Q(x, u_2), \\ w_1 w_2, & \text{where } w_1 \text{ is a sequence of 0's and } w_2 = N_Q(x, v), \\ & \text{if } v \in F_Q(x) \wedge c = C_Q(x, v) \wedge N_Q(x, v) \neq v, \\ u & \text{otherwise.} \end{cases}$$

Note that $f[x](u) \geq u$ for all $u \in U$.

Now define $E \subseteq U \times U$ by $(u, v) \in E$ if and only if $f[x](u) = v$. Then $G' = (U, E)$ is directed graph with no cycles of length ≥ 2. Let us say $u \in U$ is *isolated* when u is a loop and u's indegree is 1. It follows that for every $v \in F_Q(x)$, E contains a path $\langle 0v, 1v, \ldots C_Q(v)v \rangle$ and isolated vertices $(c+1)v, \ldots, (2^{p(|x|)} - 1)v$. The path may be part of a longer path.

Lemma 3.2.3 *A correct vertex $u = u_1 u_2 \in U$ is a loop that is not isolated if and only if $u_2 \in Q(x)$.*

Proof. ($\Rightarrow$) Suppose $u = u_1 u_2$ is a loop that is not isolated. Then $u_2 \in F_Q(x)$ because u is not isolated, and $N_Q(x, u_2) = u_2$ since u is a loop. By the definition of PLS problems, u_2 must be in $Q(x)$.

($\Leftarrow$) Suppose $v \in Q(x)$. Then, by the definition of PLS problems, $v \in F_Q(x)$ and $N_Q(x, v) = v$. The paragraph before the current lemma shows that G contains a path $\langle 0v, 1v, \ldots C_Q(v)v \rangle$, the last vertex of which is a loop. Then $u = u_1 u_2 = C_Q(x, v)v$. (Lemma 3.2.3) □

Now consider $ITER(f, p(|x|) + |x|^c + c)$; it is the set of vertices in G' whose successors are loops, which, by the above lemma, are the solutions for $Q(x)$. Thus,

$$Q(x) = \{y : (\exists u)[u = u_1u_2 \in ITER(f[x], g(x)) \wedge y = N_Q(x, u_2)]\},$$

where $g(x) = p(|x|) + |x|^c + c$, which is polynomial-time computable. Therefore, $Q \leq_{em}$ $ITER$. $\square$

Let Q be a type-2 search problem. Then we write $Q[\square_1^p]$ to denote the set of type-1 search problems Q' for which there exist functions $f \in \square_1^p$ such that $Q'(x) = Q(f[x], x)$, where $f[x] = \lambda z.f(x, z)$. Intuitively, $Q[\square_1^p]$ is the set of instances of Q such that the function given as an argument is polynomial-time computable.

Lemma 3.2.4 $ITER[\square_1^p] \subseteq$ **PLS**.

Proof. It suffices to show that, for every $f \in \square_1^p$, there is a PLS problem Q satisfying $Q(x) = ITER(f[x], x)$ for all x.

We construct Q by specifying F_Q, C_Q and N_Q. First, let $V = \mathbb{N}^{\leq |x|}$. We define $F_Q(x) = \{v \in V : f[x](v) > v\} \cup \{0\} \subseteq \mathbb{N}^{\leq |x|}$. It easily satisfies $Q(x) \subseteq F_Q(x)$ and $0 \in F_Q(x)$. For $v \in F_Q(x)$, the cost function is simply $C_Q(x, v) = v$, and the neighbourhood of v is

$$N_Q(x, v) = \begin{cases} f[x](v) & \text{if } f[x](v) \in V, \\ v & \text{otherwise.} \end{cases}$$

To see that $Q(x) = ITER(f[x], g(x))$, note that $v < f[x](v)$ and $f[x](v) = f[x](f[x](v))$ if and only if $v \in F_Q(x)$ and $N_Q(x, v) = v$. $\square$

The above lemmas establish a direct connection between $ITER$ and **PLS** with respect to $\leq_{em}$.

Theorem 3.2.5 $E[\mathbf{PLS}] = E(ITER)$.

Proof. Lemma 3.2.2 states that $\mathbf{PLS} \subseteq E(ITER)$, from which $E[\mathbf{PLS}] \subseteq E(ITER)$ follows. The opposite inclusion follows from Lemma 3.2.4, since $Q \in E(ITER)$ if and only if $Q \leq_{em} Q'$ for some $Q' \in ITER[\Box_1^p] \subseteq \mathbf{PLS}$. $\Box$

We introduce below the $\mathbf{V}^2\mathbf{P}$ type-2 search problems that originally defined by Beame, Cook, Edmonds, Impagliazzo, and Pitassi [BCE+98]. Each problem is defined on instances of the form (α, x), where α is any mapping from $\mathbb{N}$ to $\mathbb{N}$ and x is a string that defines the *search space* $V = \mathbb{N}^{\leq |x|} = \{n \in \mathbb{N} : |n| \leq |x|\}$. Given (α, x), $\alpha_V : V \mapsto V$ from α can be defined as $\alpha_V(v) = 0$ if $\alpha(v) \notin V$ and $\alpha_V(v) = \alpha(v)$ otherwise, for all $v \in V$.

Under a reasonable encoding scheme, α can be interpreted as a mapping from $\mathbb{N}$ to $\mathbb{N} \times \mathbb{N}$. When α is interpreted as such, α_V denotes a function $\alpha_V : V \mapsto V \times V$.

Definition 3.2.6 *[BCE+98] $LEAF(\alpha, x)$ is defined as follows. Let $V = \mathbb{N}^{\leq |x|}$, and define $\alpha_V : V \mapsto V \times V$ from α. Then every instance (α, x) defines an undirected graph $G = (V, E)$, where $\{u, v\} \in E(G)$ if and only if $u \neq v \wedge u \in \alpha_V(v) \wedge v \in \alpha_V(u)$. Finally,*

$$\begin{aligned} LEAF(\alpha, x) = \{v \in V : \ & (v = 0 \wedge v \textit{ is not a leaf of } G) \\ & \vee (v \neq 0 \wedge v \textit{ is a leaf of } G)\}, \end{aligned}$$

where a leaf of G is a vertex with degree 1. $\Box$

It is easy to see that G has maximum degree ≤ 2 and therefore G consists of isolated vertices and paths. Because of the parity principle "every graph of degree two or less has an even number of leaves", $LEAF$ is total.

Definition 3.2.7 *[BCE+98] The type-2 search problem $SOURCE.OR.SINK$ (SOS) is defined as follows. An instance (α, x), defines a graph in a way similar to $LEAF$ except that the graph $G = (V, E)$ is directed. There is a directed edge from u to v if and only if $u \neq v \wedge \alpha_V(u) = (*, v) \wedge \alpha_V(v) = (u, *)$, where $*$ denotes an arbitrary vertex. A vertex $v \in V$ is a* source *if it has indegree* 0 *and outdegree 1, and it is a* sink *if its indegree is*

0 and outdegree is 1. Then

$$SOS(\alpha,x)=\{v\in V:\ (v=0\wedge v \textit{ is not a source of } G) \vee (v\neq 0\wedge v \textit{ is a sink or source of } G)\}.$$

□

The directed graph G in the above definition has maximum outdegree and indegree ≤ 1, and hence it consists of isolated vertices and directed paths. Thus, the existence of a solution is guaranteed by the variant of the parity principle "every directed graph with indegree and outdegree ≤ 1 has a sink if it has a source".

Definition 3.2.8 *[BCE+98] The type-2 search problem $SINK$ is defined in the same way as $SOURCE.OR.SINK$ except that*

$$SINK(\alpha,x)=\{v\in V:\ (v=0\wedge v \textit{ is not a source of } G) \vee (v\neq 0\wedge v \textit{ is a sink } G)\}.$$

□

Finally, the following problem is total because of the injective version of the pigeonhole principle "there is no injective mapping from $a+1$ to a".

Definition 3.2.9 *[BCE+98] For every (α,x), $PIGEON(\alpha,x)$ is specified as follows. Let $V=\mathbb{N}^{\leq|x|}$, and define $\alpha_V : V\mapsto V$ from α. Then*

$$PIGEON(\alpha,x)=\{(v_1,v_2)\in V\times V:\alpha_V(v_1)=0\vee\alpha_V(v_1)=\alpha_V(v_2)\}.$$

□

Using the type-2 problems above of Beame et. al., we can define Papadimitriou's classes **PPA**, **PPAD**, **PPADS**, and **PPP** [Pap94b] in a clean way.

Definition 3.2.10

(1) **PPA** $= LEAF[\square_1^p]$,

(2) $\mathbf{PPAD} = SOS[\Box_1^p]$,

(3) $\mathbf{PPADS} = SINK[\Box_1^p]$, *and*

(4) $\mathbf{PPP} = PIGEON[\Box_1^p]$. $\Box$

It is not hard to prove that our definition and Papadimitriou's definition in [Pap94b] define the same classes. Our definition is inspired by the following, slightly different definitions by Beame et. al [BCE$^+$98]: we rename the classes to avoid confusion.

Definition 3.2.11 *[BCE$^+$98]*

(1) $\mathbf{PPA}^* = C(LEAF) \cap \mathbf{VP}$,

(2) $\mathbf{PPAD}^* = C(SOS) \cap \mathbf{VP}$,

(3) $\mathbf{PPADS}^* = C(SINK) \cap \mathbf{VP}$, *and*

(4) $\mathbf{PPP}^* = C(PIGEON) \cap \mathbf{VP}$. $\Box$

It should be noted that $\mathbf{PPA}^*$ is not the same as Papadimitriou's **PPA**, since it is possible to show that $\mathbf{PPA}^* = C(LEAF) \cap \mathbf{VP}$ contains problems with an even number of solutions, while every **PPA** problem must have an odd number of solutions. The same argument shows that $\mathbf{PPAD} \neq \mathbf{PPAD}^*$. We do not know whether $\mathbf{PPADS} = \mathbf{PPADS}^*$ and/or $\mathbf{PPP} = \mathbf{PPP}^*$ hold.

The starred classes of Definition 3.2.11 have an advantage of being closed under $\leq_m$ in **VP**. On the other hand, the classes of Definition 3.2.10 are not closed under $\leq_{em}$, assuming that they are not simply solvable in polynomial time. However, we feel that the classes of the form $E(Q)$, where Q is $LEAF$, SOS, $SINK$, or $PIGEON$ are more natural, not only because of the following lemma but also because of their relationship to theories of bounded arithmetic, which is discussed in the next chapter.

Lemma 3.2.12

(1) $E(LEAF) = E[\mathbf{PPA}] \subseteq \mathbf{V\Sigma}_1^p$

(2) $E(SOS) = E[\mathbf{PPAD}] \subseteq \mathbf{V\Sigma}_1^p$

(3) $E(SINK) = E[\mathbf{PPADS}] \subseteq \mathbf{V\Sigma}_1^p$

(4) $E(PIGEON) = E[\mathbf{PPP}] \subseteq \mathbf{V\Sigma}_1^p$

Proof. Immediate from Definition 3.2.10. □

Interestingly, if we define classes in the style of Beame et. al. (Definition 3.2.11, [BCE+98]) but intersect $C(Q)$ with $\mathbf{V\Sigma}_1^p$ instead of $\mathbf{VP}$, the resulting class is $E(Q)$.

Lemma 3.2.13

(1) $E(LEAF) = C(LEAF) \cap \mathbf{V\Sigma}_1^p$

(2) $E(SOS) = C(SOS) \cap \mathbf{V\Sigma}_1^p$

(3) $E(SINK) = C(SINK) \cap \mathbf{V\Sigma}_1^p$

(4) $E(PIGEON) = C(PIGEON) \cap \mathbf{V\Sigma}_1^p$

Proof. We sketch the proof for (1); the others can be shown by similar arguments. First, note that $E(LEAF) \subseteq C(LEAF) \cap \mathbf{V\Sigma}_1^p$ easily follows from $E(LEAF) \subseteq \mathbf{V\Sigma}_1^p$.

Our goal is to show $C(LEAF) \cap \mathbf{V\Sigma}_1^p \subseteq E(LEAF)$. Let $Q \in C(LEAF) \cap \mathbf{V\Sigma}_1^p$. We first describe a *nondeterministic* witness-oracle Turing machine N that exactly solves Q, and then we show that N can be simulated by a deterministic witness-oracle machine. N on x first computes a solution y for $Q(x)$ deterministically by making one witness query to $LEAF$. Then, N nondeterministically guesses a number z and an alleged witness w to the **NP** predicate '$z \in Q(x)$'. If w correctly witnesses '$z \in Q(x)$', N outputs z and halts. Otherwise, it halts with y. It is easy to check that $Output_N(x) = Q(x)$ and that N runs in polynomial time.

Next, we show that N can be simulated by a deterministic witness-oracle machine M that makes one witness query to $LEAF$. Since N does not exhibit nondeterminism until it queries $LEAF$, it suffices to show that, by asking a witness query to $LEAF$, M can obtain a nondeterministically chosen number in addition to the witnesses N would receive form $LEAF$.

Suppose that the simulation of N by M requires a nondeterministically chosen c-bit number. We claim that, for every functions f and g, there exist f_c and g_c such that for any $0 \leq s \leq 2^c$,

$$y \in LEAF(f[x], g(x)) \iff (y2^c) + s \in LEAF(f_c[x], g_c(x)).$$

Note that every solution for $LEAF(f_c[x], g_c(x))$ is the concatenation of y and s such that $y \in Q(x)$ and s is nondeterministically chosen c-bit number. Now our goal is to show such f_c and g_c exist.

Recall that $LEAF(f[x], g(x))$ is the set of leaves of the graph $G = (V, E)$, where $V = \mathbb{N}^{\leq |g(x)|}$ such that E is specified by $f[x]$. Consider a graph $G' = (V'E')$ consisting 2^c copies $G_0, G_1, \ldots, G_{2^c-1}$ of G, so that the vertex of G_i corresponding to $j \in V$ is named $i2^c + j$. Note that $i2^c + j$ is a leaf of V if and only if j is a leaf of G. However, we do not want $i2^c$ to be a leaf when 0 is not a solution for $LEAF(f[x], g(x))$. We ensure that $i2^c$, $1 \geq i \geq 2^c - 1$, is not a leaf by modifying G' in the following way: first, modify G_{2^c-1} so that it consists of one large cycle; then, create an edge between vertices $i2^c$ and $(i+1)2^c$, for $i = 1, 3, 5, \ldots, 2^c - 3$. Let $f_c[x], g_c(x)$ be functions that implicitly specify the resulting G': we can easily construct them from $f[x]$ and $g(x)$. □

The following can be shown similarly to Lemma 2.4.2, which states that if $E[\mathbf{PLS}] \subseteq \mathbf{VP}$, then $\mathbf{PLS} \subseteq \mathbf{SP}$.

Lemma 3.2.14 *If* $E[\mathbf{PPA}] \subseteq \mathbf{VP}$*, then* $\mathbf{PPA} \subseteq \mathbf{SP}$*. Similarly for* **PPAD***,* **PPADS***, and* **PPP**.

3.3 Reducibility among the Type-2 Search Problems

Beame et. al. obtained all possible reducibility and nonreducibility results among their type-2 search problems.

Theorem 3.3.1 *[BCE+98]*

(1) $SOS \leq_{em} SINK \leq_{em} PIGEON$, and $SOS \leq_{em} LEAF$.

(2) LEAF is not Turing reducible to PIGEON.

(3) PIGEON is not Turing reducible to SINK.

(4) SINK is not Turing reducible to LEAF.

Remark. $SOS \leq_{em} SINK$ in (1) is not obvious, and it is not shown in [BCE+98]. We briefly describe how $SOS \leq_{em} SINK$ follows from $SOS \leq_m SINK$ which is proven in [BCE+98]. We use the technique we used to prove Lemma 3.2.13. First, show that there exists a *nondeterministic* witness-oracle machine N with access to α that exactly solves SOS by making one witness query to $SINK$. Then, it is possible to show that N can be simulated by a *deterministic* witness-oracle machine M with access to α and $SINK$. □

Let us define the type-2 analog of $E(Q)$ as follows.

Definition 3.3.2 *Let Q be a type-2 total search problem. Then, $E^2(Q)$ is the set of* type-2 *problems that are exactly many-one reducible to Q, that is,*

$$E^2(Q) = \{R : R \textit{ is type-2 and } R \leq_{em} Q\}.$$

Then, Theorem 3.3.1 yields the following.

Corollary 3.3.3 *[BCE+98]*

(1) $E^2(SOS) \subsetneq E^2(SINK) \subsetneq E^2(PIGEON)$,

(2) $E^2(SOS) \subsetneq E^2(LEAF)$,

(3) $E^2(PIGEON) \not\subseteq E^2(LEAF)$ and $E^2(LEAF) \not\subseteq E^2(PIGEON)$.

Proof. Note that $Q_1 \leq_{em} Q_2$ implies $Q_1 \leq_T Q_2$. Then, the claims easily follow from Theorem 3.3.1. □

Now we present our main theorem. Using the techniques applied in [BCE+98], we show the following new nonreducibility result for $ITER$.

Theorem 3.3.4 *SOS is not Turing reducible to $ITER$.*

Corollary 3.3.5 *$SINK$, $PIGEON$, and $LEAF$ are not Turing reducible to $ITER$.*

Proof of Corollary 3.3.5. By Theorem 3.3.1, SOS is exactly many-one reducible to $SINK$, SOS, and $PIGEON$. If any of them is Turing reducible to $ITER$, then it would follow that $SOS \leq_T ITER$, which contradicts Theorem 3.3.4. □

Corollary 3.3.6 *Let Q be any of $LEAF$, $PIGEON$, $SINK$, and SOS. Then, $E^2(Q) \not\subseteq E^2(ITER)$.*

Proof of Corollary 3.3.6. This is immediate from Corollary 3.3.5. □

Because we connect $E^2[ITER]$ with a relativized theory $T_2^1(\alpha)$ of bounded arithmetic in the next chapter (Corollary 4.3.6), Theorem 3.3.4 yields interesting consequences on the provability of some combinatorial principles in $T_2^1(\alpha)$ (Corollary 4.3.7). This is a topic of Chapter 4.

It is worth remarking that we do not know whether $ITER$ is reducible to any of $LEAF$, SOS, $SINK$, and $PIGEON$.

The rest of this section is devoted to the proof of Theorem 3.3.4.

Proof of Theorem 3.3.4. Our proof is a straightforward application of the techniques used in [BCE+98] to prove Theorem 3.3.1. Assume, for the sake of contradiction, that $SOS \leq_T ITER$. Then, there exists a polynomial-time witness-oracle Turing machine M

that, on input (α, x), finds a solution to $SOS(\alpha, x)$ by making queries to α and $ITER$. We show that there exists (α, x) such that M's output is incorrect.

Fix an instance x with $|x|$ sufficiently large, and let $\alpha : \mathbb{N} \mapsto \mathbb{N}$. We interpret α as a function from V to $V \times V$, where $V = \mathbb{N}^{\leq |x|}$. For all $v \in V$, $\alpha(v)$ is a pair $(u, v) \in V \times V$ consisting of the *predecessor* u and the *successor* w of v; we write $u = pred(v)$ and $w = succ(v)$ to mean that u and w are the first element and the second element of $\alpha(v)$, respectively. Then a directed graph $G = (V, E)$ is defined by letting $(u, v) \in E$ if and only if $u = pred(v) \wedge v = succ(v)$. Note that G has indegree and outdegree at most 1. Recall that $v \in V$ is a sink if its outdegree is 0 and a source if its indegree is 0. We consider α that makes 0 a source so that $Q(x)$ is the set of sinks and sources of G other than 0.

Assume that M on (α, x) runs in time t. For $i = 0, 1, \ldots, t$, let α_i be a restriction of α such that

$$dom(\alpha_i) = \{v \in V : \alpha(v) \text{ is queried during the first } i \text{ steps of } M\text{'s computation on } (\alpha, x)\}.$$

Note that each α_{i+1} is an extension of α_i. The restriction α_i represents the part of G that is known to M at the end of the ith step.

Our goal is to show the following:

Lemma 3.3.7 *For $i = 0, 1, \ldots, t$, there exists α_i such that (i) $|dom(\alpha_i)|$ is bounded by a polynomial in $|x|$; and (ii) α_i does not specify any $v \in dom(\alpha_i)$, $v \neq 0$, to be a sink or a source.*

Suppose that the lemma holds. Since $|dom(\alpha_t)|$ is bounded by polynomial, only a small part of an exponentially large graph G is is known to M at the end of its computation. Moreover, the known part of G contains neither a source nor a sink. Thus, M is forced to output a sink or a source of G without knowing one, and we can extend α_t to α so that M's output is incorrect.

We have shown that Theorem 3.3.4 follows from Lemma 3.3.7. See below for the proof of the Lemma. □

Proof of Lemma 3.3.7 We prove the claim by induction on i. The base case is trivial. Assume α_i satisfies the conditions of the Lemma. We construct α_{i+1} that extends α_i. If M does not access α nor $ITER$ at the $i+1$st step, $\alpha_{i+1} = \alpha_i$ suffices.

There are only two cases remaining. Define

$$S_i = \{w : (\exists v \in dom(\alpha_i))\alpha_i(v) = (*, w)\}, and$$
$$P_i = \{u : (\exists v \in dom(\alpha_i))\alpha_i(v) = (u, *)\},$$

where $*$ denotes an arbitrary vertex.

Case: Step $i+1$ is a query v to α. Assume that $v \notin dom(\alpha_i)$; otherwise this case is trivial. If $v = 0$, then we set $\alpha(0) = (0, w)$ with arbitrary $w \notin S_i$, $w \neq 0$. Note that 0 does not have an incoming edge, since $succ(0) \neq 0$.

Otherwise, we answer the query with (u, w) as follows. If there exists $x \in dom(\alpha_i)$ with $succ(x) = v$, we set $u := x$. Note that such x is unique if it exists. Otherwise, pick an arbitrary u such that $u \notin P_i$. Similarly, if there is $y \in dom(\alpha_i)$ with $pred(y) = v$, set $w := y$; otherwise, find an arbitrary $w \notin S_i$.

It is easy to see that α_i thus constructed satisfies conditions (i) and (ii).

Case: Step $i+1$ is a query (β, z) to $ITER$. Let $V' = \mathbb{N}^{\leq |z|}$ and assume, without loss of generality, $\beta : V' \mapsto V'$ is such that $\beta(c) \geq c$ for all $c \in V'$. Then $ITER(\beta, z)$ can be expressed as

$$ITER(\beta, z) = \{c \in V' : [c = 0 \wedge \beta(0) = 0] \vee [c < \beta(c) \wedge \beta(c) = \beta(\beta(c))]\}.$$

Our goal now is to show that there exists a solution to $ITER(\beta, z)$ that can be found by making polynomially many queries to α, even if we answer the queries in such a way that no sink or source is specified by α.

Recall that β is required to be polynomial-time in α. Then, there exists a deterministic machine M^* such that M^* on c computes $\beta(c)$ in polynomial time by making queries to α. Consider the computation paths of M^* on c. The paths all start at the initial state, and they diverge only at the point when M^* asks a query '$\alpha(v) =?$' for $v \notin dom(\alpha_i)$, since there are multiple ways we can answer the query. Thus, the value of $\beta(c)$ depends only on the values $\alpha(v)$ for $v \notin dom(\alpha_i)$, and therefore we can express possible values of $\beta(c)$ by a decision tree $T(c)$ whose leaves of $T(c)$ are labeled by $\{\beta(c) = d\}$ for some $d \in V'$ and whose internal nodes represent queries '$\alpha(v) =?$' for $v \notin dom(\alpha_i)$.

To simplify our argument, we represent a query '$\alpha(v) =?$' by two successive queries '$pred(v) =?$' and '$succ(v) =?$'; note that there is no loss of generality in doing this. Then, an internal node of $T(c)$ is labeled by either $\{pred(v)\}$ or $\{succ(v)\}$ for some $v \in V$, and every edge is labeled with some $u \in V$. If an edge labeled $\{u\}$ leaves a node labeled $\{pred(v)\}$, it indicates that $pred(v) = u$. Similarly for an edge leaving a node with a label $\{succ(v)\}$. Let k be the maximum height of the trees. Note that k is bounded by a polynomial in $|x|$.

Next, we prune branches of $T(c)$ that specify a solution to $SOS(\alpha, x)$. First, we prune every edge specifying that $succ(u) = 0$ for any u. Then, we prune an edge $\{x\}$ leaving from a node $\{pred(y)\}$ if if either $x \in P_i$ or the path in $T(c)$ from the root reaching that node contains an edge specifying that $pred(w) = x$ for some w. Similarly, an edge $\{x\}$ is pruned if it originates from a node $\{succ(y)\}$ and if x has been defined as the successor of some element of V. It follows that a node at depth j of any $T(c)$ has at least $|V| - |dom(\alpha_i)| - j - 1$ outgoing edges.

Note that every path p from the root to a leaf of $T(c)$ defines a collection of values of $pred$ and $succ$ and thus values of α. We say two paths p, p' are *consistent* if they agree on the answers to the common queries. Let $label(p)$ denote the leaf label of a path p. We claim the following:

Lemma 3.3.8 *At least one of the following holds:*

(1) $T(0)$ has a path p with $label(p) = \{\beta(0) = 0\}$; or

(2) There are two consistent paths p in $T(c)$ and p' in $T(d)$, $c < d$, such that $label(p) = \{\beta(c) = d\}$ and $label(p') = \{\beta(d) = d\}$.

If item (1) holds, then we can find a solution to $ITER(\beta, z)$ by simply computing $\beta(\mathbf{0})$. Similarly, if (2) is true, computing $\beta(c)$ and $\beta(d)$ yields a solution. In either case, α_{i+1} has only polynomially more values than α_i, and the pruning procedure ensures that α_{i+1} specifies neither a sink nor a source (other than 0).

Our final goal is to show that Lemma 3.3.8 holds. We show that if (1) is false, (2) must be true. Thus, we assume that every leaf label of $T(0)$ states that $\beta(0) \neq 0$.

Let $\omega = 2^{|z|} - 1$; in other words, ω is the greatest element of V'. Then, every leaf of $T(\omega)$ is labeled by '$\{\beta(\omega) = \omega\}$'. From this fact and the assumption we made in the last paragraph, it follows that there exists $c \in V'$ which is the greatest element whose tree $T(c)$ contains a path p with $label(p) = \{\beta(c) = d\}$, $c < d$. Since every path of $T(d)$ has a leaf label $\{\beta(d) = d\}$, it suffices to show that $T(d)$ has at least one path p' consistent with p.

We prune every branch of $T(d)$ that is inconsistent with p. Let $\{pred(x)\}$ be a node of $T(d)$. If p contains the same query, then we prune all outgoing edges except at most one edge which agrees with the answer specified in p. If the query is not asked in p, then we prune an outgoing edge $\{y\}$ if p specifies y as the predecessor of a node other than x. Note that in this case at most k outgoing edges of $\{pred(x)\}$ are pruned, and therefore the node still has at least $|V| - |dom(\alpha_i)| - j - 1 - k$ outgoing edges, where j is the depth of the node. We do similarly for an edge specifying a successor.

It is easy to see that, after the pruning, every node of $T(d)$ reachable from its root has at least one outgoing edge. Hence there must be a path from the root to a leaf. □

3.4 The Maximization Problems

In this section, we introduce and study type-2 problems with higher verification complexity.

Definition 3.4.1 *A type-2 search problem $MAXIMIZER$ is defined as follows. For an instance (α, x), let $V = \mathbb{N}^{\leq |x|}$ and define $\alpha_V : V \mapsto V$ from α by*

$$\alpha_V(v) = \begin{cases} \alpha(v) & \textit{if } \alpha(v) \in V, \\ 0 & \textit{otherwise.} \end{cases}$$

Then,

$$MAXIMIZER(\alpha, x) = \{v \in V : (\forall u \in V)\alpha_V(u) \leq \alpha_V(v)\}.$$

□

$MAXIMIZER$ is the problem of finding an *argument* of α_V that realize the maximum value of α_V. Our definition of the problem $MAXIMIZER$ is inspired by the *function maximization problem* of Chiari and Krajicek [CK98].

If we place a restriction on the size of the range of α_V) so that $\alpha_V : V \mapsto \{0, 1, \ldots \log |V|\}$, then we get another interesting search problem $L\text{-}MAXIMIZER$, which is the type-2 counterpart of the *sharply bounded function maximization problem* of [CK98].

Definition 3.4.2 *A type-2 search problem $L\text{-}MAXIMIZER$ is defined as follows. For an instance (α, x), let $V = \mathbb{N}^{\leq |x|}$. Define α^* as*

$$\alpha^*(v) = \begin{cases} \alpha(v) & \textit{if } \alpha(v) \leq |x| \\ 0 & \textit{otherwise} \end{cases}$$

Then,

$$L\text{-}MAXIMIZER(\alpha, x) = \{v \in V : (\forall u \in V)\alpha^*(u) \leq \alpha^*(v)\}.$$

□

The 'L' in $L\text{-}MAXIMIZER$ stands for log, since the range of the function α_V has size logarithm in the size of the domain.

From the definitions above, it is clear that $MAXIMER$ and $L\text{-}MAXIMIZER$ are in $\mathbf{V}^2\mathbf{\Pi}_1^p$.

We show the first type-2 characterization of $\mathbf{EP}^{\mathbf{NP}}[\textit{oblivious}]$ and $\mathbf{EP}^{\mathbf{NP}}[\textit{oblivious}, O(\log n]$.

Theorem 3.4.3

(1) $\mathbf{EP}^{\mathbf{NP}}[\textit{oblivious}] = E(MAXIMIZER)$

(2) $\mathbf{EP}^{\mathbf{NP}}[\textit{oblivious}, O(\textit{log } n)] = E(L\text{-}MAXIMIZER)$

Proof. We show (1) as follows. First, $E(MAXIMIZER) \subseteq \mathbf{EP}^{\mathbf{NP}}[\textit{oblivious}]$ is easy. If $Q \leq_{em} MAXIMIZER$, then

$$Q(x) = \{y : (\exists z)[z \in MAXIMIZER(f[x], g(x)) \wedge y = h(x, z)]\}.$$

Since the function $f[x]$ is polynomial-time, its maximum value can be computed by making $\log |V| = |g(x)|$ NP queries by binary search. Since $|g(x)|$ is bounded by a polynomial in $|x|$, the claim holds.

Next, we show that $\mathbf{EP}^{\mathbf{NP}}[\textit{oblivious}] \subseteq E(MAXIMIZER)$. Let $Q \in \mathbf{EP}^{\mathbf{NP}}[\textit{oblivious}]$ and M be a polynomial-time, oblivious witness-oracle machine with access to an NP predicate R such that $Output_M(x) = Q(x)$ for all x. Let $R(a) \equiv (\exists z)B(a, z)$, $B \in \mathbf{P}$, be the witness oracle that M has access to. Note that there exists a constant c such that, for all x, if (q, w) is an oracle answer to M's witness query, then $|(q, w)| \leq |x|^c + c$. Moreover, there exists a polynomial $p(n)$ such that M on x asks at most $p(|x|)$ witness queries.

Fix an instance x, $|x| = n$, and define $V = \mathbb{N}^{\leq p(n)\cdot(n^c+c)}$. Then, every possible (not necessarily valid) sequence of answers to M's witness queries is encoded by some $v \in V$ of the form $v = (q_1, w_1), (q_2, w_2), \ldots, (q_{p(|x|)}, w_{p(|x|)})$. Define $f : V \mapsto V$ to be a function that maps each $v \in V$ to a binary number $f(v) = b_1 b_2 \ldots b_{p(|x|)}$, where each $b_i \in \{0, 1\}$. Given v, $f(v)$ is computed by a deterministic polynomial-time machine M' as follows.

Given v, M' starts simulating M on x. For $i = 1, 2, \ldots, p(n)$, when M makes the ith witness query a_i, M' test whether $q_i = 1$ and $B(a_i, w_i)$ holds. If succeeds, M' sets $b_i = 1$ and continues the simulation using (q_i, w_i) as an oracle answer. Otherwise, M' sets $b_i = 0$ and use $(q_i, w_i) = (0, 0)$.

Since M is oblivious, all correct sequences of oracle answers have the same $q_1, q_2, \ldots, q_{p(n)}$ and therefore the same $b_1 b_2 \ldots b_{p(n)}$. In fact, if we let $f^* := max_{v \in V} f(v)$, $f(v) = f^*$ if and only if $v = (b_1, w_1), (b_2, w_2), \ldots, (b_{p(n)}, w_{p(n)})$ is a sequence of correct oracle answers. It follows that $MAXIMIZER(f, p(|x|) \cdot |x|^c + c)$ is the set of all possible sequences of correct oracle answers to M's queries. The claim holds, since

$$\begin{aligned} Q(x) = \{y : (\exists v)[\quad & v \in MAXIMIZER(f, p(|x|) \cdot (|x|^c + c)) \\ & \wedge M \text{ on } x \text{ with a sequence } v \text{ of answers outputs } y]\}. \end{aligned}$$

For $E(L\text{–}MAXIMIZER) \subseteq \mathbf{EP}^{\mathbf{NP}}[oblivious, O(\log n)]$, simply observe that the maximum value of $f[x]$ can be found by binary search using $\log |g(x)| \in O(\log n)$ queries. For the other direction, a reduction similar to the above works because $p(n)$ is in $O(\log n)$ in this case. $\square$

Note that it is trivial that $L\text{–}MAXIMIZER \leq_{em} MAXIMIZER$. We have not proved that $MAXIMIZER$ is not exactly reducible to $L\text{–}MAXIMIZER$, but it must be the case because of Chiari and Krajicek's result that is stated in the next chapter. The relationship between the maximizer problems and theories S^1_2 and T^1_2 of bounded arithmetic will also be shown below.

Chapter 4

Bounded Arithmetic and Type-2 Search Problems

4.1 Theories of Bounded Arithmetic

We connect our framework with Buss's theories S_2^i and T_2^i of bounded arithmetic, which have been shown to be closely linked to complexity classes in the polynomial hierarchy [Bus86, Bus95, Bus98, Kra95]. These theories are defined over the language L_{BA} of bounded arithmetic. Let **BASIC** is the set of axioms that define the meaning of the nonlogical symbols in L_{BA} .

Definition 4.1.1 *[Bus86, Bus98] Let Φ be a set of formulas. The Φ–$PIND$ axioms are the formulas*

$$A(0) \wedge (\forall x)[A(\lfloor \frac{x}{2} \rfloor) \supset A(x)] \supset (\forall x)A(x)$$

for all formulas $A \in \Phi$. Similarly, Φ–IND axioms are the formulas

$$A(0) \wedge (\forall x)[A(x) \supset A(Sx)] \supset (\forall x)A(x)$$

for all $A \in \Phi$.

For $i \geq 0$, S_2^i is the theory axiomatized by **BASIC** *axioms plus Σ_i^b-PIND, and T_2^i is the theory axiomatized by* **BASIC** *plus Σ_i^b-IND.* □

The theories S_2^i and T_2^i are related to each other in the following way [Bus86]:

$$S_2^1 \subseteq T_2^1 \subseteq S_2^2 \subseteq T_2^2 \subseteq S_2^3 \subseteq \ldots.$$

However, it is not known whether this hierarchy extends infinitely or collapses at a finite level. This question is related to whether the polynomial-time hierarchy collapses at a finite level [Bus95].

Definition 4.1.2 *[Bus86, Bus98] Let $R(\vec{x})$ be a predicate. We say R is Δ_i^b-defined in theory T if there is a Σ_i^b-formula $A(\vec{x})$ and a Π_i^b-formula $B(\vec{x})$ such that (1) R has a defining axiom $R(\vec{x}) \leftrightarrow A(\vec{x})$ and (2) $T \vdash (\forall \vec{x})[A(\vec{x}) \leftrightarrow B(\vec{x})]$.* □

In other words, a theory T Δ_i^b-defines a predicate R if T can prove that R is representable by both a Σ_i^b formula and a Π_i^b formula.

We write $(\exists! x)$ to mean that "there exists a *unique* x such that ...".

Definition 4.1.3 *[Bus86, Bus98] Let f be a function. We say f is Σ_i^b-defined in theory T if f has a defining axiom $y = f(\vec{x}) \leftrightarrow A(\vec{x}, y)$, where $A \in \Sigma_i^b$ with all free variables indicated, such that $T \vdash (\forall \vec{x})(\exists! y)A(\vec{x}, y)$.* □

The following theorems of Buss show that the computational complexity of a function corresponds to its definability in the hierarchy of bounded arithmetic theories.

Theorem 4.1.4 *[Bus86, Bus98]*

(1) A function is polynomial-time computable if and only if it is Σ_1^b-definable in S_2^1.

(2) Let $i \geq 1$. A function is in $\square_i^p$ if and only if it is Σ_i^b-definable in S_2^i.

Theorem 4.1.5 *[Bus86, Bus98]*

(1) A predicate is polynomial-time computable if and only if it is Δ_1^b-definable in S_2^1.

(2) Let $i \geq 1$. A predicate is in Δ_i^p if and only if it is Δ_i^b-definable in S_2^i.

Chapter 7 of [Kra95] contains more results of this kind.

4.2 Definability of Search Problems

Since our object of study is search problems, the notion of definability should be extended to take into account the multiplicity of solutions. Buss, Krajicek, and Takeuti [BKT93] introduced the following definition.

Definition 4.2.1 *[BKT93] Let Q be a total search problem and T be a theory of bounded arithmetic. Then we say Q is* Σ_i^b-definable *in T if and only if for some Σ_i^b-formula $\phi(x, y)$,*

1. *$T \vdash (\forall x)(\exists y)\phi(x, y)$, and*
2. *$\underline{\mathbb{N}} \models \phi(n, m)$ implies $m \in Q(n)$.*

$\square$

Note that, in the above definition, the formula $\phi(a, b)$ is not required to hold for every solution b of $Q(a)$; in other words, ϕ is not required to represent a defining predicate for Q. This requirement is not strong enough, since the Σ_i^b-definability of Q provides a meaningful bound on neither the verification complexity nor the computational complexity of Q with respect to exact solvability. For example, the search problem U in Example 2.2.4 is Σ_1^b-definable in S_2^1, since $S_2^1 \vdash (\forall x)(\exists y)y = 0$.

We need a stronger notion of definability for search problems, which corresponds to exact solvability. It turns out that *strong Σ_i^b-definablity*, which Buss, Krajicek, and Takeuti introduced in [BKT93] but used in a limited way, is precisely what is required for our purpose. We call it *exact* definability, because of its obvious connection to exact many-one reducibility and exact solvability.

Definition 4.2.2 *Let Q be a search problem, R be a defining predicate of Q, and ϕ a formula of bounded arithmetic. Then, we say ϕ is a* defining formula *of Q if ϕ represents R.* $\square$

Definition 4.2.3 *[BKT93] Let Q be a total search problem and T be a theory. Then we say Q is* exactly Σ_i^b-definable *in T if and only if there exists a Σ_i^b-formula $\phi(x,y)$ such that*

1. *$\phi(a,b)$ is a defining formula of Q, and*
2. *$T \vdash (\forall x)(\exists y)\phi(x,y)$.*

□

We know that a predicate is $\mathbf{\Sigma}_i^p$ if and only if it is represented by a Σ_i^b formula (Theorem 1.3.3). Thus, if Q has a defining predicate $R \in \mathbf{\Sigma}_i^p$, the condition (1) in the above definition is automatically satisfied. On the other hand, if Q is exactly Σ_i^b-definable in T, then it is immediate that $Q \in \mathbf{V\Sigma}_i^p$.

In order to simplify our discussion of classes of search problems exactly definable in theories of bounded arithmetic, we introduce the following notation.

Definition 4.2.4 *Let T be a theory of bounded arithmetic. Then $d[\Sigma_i^b, T]$ and $ed[\Sigma_i^b, T]$ denote the classes of search problems that are Σ_i^b-definable and exactly Σ_i^b-definable in T, respectively.* □

The following lemma shows that exact Σ_i^b-definability is the right notion of definability with respect to exact solvability and $\leq_{em}$.

Lemma 4.2.5 *Let T be a theory such that $S_2^1 \subseteq T$. Then, $ed[\Sigma_i^b, T]$ is closed under $\leq_{em}$.*

Proof. Let $Q_1 \leq_{em} Q_2$ and $Q_2 \in ed[\Sigma_i^b, T]$. Then, there exist functions $f, g \in \square_1^p$ such that

$$Q_1(x) = \{y : (\exists z)[z \in Q_2(f(x)) \wedge y = g(x,z)]\},$$

where '$z \in Q_2(f(x))$' is represented by a Σ_i^b-formula ϕ such that T proves $(\forall x)(\exists y)\phi(x,y)$.

Since $S_2^1 \subseteq T$, T Σ_1^b-define polynomial-time functions f and g; thus, T proves $(\forall x)(\exists y) y = f(x)$ and $(\forall x, z)(\exists y) y = g(x, z)$, where '$y = f(x)$' and '$y = g(x, z)$' are represented by Σ_1^b-formulas. Then, we can construct a Σ_i^b-formula ψ as

$$\psi(a, b) \equiv (\exists w \leq t(a))(\exists z \leq s(a))[w = f(a) \wedge \phi(w, z) \wedge b = g(a, z)] \equiv b \in Q(a),$$

where t and s are terms. It is easy to see that $\psi(a, b)$ represents '$b \in Q_1(a)$' and $T \vdash (\forall x)(\exists y)\psi(x, y)$. □

Chiari and Krajicek [CK98] obtained many interesting results on the relationship between various classes of search problems and classes of Σ_i^b-definable search problems of various theories. Their results are stated in the following framework.

Definition 4.2.6 *[CK98] Let T be a theory of bounded arithmetic and* **S** *be a class of search problems.* **S** *is said to* characterize $d[\Sigma_i^b, T]$ *if the following conditions are satisfied:*
(1) Every problem $Q \in$ **S** *is Σ_i^b-definable in T.*
(2) If $T \vdash (\forall x)(\exists y)\phi(x, y)$ for $\phi \in \Sigma_i^b$, there exists a search problem $Q' \in$ **S** *such that for every $a \in \mathbb{N}$, if $b \in Q'(a)$ then b is of the form $b = \langle c, d_1, d_2, \ldots, d_k \rangle$ and $\phi(a, c)$ holds.* □

The above definition is not satisfactory for our goal because (i) it is based on the notion of simple solvability and (ii) the condition (2) is unnecessarily complicated. (i) can be resolved easily by requiring exact Σ_i^b-definability in (1). Since condition (2) is saying that $\{y : \phi(x, y)\}$ is reducible to Q', (ii) disappears if we always use a class closed under $\leq_{em}$ in place of **S**.

Our results will be stated in the form $\mathbf{S} = ed[\Sigma_i^b, T]$, where **S** is some search class closed under $\leq_{em}$. Such a result shows a close connection between the class **S** and the theory T in the following sense: $Q \in \mathbf{S}$ if and only if $T \vdash (\forall x)(\exists y)[y \in Q(x)]$, where $[y \in Q(x)]$ is represented by a Σ_i^b-formula.

4.3 Exactly Σ_1^b-definable Search Problems

Theorem 4.3.1 $ed[\Sigma_1^b, S_2^1] = \mathbf{SP} \cap \mathbf{V\Sigma}_1^p$.

Proof. First, we show that $ed[\Sigma_1^b, S_2^1] \subseteq \mathbf{SP} \cap \mathbf{V\Sigma}_1^p$. Let $Q \in ed[\Sigma_1^b, S_2^1]$ represented by a Σ_1^b-formula $\phi_Q(a,b)$; thus, $Q \in \mathbf{V\Sigma}_1^p$. Since S_2^1 proves $(\forall x)(\exists y)\phi_Q(x,y)$, by Buss's witness theorem [Bus86, Bus98], there exists $f \in \Box_1^p$ such that for all $n \in N$, $\phi_Q(n, f(n))$ holds. Thus, $Q \in \mathbf{SP}$ and $Q \in \mathbf{V\Sigma}_1^p$.

Next, we prove that any Q in $\mathbf{SP} \cap \mathbf{V\Sigma}_1^p$ is exactly Σ_1^b-definable in S_2^1. Because $Q \in \mathbf{SP}$, there exists a polynomial-time function f such that $f(x) \in Q(x)$ for all x. By (1) of Theorem 4.1.4, there exists a Σ_1^b-formula $\psi_f(a,b)$ that represents '$b = f(a)$' such that $S_2^1 \vdash (\forall x)(\exists y)\psi_f(x,y)$. It then easily follows that $S_2^1 \vdash (\forall x)(\exists y)[\psi_f(x,y) \vee \phi_Q(x,y)]$, where ϕ_Q is a Σ_1^b-formula that represents Q. Note that $b \in Q(a) \equiv \psi_f(a,b) \vee \phi_R(a,b)$; thus, the claim holds. $\Box$

The following can be shown in a similar way, based on (2) of Theorem 4.1.4.

Theorem 4.3.2 *For any $i \geq 2$, $ed[\Sigma_i^b, S_2^i] = \mathbf{SP}^{\mathbf{\Sigma}_{i-1}^p} \cap \mathbf{V\Sigma}_i^p$.*

Proof. Omitted. $\Box$

Buss and Krajicek [BK94] related **PLS** and the $\forall\exists\Sigma_1^b$-consequences of T_2^1:

Theorem 4.3.3 *[BK94]*

*(1) For every $Q \in$ **PLS**, $T_2^1 \vdash (\forall x)(\exists y)\phi_Q(x,y)$, where $\phi_Q(x,y)$ is a Σ_1^b formula that represents '$y \in Q(x)$'.*

(2) If $\psi \in \Sigma_1^b$ and if $T_2^1 \vdash (\forall x)(\exists y)\psi(x,y)$, then there is a polynomial-time projection function f and a PLS problem Q such that whenever $b \in Q(a)$, $\psi(a, f(b))$ holds.

The above theorem is stated in our framework as follows.

Corollary 4.3.4

(1) $\mathbf{PLS} \subseteq ed[\Sigma_1^b, T_2^1]$, *and*

(2) $ed[\Sigma_1^b, T_2^1] \subseteq C[\mathbf{PLS}]$.

We strengthen the above corollary to obtain an exact characterization of $ed[\Sigma_1^b, T_2^1]$.

Theorem 4.3.5 $E(ITER) = ed[\Sigma_1^b, T_2^1]$.

Proof. Recall that $ed[\Sigma_1^b, T_2^1]$ is closed under $\leq_{em}$ (Lemma 4.2.5); therefore, $E[\mathbf{PLS}] \subseteq ed[\Sigma_1^b, T_2^1]$ follows from (1) above. Since $E(ITER) = E[\mathbf{PLS}]$ (Theorem 3.2.5), we have $E(ITER) \subseteq ed[\Sigma_1^b, T_2^1]$.

Next, we show that $ed[\Sigma_1^b, T_2^1] \subseteq E[\mathbf{PLS}]$. Let $Q \in ed[\Sigma_1^b, T_2^1]$ and let $A \in \mathbf{NP}$ be a defining predicate of Q such that $b \in Q(a) \equiv A(a,b) \equiv (\exists z)B(\langle a,b\rangle, z)$. Then, by (2) of Corollary 4.3.4, there exists $Q' \in \mathbf{PLS}$ such that $Q \leq_m Q'$.

We use the technique we used in the proof of Lemma 3.2.13. Let N be a nondeterministic witness-oracle machine with access to Q'. Given x, N computes one solution y for $Q(x)$ by querying Q' once; note that this is done without nondeterminism. Then, N nondeterministically guesses another solution z and a witness w to the $\mathbf{NP}$ predicate '$z \in Q(x)$'. If w witnesses $z \in Q(x)$, then N outputs z and halts. Otherwise, N halts with y in its output tape. Note that $Output_N(x) = Q(x)$.

Suppose N on x requires at most c-bit nondeterministically chosen binary string. Then, it is not hard to construct a PLS problem Q'' such that

$$y \in Q'(x) \iff y2^c + s \in Q''(x),$$

for all $0 \leq s \leq 2^c - 1$. Thus, a deterministic witness-oracle machine M can simulate N by making one witness query to Q''. It follows that $Q \leq_{em} Q''$ and therefore $Q \in E[\mathbf{PLS}]$.

□

Let $T(\alpha)$ be a theory corresponding to T_2^1 in the language $L \cup \{\alpha\}$, where α is a new function symbol with no defining axiom, and let $\Sigma_i^b(\alpha)$, $i \geq 0$, be the set of Σ_i^b-formulas in the language $L_{BA} \cup \{\alpha\}$. Then, the above theorem easily relativizes in the following sense.

Corollary 4.3.6 $E^2(ITER) = ed[\Sigma_1^b(\alpha), T_2^1(\alpha)]$.

Proof. Omitted. $\square$

Let $\alpha : \mathbb{N} \mapsto \mathbb{N}$ be any function. Then, $\exists PIGEON(\alpha)$ is the formula

$$(\exists v_1 \in V)(\exists v_2 \in V)[\alpha_V(v_1) = 0 \vee \alpha_V(v_1) = \alpha_V(v_2)],$$

where $V = \mathbb{N}^{\leq |a|}$ and $\alpha_V : V \mapsto V$ are defined in the same way as in the definition of $PIGEON$ (Definition 3.2.9). Note that a is a free variable. Thus, $\exists PIGEON(\alpha)$ is a formula which asserts the existence of a solution for $PIGEON(\alpha, a)$ for all $a \in \mathbb{N}$, that is, $\exists PIGEON(\alpha)$ is a defining formula for $PIGEON$.

Similarly, let $\exists LEAF(\alpha)$, $\exists SOS(\alpha)$, and $\exists SINK(\alpha)$ be defining formulas for $LEAF$, SOS, and $SINK$.

From the Corollaries 3.3.5 and 4.3.6, we obtain the following unprovability result for $T_2^1(\alpha)$.

Corollary 4.3.7 $T_2^1(\alpha)$ *does not prove any of the following:*

(1) $\exists PIGEON(\alpha)$.

(2) $\exists LEAF(\alpha)$,

(3) $\exists SINK(\alpha)$, *and*

(4) $\exists SOS(\alpha)$,

Proof. We argue for (1). Suppose $T_2^1(\alpha)$ proves $\exists PIGEON(\alpha)$. Then, by Corollary 4.3.6, $PIGEON \in E^2(ITER)$, since $\exists PIGEON(\alpha)$ is a defining formula of $PIGEON(\alpha, x)$. It follows that $PIGEON \leq_{em} ITER$, which contradicts Corollary 3.3.5. $\square$

The unprovability of $\exists PIGEON(\alpha)$ in $T^1_2(\alpha)$ has been known; in fact, $T^1_2(\alpha)$ does not prove that there is no injective mapping from $2a$ to a, $a \geq 1$ [Kra95]. However, we are not aware of any result on the unprovability of the other combinatorial principles $\exists LEAF(\alpha)$, $\exists SINK(\alpha)$, and $\exists SOS(\alpha)$. Moreover, the way we demonstrated this result is new: it is obtained by showing that there is no reduction between the corresponding type-2 search problems.

Let $S^1_2(\Box^p_1)$ be a theory corresponding to S^1_2 obtained by adding to the language a new functions symbol for every function in $\Box^p_1$. Note that $S^1_2(\Box^p_1)$ is conservative over S^1_2 [Bus86, Bus98].

Let Q be a total type-2 search problem and $\mathbf{F}$ be a class of functions. Then $\exists Q(\mathbf{F})$ denotes the set of formulas of the form $\exists Q(f)$ for every $f \in \mathbf{F}$. In other words, $\exists Q(\mathbf{F})$ is the set of defining formulas for $Q(f,a)$ for all $a \in \mathbb{N}$ and $f \in \mathbf{F}$.

The following statement appears without a proof in [Kra95]: a search problem is Σ^b_1-definable in $S^1_2(\Box^p_1) + \exists LEAF(\Box^p_1)$ if and only if it can be witnessed by PPA problem (Theorem 7.5.5). The above statement inspired the following theorem, which explicitly connects the type-2 problems in [BCE+98] and the theories of bounded arithmetic.

Theorem 4.3.8

(1) $E[\mathbf{PPP}] = E(PIGEON) = ed[\Sigma^b_1(\Box^p_1), S^1_2(\Box^p_1) + \exists PIGEON(\Box^p_1)]$.

(2) $E[\mathbf{PPA}] = E(LEAF) = ed[\Sigma^b_1(\Box^p_1), S^1_2(\Box^p_1) + \exists LEAF(\Box^p_1)]$,

(3) $E[\mathbf{PPADS}] = E(SINK) = ed[\Sigma^b_1(\Box^p_1),\ S^1_2(\Box^p_1) + \exists SINK(\Box^p_1)]$*, and*

(4) $E[\mathbf{PPAD}] = E(SOS) = ed[\Sigma^b_1(\Box^p_1), S^1_2(\Box^p_1) + \exists SOS(\Box^p_1)]$.

Proof. We sketch a proof for (1), since the rest can be shown analogously. Note that $\exists PIGEON(\Box^p_1)$ is the set of formulas that represent defining predicates for problems in

$PIGEON(\square_1^p) = \mathbf{PPP}$. Since $S_2^1(\square_1^p) + \exists PIGEON(\square_1^p)$ is closed under $\leq_{em}$ (Lemma 4.2.5), we have $E[\mathbf{PPP}] \subseteq ed[\Sigma_1^b(\square_1^p), S_2^1(\square_1^p) + \exists PIGEON(\square_1^p)]$.

For the other direction, let $Q \in ed[\Sigma_1^b(\square_1^p), S_2^1(\square_1^p) + \exists PIGEON(\square_1^p)]$; we show that $Q \leq_{em} PIGEON$. Note that Q has a defining formula $\phi_Q(x,y) \in \Sigma_1^b(\square_1^p)$ such that

$$S_2^1(\square_1^p) + \exists PIGEON(\square_1^p) \vdash (\forall x)(\exists y)\phi_Q(x,y).$$

Assume, for simplicity, that the proof of $(\forall x)(\exists y)\phi_Q(x,y)$ involves only one instance of $\exists PIGEON(\square_1^p)$; let us call it $\exists PIGEON(f)$. Then,

$$S_2^1(\square_1^p) \vdash \exists PIGEON(f) \supset (\forall x)(\exists y)\phi_Q(x,y).$$

By Buss's witness theorem [Bus86, Bus98], there exists a deterministic polynomial-time machine M such that M on (a,w), where w is a witness to $\exists PIGEON(f)$, outputs b satisfying $\phi_Q(a,b)$ and therefore $b \in Q(a)$. Since w can be obtained by a witness query to $PIGEON$, it follows that $Q \leq_m PIGEON$.

When the proof of $(\forall x)(\exists y)\phi_Q(x,y)$ contains multiple instances of $\exists PIGEON(\square_1^p)$, we obtain witnesses to all of them by one witness query to $PIGEON$. This can be done because a constant number of instances of $PIGEON$ can be combined in a straightforward way so that a solution for the combined instance contains solutions for all the instances.

Finally, $Q \leq_m PIGEON$ can be strengthened to $Q \leq_{em} PIGEON$ by the techniques we used in the proof of Theorem 4.3.5. □

4.4 Exactly Σ_2^b-definable Search Problems

Chairi and Krajicek [CK98] used their *function maximization problem* and *sharply bounded function maximization problem* to characterize $d[\Sigma_2^b, T_2^1]$ and partially $d[\Sigma_2^b, S_2^1]$ in the sense of Definition 4.2.6. Their results inspired our definition of type-2 problems $MAXIMIZER$ and $L\text{-}MAXIMIZER$ (Definitions 3.4.1 and 3.4.2) and our type-2characterization of

$ed[\Sigma_2^b, T_2^1]$ and $ed[\Sigma_2^b, S_2^1]$ below. Since our characterization is based on exact reducibility and exact definability, it is slightly stronger than Chairi and Krajicek's results.

We base our result on the following theorem of Krajicek.

Theorem 4.4.1 *[Kra95]*
(1) $\mathbf{SP}^{\mathbf{NP}}[\mathit{wit}, O(\mathit{log}\ n)] = d[\Sigma_2^b, S_2^1]$, *and*
(2) $\mathbf{SP}^{\mathbf{NP}}[\mathit{wit}] = d[\Sigma_2^b, T_2^1]$.

Note that (2) is not explicitly stated in [Kra95], but it can be shown by modifying Krajicek's proof of (1). We strengthen the above result as below.

Theorem 4.4.2
(1) $\mathbf{EP}^{\mathbf{NP}}[\mathit{oblivious}, O(\mathit{log}\ n)] = ed[\Sigma_2^b, S_2^1]$.
(2) $\mathbf{EP}^{\mathbf{NP}}[\mathit{oblivious}] = ed[\Sigma_2^b, T_2^1]$.

Proof. First, let $Q \in \mathbf{EP}^{\mathbf{NP}}[\mathit{oblivious}, O(\log n)]$ and assume M is a witness-oracle machine that exactly solves Q by asking $O(\log n)$ witness queries. That $Q \in ed[\Sigma_2^b, S_2^1]$ follows directly from Krajicek's proof for (1) of Theorem 4.4.1 (Theorem 6.3.3 of [Kra95]). The idea is to construct a formula $\mathit{WitComp}_M(x, w)$ which states that w encodes a legal computation of M on x. Note that w contains (possibly incorrect) answers to M's witness queries. S_2^1 can prove the existence of w with all answers correct by proving the existence of the largest sequence $q = q_1, q_2, \ldots, q_k$, where $k \in O(\log |x|)$, of yes/no answers, using the Σ_1^b-LENGTH-MAX principle. The output of M on x can be easily extracted from w.

For (2), this direction can be carried out in T_2^1 using the Σ_1^b-MAX principle.

We show the other direction of (1). Let Q be a search problem defined by a Σ_2^b-formula $\phi_Q(x, y)$ of the form $(\exists z \le t(x, y))\psi(x, y, z)$, where $\psi \in \Pi_1^b$. Our goal is to show that if $Q \in ed[\Sigma_2^b, S_2^1]$, then $Q \in \mathbf{EP}^{\mathbf{NP}}[\mathit{oblivious}, O(\log n)]$. Assume $Q \in ed[\Sigma_2^b, S_2^1]$. Then,

$$Q \in ed[\Sigma_2^b, S_2^1] \subseteq d[\Sigma_2^b, S_2^1] = \mathbf{SP}^{\mathbf{NP}}[\mathit{wit}, O(\log n)] = \mathbf{SP}^{\mathbf{NP}}[\mathit{oblivious}, O(\log n].$$

The first inclusion is trivial, and the first equality is from Krajicek's theorem above. The last equality is by Corollary 2.2.14.

Thus, there exists an oblivious polynomial-time witness-oracle machine M such that (i) $Output_M(x) \subseteq Q(x)$ for all x; and (ii) M on x makes at most $c\log|x|$ witness queries to some $R \in \mathbf{NP}$. By applying a technique that Buss, Krajicek, and Takeuti used in [BKT93], we construct another witness-oracle machine M' with access to an $\mathbf{NP}$-complete predicate P that exactly solves Q. Given x, M' simulates M on x until it is about to halt with some $y \in Q(x)$. M' makes two more queries so that M' can output every solution for $Q(x)$. The first query is to obtain two numbers z, w nondeterministically. This is done by asking $(\exists z, w)[|z| \leq |x|^c + c \wedge |w| \leq |x|^d + d]$ with some appropriate c, d that depend on the formula ϕ_Q. Then, M' tests if z is a solution for $Q(x)$ by asking whether $\neg\psi(x, z, w)$ holds, which is an $\mathbf{NP}$ question. Finally, M' outputs z if $z \in Q(x)$, and otherwise M' outputs y, which is obtained by the simulation of M on x.

This direction of (2) is similar. □

Corollary 4.4.3

(1) $E(L\text{-}MAXIMIZER) = ed[\Sigma_2^b, S_2^1]$.

(2) $E(MAXIMIZER) = ed[\Sigma_2^b, T_2^1]$.

Proof. This trivially follows from Theorems 3.4.3 and 4.4.2. □

The above is the first characterization of $ed[\Sigma_2^b, T_2^1]$ and $ed[\Sigma_2^b, S_2^1]$.

Note that Chiari and Krajicek [CK98] showed that the oracle function maximization problem, is not Σ_2^b-definable in a relativized version of the theory S_2^1. Since teh oracle function maximization problem corresponds to $MAXIMIZER$, the above statement implies that $MAXIMIZER \not\leq_{em} L\text{-}MAXIMIZER$.

Chapter 5

Conclusions and Future Works

We developed a new framework for the study of search problems and their definability in bounded arithmetic. It would be fruitful to extend it further by importing into our setting more results obtained in the other contexts outlined in Section 1.1. Another interesting direction is to see if more classes of the form $ed[\Sigma_i^b, T]$ for various i's and T's can be characterized by type-2 search problems. Results in this direction would enable us to separate theories by exhibiting separations of the type-2 problems. The paper by Chiari and Krajicek [CK98] is a natural starting point for this direction.

In Section 3.3, we showed that none of $LEAF$, $SINK$, SOS, and $PIGEON$ is Turing reducible to $ITER$. However, we do not know whether $ITER$ is reducible to any of the above. This question is equivalent to the question of whether $S_2^1(\alpha) + \exists Q(\alpha)$ is $\forall\exists\Sigma_1^b$-conservative over $T_2^1(\alpha)$, where Q is any of the type-2 problems of Beame et. al.

Chiari and Krajicek [CK98] showed that the relativized version of their *function maximization problem*, which is essentially our type-2 $MAXIMIZER$ problem, is not Σ_1^b-definable in a relativized S_2^1. In our context, it implies that $MAXIMIZER$ is not reducible to $L\text{–}MAXIMIZER$. It may be interesting to demonstrate directly the nonexistence of reduction between the problems.

Recall that we obtained the equality

$$E(L\text{–}MAXIMIZER) = ed[\Sigma_2^b, S_2^1]$$

(Corollary 4.4.3) by showing that both the RHS and LHS are equal to $\mathbf{EP}^{\mathbf{NP}}[oblivious, O(\log n)]$ (Theorems 3.4.3 and 4.4.2). It would be interesting to directly show the equality without going through $\mathbf{EP}^{\mathbf{NP}}[oblivious, O(\log n)]$. Similarly, it is open whether we can prove

$$E(MAXIMIZER) = ed[\Sigma_2^b, T_2^1]$$

without going through $\mathbf{EP}^{\mathbf{NP}}[oblivious]$.

Finally, recall that $\mathbf{SP}^{\mathbf{\Sigma}_i^p}[wit, O(f(n))] = \mathbf{SP}^{\mathbf{\Sigma}_i^p}[oblivious, O(f(n))]$ for any $i \geq 1$ and any function f (Lemma 2.2.14), and therefore the oblivious witness-oracle machines are as powerful as the ordinary witness-oracle machines with respect to simple solvability. We do not know whether $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[wit, O(f(n))]$ properly contains $\mathbf{EP}^{\mathbf{\Sigma}_i^p}[oblivious, O(f(n))]$.

Bibliography

[BCE$^+$98] P. Beame, S. A. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences*, 57:3–19, 1998.

[BK94] S. R. Buss and J. Krajicek. An application of Boolean complexity to separation problems in bounded arithmetic. *Proceedings of the London Mathematical Society*, 69:1–27, 1994.

[BKT93] S. R. Buss, J. Krajicek, and G. Takeuti. Provably total functions in bounded arithmetic theories R^i_3, U^i_2 and V^i_2. In P. Clote and J. Krajicek, editors, *Arithmetic, Proof Theory, and Computational Complexity*, pages 116–161. Oxford University Press, 1993.

[Bus86] S. R. Buss. *Bounded Arithmetic.* Bibliopolis, 1986.

[Bus95] S. R. Buss. Relating the Bounded Arithmetic and Polynomial Time Hierarchies. *Annals of Pure and Applied Logic*, 75:67–77, 1995.

[Bus98] S. R. Buss. First-order proof theory of arithmetic. In S. R. Buss, editor, *Handbook of proof theory*, pages 79–147. Elsevier Science, 1998.

[CIY97] S. A. Cook, R. Impagliazzo, and T. Yamakami. A tight relationship between generic oracles and type-2 complexity. *Information and Computation*, 137(2):159–170, 1997.

[CK98] M. Chiari and J. Krajicek. Witnessing functions in bounded arithmetic and search problems. *The Journal of Symbolic Logic*, 63:1095–1115, 1998.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 83–97, 1971.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and intractability.* W. H. Freeman, 1979.

[GKR95] W. I. Gasarch, M. W. Krentel, and K. J. Rappoport. **OptP** as teh normal behabior of NP-complete problems. *Mathematical Systems Theory*, 28:487–514, 1995.

[JPY88] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37:79–100, 1988.

[KPS90] J. Krajicek, P. Pudlak, and J. Sgall. Interactive computations of optimal values. In *MFCS'90, Lecture Notes in Computer Science 452*, pages 48–60, 1990.

[Kra95] J. Krajicek. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, volume 60 of *Encyclopedia of Mathematics and Its Applications.* Cambridge University Press, 1995.

[Kre88] M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.

[Kre92] M. W. Krentel. Generalizations of **OptP** to the polynomial hierarchy. *Theoretical Computer Science*, 97:183–198, 1992.

[MP91] N. Megiddo and C. H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81:317–324, 1991.

[Pap90] C. H. Papadimitriou. On graph-theoretic lemmata and complexity classes. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 794–801, 1990.

[Pap94a] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Pap94b] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48:498–532, 1994.

[PY84] C. H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28:244–259, 1984.

[Sel94] A. L. Selman. A taxonomy of complexity classes of functions. *Journal of Computer and System Sciences*, 48:357–381, 1994.

[SP98] U. Schöning and R. Pruim. *Gems of Theoretical Computer Science.* Springer-Verlag, 1998.

[Sto77] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[Tow90] M. Townsend. Complexity for type-2 relations. *Notre Dame Journal of Formal Logic*, 31(2):241–262, 1990.

[Wra77] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1977.

[Yan97] M. Yannakakis. Computational complexity. In E. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 19–55. John Wiley and Sons Ltd., 1997.