Relativized NP search problems and propositional proof systems

Joshua Buresh-Oppenheim   Tsuyoshi Morioka

University of Toronto
{bureshop, morioka}@cs.toronto.edu

Abstract

An **NP** *search problem is the problem of finding a witness to a given* **NP** *predicate, and* **TFNP** *is the class of* total **NP** *search problems.* **TFNP** *contains a number of subclasses containing natural problems; for example,* **PLS** *is the class of efficient local search heuristics. These classes are characterized by the combinatorial principle that guarantees the existence of a solution; for example,* **PLS** *is the class of such problems whose totality is assured by the principle "every dag with at least one edge has a sink."*

We show many strong connections between these search classes and the computational power—in particular the proof complexity—of their underlying principles. These connections, along with lower bounds in the propositional proof systems Nullstellensatz and bounded-depth LK, allow us to prove several new relative separations among **PLS***, and Papadimitriou's classes* **PPP***,* **PPA***,* **PPAD***, and* **PPADS***.*

1. Introduction

Traditionally the study of computational complexity has been largely a study of decision problems, or the problem of deciding whether the input satisfies a certain property. Consequently, search problems, or the problems of finding an object satisfying a desired property, have been studied in terms of their equivalent decision counterparts. For example, the complexity of finding a Hamiltonian cycle of a graph (if one exists) is studied indirectly via the problem of deciding if the input graph has a Hamiltonian cycle. A justification for this indirect approach is that these search and decision problems are polynomially equivalent, i.e., they are polynomial-time Turing reducible to each other.

However, when a search problem is total, i.e., every instance of it is guaranteed to have a solution, it seems to have no polynomially equivalent decision problem. Such total search problems are commonplace in computer science and mathematics: examples include optimization problems such as the problem of finding a Traveling Salesman tour that is locally optimal with respect to the 2-OPT heuristic, and problems in game theory such as finding a Nash equilibrium given payoff matrices for two players. Thus it is important for us to understand the complexity of total search problems, and we need to study them directly.

In the papers [14, 21], total search problems are classified according to the combinatorial principle in the finite domain that guarantees the totality of the problems. These classes contain numerous natural problems, some of which are complete. For example, *Polynomial Local Search* (**PLS**), which is the class of problems efficiently solvable by local search, is characterized by the iteration principle "every finite dag has a sink"; and *Polynomial Pigeonhole Principle* (**PPP**), which has relevance to cryptographic hash functions, corresponds to the pigeonhole principle "there is no injective mapping from $[n+1]$ to $[n]$." The class *Polynomial Parity Argument* (**PPA**) is defined by the parity principle "there is no perfect matching in an odd-sized graph" and contains the problems of finding various economic equilibria. And its variants **PPAD** and **PPADS** are defined in a similar manner (**PPAD** was called **PSK** in [21], and it is given this name in [2]). All of these problems are **NP** *search problems* in the sense that, given a solution, we can verify its validity in polytime (i.e. each class is contained in **FNP**).

Beame et al. [2] reformulate the search classes in terms of type-2 search problems, or search problems whose input contains not only numbers and strings (type-0 objects) but also functions and relations (type-1 objects) that are presented as oracles. This type-2 approach results in much cleaner definitions of the original type-1 search classes: each class essentially becomes a collection of the type-1 instances of a single type-2 problem. Thus the relationship among these classes can be studied through the corresponding type-2 search problems. In many cases we can obtain unconditional separations of type-2 search problems, which imply oracle separations of the corresponding type-1 search classes. Many such relative containments and separations among the above five search classes are obtained via the type-2 methods in [2, 19].

Since an unrelativized separation of any two **NP** search classes implies $\mathbf{P} \neq \mathbf{NP}$, such relative separations are currently the best results we can hope for. However, we would like to argue that this particular type of oracle result is more meaningful than your garden-variety oracle result, whose relevance has been repeatedly brought into question (starting with [1]). This intuition comes from two sources: (i) Each of these type-2 separations implies that the *generic oracle test* separates the corresponding type-1 classes ([11]). While generic oracles ([4]) are not infallible ([12]), they capture the intuition that an "arbitrary" oracle should not affect the equality of two classes; (ii) Oracle separations of complexity classes are usually obtained by exploiting the difference in the ways these classes access the oracle. In other words, most oracle separations are actually separations of different oracle access methods, and this is why we can 'separate' two classes that are actually equal ([13]), such as **PSPACE** and **IP**. On the other hand, since all the relativized **NP** search classes access an oracle in the same way (via deterministic polynomial-time machines), oracle separations of those classes may better reflect the unrelativized world.

We extend the framework of [2] into a systematic method of formulating type-2 search problems from combinatorial principles. The method is essentially as follows. Let Φ be a first-order existential sentence over an arbitrary language such that Φ holds in every finite structure, and define Q_Φ to be the corresponding type-2 search problem of finding a witness to Φ in a finite structure given as the input. For example, the type-2 problem **PIGEON** of [2] that characterizes the class **PPP** arises from the following sentence:

$$(\forall x)[\alpha(x) \neq 0] \supset (\exists x,y)[x \neq y \wedge \alpha(x) = \alpha(y)],$$

which states that, if 0 is not in the range of a function α, then there must exist two elements that are mapped to the same element by α; this is the injective pigeonhole principle, which holds in every finite structure. Formulated as above, it is clear that studying the complexity of a type-2 search problem Q_Φ amounts to the study of the 'computational power' of the combinatorial principle Φ, which is an interesting mathematical endeavour in its own right.

The most important contribution of this paper is the insight that the proof complexity of Φ is intimately linked to the computational strength of Q_Φ, where the proof complexity of Φ is measured in terms of the size of the shortest proof of the propositional translation of Φ in a given proof system. This close link between proof complexity and computational complexity allows us to utilize the extensive knowledge that has been accumulated in proof complexity research to derive a number of relative separations of **NP** search classes. Our approach is made possible by the type-2 approach that makes explicit the connection between combinatorial principles and search problems.

Main Result 1: Let Q_Φ and Q_Ψ be two type-2 search problems corresponding to the combinatorial principles Φ and Ψ. If $Q_\Phi \leq_m Q_\Psi$, then there are "simple" reductions from the propositional translation of Φ to the propositional translation of Ψ in depth-1.5 tree-like LK and in Nullstellensatz. This result can be seen as a generalization of a technique used in [2], where a relative separation is proven using a Nullstellensatz degree lower bound.

As corollaries, we obtain relative separations of search classes that were not known, such as $\mathbf{PLS}^G \not\subseteq \mathbf{PPA}^G$ with G a generic oracle. Our result generalizes the proof techniques of Beame et al. and hence it provides alternative proofs for some of their results via the proof complexity separations. Moreover, since the combinatorial principle characterizing **PPA** has low-complexity proofs in Nullstellensatz, it follows that the totality of every **PPA** problem has a low-complexity proof. This is interesting because **PPA** contains the witnessing problems for the fixed point theorems of Brouwer, Nash, and Kakutani [21].

We also provide a partial solution for the open question of whether there is an oracle such that **PLS** is not contained in **PPP**.

Main Result 2: There is no 'nice reduction' from **ITERATION** (which characterizes to **PLS**) to **PIGEON** (which characterizes **PPP**).

Our third main result is a sufficient condition for Q_Φ to be nonreducible to **ITERATION**.

Main Result 3: If Φ is a combinatorial principle that does not involve the ordering relation, and if Φ fails in an infinite structure, then Q_Φ is not reducible to **ITERATION**.

This generalizes the relative separation in [19], and it implies that, in a generic relativized world, **PLS** does not contain any of **PPP**, **PPA**, and **PPAD**. This may be interpreted as evidence that efficient local search heuristics are unlikely to exist for these classes. Also we obtain from Main Result 3 an alternative proof for Riis's independence criterion for the bounded arithmetic theory $S^2_2(R)$ [26, 15].

This paper is organized as follows. Section 2 introduces basic definitions of search problems and the proof systems that we use. The search classes of [14, 21, 2] are introduced here, and the known relative separations are stated. In Section 3 we show how to translate combinatorial principles in first-order logic into unsatisfiable propositional formulas and unsatisfiable set of polynomials. Section 4 contains our Main Results 1 and 2. Section 5 presents some of the known proof complexity separations, which imply a number of the search problem separations in Section 6. Section 7 is an exposition of Main Result 3. Section 8 contains concluding remarks and some open problems.

2. Preliminaries

Throughout this paper we write V_n to denote the set of all n-bit strings.

2.1. Search Problems

NP is the class of decision problems that are representable as $(\exists y)R(x,y)$, where R is a polynomial-time predicate such that $R(x,y)$ implies $|y| \leq p(|x|)$ for some polynomial p.

The corresponding **NP** search problem Q_R is the following problem: given x, find any y such that $R(x,y)$ holds. The input x is called an *instance* of Q_R and any y satisfying $R(x,y)$ is called a *solution* for instance x. For every x, $Q_R(x) = \{y : R(x,y)\}$ denotes the set of solutions for instance x. Usually we omit the subscript R. We say that Q is *total* if $Q(x)$ is nonempty for all x. **TFNP** is defined to be the class of total NP search problems in [18] (see also [20]); the same class is called **VP** (for Verification of solutions in Polytime) in [19]. A number of interesting subclasses of **TFNP** have been identified and studied: these classes are **PLS** of [14], and **PPP**, **PPA**, **PPAD**, and **PPADS** of [21, 2]. All of these classes contain natural problems, some of which are complete (under an appropriate notion of reducibility). We will formally define these classes below.

Beame et al. [2] generalize the notion of search problem so that the instances of search problem Q consist not only of strings, which are type-0 objects, but also functions and relations, which are type-1 objects. More formally, let R be a type-2 relation with arguments $(\alpha_1,\ldots,\alpha_k,x,y)$, where x and y are strings and for each i, $1 \leq i \leq k$, α_i is either a string function or a string relation. R defines a type-2 search problem Q_R in the usual way.

The complexity of type-2 relation, functions, and search problems is measured with respect to a Turing machine that receives the type-0 arguments on its input tape and is allowed to access the type-1 arguments as oracles [27]. In particular, a type-2 function $F(\alpha_1,\ldots,\alpha_k,x)$ is said to be polynomial-time computable if it is computed by a deterministic Turing machine in time polynomial in $|x|$ with oracle access to $\alpha_1,\ldots,\alpha_k$.

2.2. Combinatorial Principles and Search Problems

Beame et al. [2] introduce several type-2 search problems that correspond to the combinatorial principles that characterize the search classes of [21]. We extend their approach into a systematic method of defining type-2 search problems from combinatorial principles that are represented as sentences of first-order logic with equality.

Let L be an arbitrary first-order language and let Φ be a sentence over L of the form

$$\Phi \equiv (\exists x_1 \ldots \exists x_k)\phi(x_1,\ldots,x_k)$$

for some quantifier-free ϕ. Let us call such sentences $\exists$-sentences. As usual, we allow the equality symbol $=$ in Φ even though we do not explicitly include it in L. Φ is interpreted in a structure $\mathcal{M}$ which defines the universe of discourse and the meaning of constants, functions, and relations of L. Some symbols of L may be designated as *built-in symbols* with which we associate predetermined interpretation.

Definition 1. *Define a* canonical structure *to be a structure such that (1) the universe of discourse is V_n for some $n \geq 1$; and (2) every* built-in *symbol of L assumes the predetermined interpretation. We abuse the notation and write V_n to denote the canonical structure with the set V_n the universe of discourse.*

The only built-in symbols we use in this paper are $\leq$ and 0, which we interpreted as the standard ordering of n-bit binary numbers and 0^n, respectively.

Assume that Φ holds in every canonical structure. Then the corresponding witness problem is the following: given a canonical structure V_n, find a tuple $\langle v_1,\ldots,v_k\rangle \in (V_n)^k$ such that $\phi(v_1,\ldots,v_k)$ holds in V_n. We formulate the witness problem as the type-2 search problem Q_Φ whose type-0 argument x specifies the universe of discourse $V_{|x|}$ and whose type-1 arguments are the functions and relations of L. Built-in symbols are not part of the type-1 arguments, since their meaning in $V_{|x|}$ is already fixed. Finally, since only the length of x is used to define $V_{|x|}$, we assume without loss of generality that the type-0 argument of Q_Φ is always of the form 1^n for $n \geq 1$.

We introduce below the five combinatorial principles that are of particular interest in the study of search problems. For readability we present them in implicational form; it is easy to see that all of them are $\exists$-sentences. Moreover, all of them hold in every canonical structure.

(1) $f(0) = 0 \wedge (\forall x)[x = f(f(x))] \supset (\exists x)[x \neq 0 \wedge x = f(x)]$
This sentence states that, if every element is either *lonely* (i.e., is matched with itself) or matched with a unique partner, and if 0 is lonely, then there exists another lonely element. This is essentially the parity principle 'no odd-sized graph has a perfect matching', and it holds in every structure whose size is even; therefore, it holds in every canonical structure. **LONELY** is the corresponding search problem.

(2) $(\forall x)[f(x) \neq 0] \supset (\exists x,y)(x \neq y \wedge f(x) = f(y)]$
This states that if 0 is not in the range of f, then there exist two distinct elements that are mapped to the same element by f; this is the injective, functional pigeonhole principle,

and the corresponding search problem is **PIGEON**.

(3) $g(0)=0 \wedge f(0) \neq 0 \wedge (\forall x)[x=g(f(x))] \wedge$
$(\forall x)[x \neq 0 \supset x = f(g(x))] \supset (\exists x,y)[x \neq y \wedge f(x)=f(y)]$
This is a weaker variant of the pigeonhole principle of (2). The additional assumptions essentially state that f is onto and that g is the inverse of f. This is the onto-pigeonhole principle, and the corresponding search problem is **OntoPIGEON**.

(4) $(\exists x_1,y_1,x_2,y_2)[(x_1 \neq x_2 \vee y_1 \neq y_2) \wedge$
$f(x_1,y_1) = f(x_2,y_2)]$
This is the weak pigeonhole principle, which is similar to (2) but the domain size is the square of the range size. We call the corresponding problem **WeakPIGEON**.

(5) $f(0)>0 \wedge (\forall x)[f(x) \geq x] \supset$
$(\exists x)[x < f(x) \wedge f(x) = f(f(x))]$
This is the iteration principle of [8, 9], and we call the corresponding type-2 problem **ITERATION**. It states that, if f is nondecreasing and $f(0)>0$, then there exist x such that $f(x)>x$ and $f(x)=f(f(x))$. Note that it contains a built-in ordering $\leq$. It is equivalent to the principle 'every dag with at least one edge has a sink'.

2.3. Reductions

Let Q be a type-2 search problem. Q can be used as an oracle in the following way. A Turing machine M presents a query to Q in the form $(\beta_1,\ldots,\beta_k,1^m)$, where each of $\beta_1,\ldots,\beta_k$ is a polynomial-time function or relation. In the next step M receives in its answer tape some z that is a solution for $Q(\beta_1,\ldots,\beta_k,1^m)$.

Let Q_1 and Q_2 be two type-2 search problems. We say Q_1 is *Turing reducible to* Q_2 and write $Q_1 \leq_T Q_2$ iff there exists an oracle Turing machine M that, given an instance $(\alpha_1,\ldots,\alpha_k,1^n)$ of Q_1, outputs some $z \in Q_1(\alpha_1,\ldots,\alpha_k,1^n)$ in polynomial-time using $\alpha_1,\ldots,\alpha_k$ and Q_2 as oracles, where each query to Q_2 is of the form $(\beta_1,\ldots,\beta_l,1^m)$ with $m \in n^{O(1)}$ and with each β_i for each $1 \leq i \leq l$, a function or a relation that is polynomial-time computable using $\alpha_1,\ldots,\alpha_k$ as oracles.

Q_1 is *many-one reducible to* Q_2, written $Q_1 \leq_m Q_2$, if $Q_1 \leq_T Q_2$ by an oracle Turing machine that asks at most one query to Q_2. We write $Q_1 \equiv_m Q_2$ if Q_1 and Q_2 are many-one reducible to each other.

When Q_1 is type-1, then Q_1 is treated as a type-2 problem with no type-1 arguments.

Definition 2. *Let Q be a type-2 search problem. Then $C(Q)$ is defined as*

$$C(Q) = \{Q' : Q' \text{ is type-1 and } Q' \leq_m Q\} \cap \mathbf{TFNP}.$$

Thus, $Q' \in C(Q)$ means that Q' is polynomial-time solvable by finding a solution for a type-1 instance of Q that is obtained by fixing the type-1 arguments of Q to be certain predicates and functions that are polynomial-time computable. We take intersection with **TFNP** in the above definition of $C(Q)$ for a technical reason that, without it, $C(Q)$ may not be a set of **NP** search problems [19].

Now we are ready to formulate the search classes of [JPY88, P94] in terms of the type-2 search problems. **PPA** stands for Polynomial Parity Argument, and it is characterized as **PPA** $= C($**LONELY**$)$. This class contains the problems of finding various economic equilibria, some of which are complete. Polynomial Pigeonhole Principle is defined as **PPP** $= C($**PIGEON**$)$, and it has relevance in the study of cryptographic hash functions. These two definitions are from [2]. **PPAD** is an analogue of **PPA** in the directed graphs, and hence the name (**D** is for 'Directed'). **PPAD** $= C($**OntoPIGEON**$)$. Beame et al. characterizes **PPAD** by different combinatorial principles, but they are equivalent to the onto-pigeonhole principle, which we use in this paper. Polynomial Local Search is the class of optimization problems for which efficient local-search heuristics exist, and **PLS** $= C($**ITERATION**$)$. This characterization is essentially in [8] (also in [9]) in the context of bounded arithmetic. Also [19] contains a direct proof in a complexity theoretic setting. For more information on these classes, see [14, 28] for **PLS** and [21] for the other classes.

For a type-2 search problem Q with its type-1 arguments $\beta_1,\ldots,\beta_k$ and an oracle A, $C(Q)^A$ is the class of type-1 **NP** search problems Q' such that $Q' \leq_m Q$ by a reduction that has oracle access to A. This means that the machine M that solves Q' by asking a query to Q and the functions/predicates $\beta_1,\ldots,\beta_k$ provided in the query to Q are all polynomial-time computable with access to A.

Theorem 3. *[11] Let Q_1 and Q_2 be type-2 search problems defined by $\exists$-sentences. The following are equivalent: (i) $Q_1 \leq_m Q_2$; (ii) for all oracles A, $C(Q_1)^A \subseteq C(Q_2)^A$; and (iii) there exists a generic oracle G such that $C(Q_1)^G \subseteq C(Q_2)^G$.*

Theorem 4. *[2] The following hold:*
(i) **OntoPIGEON** $\leq_m$ **LONELY***;*
(ii) **OntoPIGEON** $\leq_m$ **PIGEON***; and*
(iii) **LONELY** *and* **PIGEON** *are incomparable, i.e., neither is many-one reducible to the other.*

The above result completely characterizes the relationship among **PPAD**, **PPA**, and **PPP** via Theorem 3. Clause (iii) of Theorem 4 also follows from our main results below (see Section 6).

PLS is not discussed in [2], and progress for resolving the relative complexity of **PLS** is made in [19]:

Theorem 5. *[19]* **OntoPIGEON** *is not many-one reducible to* **ITERATION**.

Thus, **PLS** contains none of **PPP**, **PPA**, and **PPAD** in a generic relativized world. We present in this paper a generalization of the above result (Theorem 21). It was still unresolved in [19] whether **PLS** is contained in any of the other classes, and we will present below solutions to some of these open problems.

2.4. Search Trees and Reduction

Let M be a Turing machine that, on input of length n, accesses $\alpha : V_n \mapsto V_n$ as an oracle. For each n, a partial function $\rho_n : V_n \mapsto V_n$ is called a *restriction*.

For each n, M gives rise to a *search tree* T_n, which encodes all possible computations of M in terms of α. More specifically, each internal node of T_n denote the queries to α with an outgoing edge for each of $N = |V_n| = 2^n$ possible ways the query can be answered. Each path P of T_n corresponds to a computation of M for a given α and also to a restriction π_P. Each leaf node of T_n is labeled with the output of M for the corresponding computation. The *height* of T_n is defined to be the length of the longest paths from the root to a leaf node, which is the maximum number of queries M asks to α on input of length n. It is easy to define T_n for M that accesses more than one oracle function.

If M runs in polynomial time, then T_n has height polynomial in n and polylogarithmic in N, and the size of T_n is exponential in n and polynomial in N. From now on we measure the size and height of search trees in terms of the maximum branching factor N.

Let Q_1 and Q_2 be type-2 search problems, and assume that the type-1 arguments of Q_1 and Q_2 are $\alpha_1, \ldots, \alpha_k$ and $\beta_1, \ldots, \beta_l$, respectively. Assume that $Q_1 \leq_m Q_2$ by an oracle Turing machine M. Later it becomes useful to split M into the following two polytime oracle Turing machines M_1 and M_2: M_1 is identical to M except that it terminates when M has produced a query $q = (\beta_1, \ldots, \beta_l, 1^m)$ to Q_2, and M_2 simulates the computation of M after it has received an answer to the query. Note that the input to M_2 is therefore of the form $(\alpha_1, \ldots, \alpha_k, 1^n, w)$, where $w \in Q_2(q)$.

Let $T_n^{M_1}$ be a search tree encoding the computations of M_1 on $(\alpha_1, \ldots, \alpha_k, 1^n)$; note that its leaves are labeled with queries to Q_2. Similarly, let $T_n^{M_2}(q,w)$ be the search tree for M_2 on $(\alpha_1, \ldots, \alpha_k, 1^n, w)$, where $w \in Q_2(q)$. Since M on any computation path outputs a solution for Q_1, the following condition is satisfied: If P_1 is a path in $T_n^{M_1}$ ending with a query q to Q_1, and if P_2 is a path in $T_n^{M_2}(q,w)$ with $w \in Q_2(q)$, then the restriction $\pi_{P_1} \cup \pi_{P_2}$ contains a solution for Q_1.

Finally, recall that, if the reduction M asks a query $(\beta_1, \ldots, \beta_l, 1^m)$ to Q_2, then $m \leq p(n)$ for some polynomial p and each β_i is polynomial-time computable using oracles $\alpha_1, \ldots, \alpha_k$. For each $\overline{v} \in (V_m)^{arity(\beta_i)}$, we define $T_m^{\beta_i(\overline{v})}$ to be the search tree for the polynomial-time algorithm computing $\beta_i(\overline{v})$ in terms of $\alpha_1, \ldots, \alpha_k$.

2.5. The Instance Extension Property

Let Q be a type-2 NP search problem. Intuitively, we say that Q has the *instance extension property* if any instance can be converted to an arbitrarily large instance so that the solutions in the large instance correspond to the solutions in the original instance. More formally, Q has the instance extension property iff the following holds: for every polynomial p such that $p(n) \geq n$ for every $n \in \mathbb{N}$, there exist functions $f, \beta_1, \ldots, \beta_k$ such that, if $w \in Q(\beta_1, \ldots, \beta_k, 1^{poly(n)})$, then $f(w) \in Q(\alpha_1, \ldots, \alpha_k, 1^n)$, where f and β_i for each $1 \leq i \leq k$ are polynomial-time computable using α_i as an oracle.

Lemma 6. *Let Q_1 and Q_2 be two type-2 NP search problems such that $Q_1 \leq_m Q_2$. If Q_2 has the instance extension property, then Q_1 is many-one reducible to Q_2 by an oracle Turing machine M whose first oracle query is to Q_2.*

Proof. For simplicity, assume that both Q_1 and Q_2 take one unary function as their type-1 argument. We write α and β to denote the type-1 argument of Q_1 and Q_2, respectively. Let $Q_1 \leq_m Q_2$ by a reduction M which, given $(\alpha, 1^n)$, composes queries $q = (\beta, 1^m)$ to Q_1 based on the answers to the queries to α. Then there exists a polynomial p such that $m \leq p(n)$ for all n and all possible computations of M. We can assume that $p(n) > n$.

Define another reduction M' as follows. Given $(\alpha, 1^n)$, it asks a query $q = (\beta', 1^{p(n)})$ to Q_2 before asking any query to α, where β' is computed by a polynomial-time oracle machine $M_{\beta'}$ as follows. Given $v \in V_{p(n)}$, $M_{\beta'}$ first simulates M to compute $q = (\beta, 1^m)$. By the instance extension property of Q_2, q can be extended to a larger instance $r = (\gamma, 1^{p(n)})$. Finally, $M_{\beta'}$ computes $\gamma(v)$ and outputs it as $\beta'(v)$. After receiving a solution $w \in Q_2(r)$, M' computes $f(w) \in Q_2(q)$, and then simulates M using $f(w)$ as the answer from the Q_2 oracle. $\square$

Assume that $Q_1 \leq_m Q_2$ by a reduction M whose first oracle query is always to Q_2 and split M into two machines M_1 and M_2 as in Section 2.4. Then the tree $T_n^{M_1}$ is empty and hence every path P in $T_n^{M_2}(q,w)$ contains a solution for Q_1 if q is the query that M asks on 1^n and $w \in Q_2(q)$.

All the type-2 search problems introduced in Section 2.2 have the instance extension property. For example, the instance $(\alpha, 1^n)$ of **PIGEON** can be extended to $(\beta, 1^m)$ with any $m > n$ by letting $\beta(u \odot v) = u \odot \alpha(v)$ for any $u \in V_{m-n}$ and $v \in V_n$, where $\odot$ is string concatenation.

2.6. Proof Systems

We consider two propositional proof systems in this paper. The first is called *propositional LK* or *sequent calculus*. Let $A_1,\ldots,A_k,B_1,\ldots,B_\ell$ be propositional formulas over some set of variables $\bar{X}$ and the connectives $\neg,\vee,\wedge$. A sequent is a syntactic object of the form

$$A_1,\ldots,A_k \longrightarrow B_1,\ldots,B_\ell,$$

with the intended meaning

$$A_1 \wedge \ldots \wedge A_k \supset B_1 \vee \ldots \vee B_\ell.$$

The *depth* of such a sequent is the maximum over the depths of each of the formulas. LK usually includes the following rules (A,B are formulas, $\Delta,\Gamma,\Delta',\Gamma'$ are sets of formulas):

Logical Axiom: $\frac{}{A\longrightarrow A}$

Weakening: $\frac{\Gamma\longrightarrow\Delta}{\Gamma'\longrightarrow\Delta'}$ for $\Gamma\subset\Gamma',\Delta\subset\Delta'$

Contraction: $\frac{A,A,\Gamma\longrightarrow\Delta}{A,\Gamma\longrightarrow\Delta}, \frac{\Gamma\longrightarrow\Delta,A,A}{\Gamma\longrightarrow\Delta,A}$

Exchange: $\frac{A,B,\Gamma\longrightarrow\Delta}{B,A,\Gamma\longrightarrow\Delta}, \frac{\Gamma\longrightarrow\Delta,A,B}{\Gamma\longrightarrow\Delta,B,A}$

$\wedge$-introduction: $\frac{A,B,\Gamma\longrightarrow\Delta}{A\wedge B,\Gamma\longrightarrow\Delta}, \frac{\Gamma\longrightarrow\Delta,A \qquad \Gamma\longrightarrow\Delta,B}{\Gamma\longrightarrow\Delta,A\wedge B}$

$\vee$-introduction: $\frac{\Gamma\longrightarrow\Delta,A,B}{\Gamma\longrightarrow\Delta,A\vee B}, \frac{A,\Gamma\longrightarrow\Delta \qquad B,\Gamma\longrightarrow\Delta}{A\vee B,\Gamma\longrightarrow\Delta}$

$\neg$-introduction: $\frac{\Gamma\longrightarrow\Delta,A}{\neg A,\Gamma\longrightarrow\Delta}, \frac{A,\Gamma\longrightarrow\Delta}{\Gamma\longrightarrow\Delta,\neg A}$

Cut: $\frac{A,\Gamma\longrightarrow\Delta \qquad \Gamma\longrightarrow\Delta,A}{\Gamma\longrightarrow\Delta}$

An LK derivation of B from $A_1,\ldots,A_k$ is a sequence of sequents $S_1,\ldots,S_m$ where S_m is $\longrightarrow B$ and each S_i either follows via one of the above rules from earlier sequents or is $\longrightarrow A_j$ for some j. LK is well-known to be derivationally sound and complete; that is, there is a derivation of B from $A_1,\ldots,A_k$ iff $\bigwedge_{i=1}^k A_i$ logically implies B. Thus, if there is a derivation of the empty sequent $\longrightarrow$ then $\bigwedge_{i=1}^k A_i$ is unsatisfiable, and such a derivation is called a *refutation* of $\bigwedge_{i=1}^k A_i$. See [15] for more information on bounded-depth *LK*.

The *size* of a derivation is the sum of the sizes of all formulas mentioned in the derivation, while the depth is the maximum depth of any sequent in the derivation. A derivation is called *tree-like* if each sequent is used at most once to derive a new sequent.

Nullstellensatz is an algebraic proof system. Let F be a field and let $\bar{X}$ be a set of variables. Given polynomials $q_1,\ldots,q_m,p\in F[\bar{X}]$, a Nullstellensatz derivation of p from $q_1,\ldots,q_m$ is another set of polynomials $r_1,\ldots,r_m\in F[\bar{X}]$ such that

$$r_1q_1+\ldots+r_mq_m = p,$$

identically. A *refutation* is a derivation of 1. The *degree* of a Nullstellensatz derivation is the maximum over the degrees of r_iq_i. Nullstellensatz is derivationally sound and complete over any field F in the sense that p can be derived iff it is in the ideal generated by $q_1,\ldots,q_m$. See [7] for more information on Nullstellensatz.

3. Propositional Translations of Type-2 Problems

Let Φ be an $\exists$-sentence over language L. We say that Φ is *basic* if its quantifier-free part is in DNF and contains no nesting of symbols of L. More specifically, Φ is basic iff every atomic formula in Φ is of the form $R(\bar{x})$, $y=f(\bar{x})$, or $x=y$, where R is a predicate symbol and f is a function symbol.

For a type-2 search problem Q_Φ defined by a basic $\exists$-sentence Φ, $CNF(Q_\Phi,n)$ for each n will be the *unsatisfiable* propositional CNF which (falsely) states that Q_Φ is not total. It is the result of a standard translation of $\neg\Phi$ into propositional CNF formulas due to [22]. The following is a more detailed description of the translation: There will be a set of variables in $CNF(Q_\Phi,n)$ for each type-1 argument α for Q_Φ. If α is an m-ary relation, then, for each m-tuple $\bar{v}$ in the domain of α, there is a propositional variable $X_{\alpha(\bar{v})}$, which is intended to assert that $\alpha(\bar{v})$ is true. If α is a function, we add propositional variables for the relation $graph(\alpha)$: that is, for each $\bar{v}$ in the domain of α and each w in the target of α, we add $X_{\alpha(\bar{v}),w}$, which asserts that $\alpha(\bar{v})=w$.

More specifically, $CNF(Q_\Phi,n)$ is defined as

$$CNF(Q_\Phi,n) =_{syn} F_\Phi \wedge F_{Def} \wedge F_{SingleDef},$$

where F_Φ, F_{Def}, and $F_{SingleDef}$ are CNF formulas described below. First, F_Φ is of the following form:

$$F_\Phi =_{syn} \bigwedge_{x_1,\ldots,x_k\in V_n} \neg\phi(x_1,\ldots,x_k),$$

where $\neg\phi$ is in CNF with each atomic formula replaced by its corresponding propositional variable or propositional constants. If an atom either contains any built-in predicate or function, or of the form $x=y$, then it is replaced with either $true$ or $false$, depending on their truth value in the canonical structure V_n.

F_{Def} is a CNF formula stating that every function in L is total; more specifically, F_{Def} is the conjunction of the following clauses

$$def_n(\alpha(\bar{v})) =_{syn} \bigvee_{w\in V_n} X_{\alpha(\bar{v}),w}$$

for every function $\alpha \in L$ and every $\bar{v} \in (V_n)^{arity(\alpha)}$. Similarly, $F_{SingleDef}$ is the conjunction of the formulas

$$singledef_n(\alpha(\bar{v})) =_{syn} \bigwedge_{u,w \in V_n} \neg X_{\alpha(\bar{v}),w} \vee \neg X_{\alpha(\bar{v}),w'}$$

for every α, every $\bar{v} \in (V_n)^{arity(\alpha)}$, and $w, w' \in V_n$ with $w \neq w'$.

Note that all the clauses in $CNF(Q_\Phi, n)$ have constant size except the clauses of F_{Def}, which have size $|V_n| = N$. The number of clauses in the CNF is polynomial in N.

Let $CNF(Q_\Phi)$ be the family of formulas $\{CNF(Q_\Phi, n)\}_{n \in \mathbb{N}}$. We emphasize that $CNF(Q_\Phi)$ is a family of *unsatisfiable* CNF formulas asserting, for each n, that Q_Φ has no solutions.

If we start with Φ that is in DNF but not basic, we modify it to obtain a basic sentence as follows. If Φ contains an atomic formula ϕ with nesting of symbols, say $y = f(g(x))$, then replace ϕ with $(\exists z)[z = g(x) \wedge y = f(z)]$ if ϕ is unnegated and with $(\forall z)[z = g(x) \supset y = f(z)]$ if ϕ is negated. Treat the other cases $f(x) = g(y)$ and $R(f(x))$ in a similar way. After all atoms are made basic, then make the whole sentence prenex with the quantifier-free part in DNF. Let Φ' be the resulting $\exists$-sentence. It is clear that $Q_\Phi \equiv_m Q_{\Phi'}$.

We also translate of search problems into sets of polynomials in the following way. First, $poly_\Phi$ contains a polynomial for each clause C of F_Φ that is obtained in the usual way: each literal of C forms a linear factor of the polynomial, where a positive literal x becomes a factor $1 - x$ and a negative literal $\neg x$ becomes a factor x.

Next, $poly_{Def}$ contains a polynomial

$$\sum_{w \in V_n} X_{\alpha(\bar{v}),w} - 1,$$

for each function α of L and $\bar{v} \in (V_n)^{arity(\alpha)}$. Similarly, $poly_{SingleDef}$ consists of polynomials

$$X_{\alpha(\bar{v}),w} X_{\alpha(\bar{v}),w'}$$

for each function α of L, $\bar{v} \in (V_n)^{arity(\alpha)}$, and $w, w' \in V_n$ with $w \neq w'$.

Finally, $poly(Q_\Phi, n)$ is the union of $poly_\Phi$, $poly_{Def}$, $poly_{SingleDef}$ plus the additional polynomials forcing each variable x to take on 0/1 values: $x - x^2$. Let $poly(Q_\Phi)$ be the family $\{poly(Q_\Phi, n)\}_{n \in \mathbb{N}}$.

4. Search Problem Reductions and Proof Complexity Reductions

4.1. Bounded-depth LK

Let L be an arbitrary first-order language consisting of functions. Let M be a polynomial-time oracle Turing machine that accesses the functions of L as oracles on a domain of size $N = 2^n$. As in Section 2.4, M gives rise to the family $\{T_n\}_{n \geq 1}$ of search trees of height polylogarithmic in N. Define, as in Section 3, the propositional variables $X_{\alpha(\bar{v}),w}$ for each function $\alpha_j \in L$. Then each path P of T_n uniquely corresponds to the conjunction Π_P of variables such that a variable $X_{\alpha(\bar{v}),w}$ appears in Π_P iff the query '$\alpha(\bar{v}) = ?$' is answered with w in P. It is clear that the size of Π_P is equal to the length of P. We will simply write P to mean a path and the corresponding conjunction. We define $disj(T_n)$ to be the DNF formula $\bigvee_{P \in S} P$, where S is the set of all paths in T_n.

We will use the following two lemmas many times in our LK derivations. The first one says that we can easily prove the totality of any function defined as a decision tree of total functions. The second one says that we can easily prove that any function defined as a decision tree of well-defined functions is well-defined.

Lemma 7. *Let L, M, and $\{T_n\}_{n \geq 1}$ be defined as above, and let F_{Def} be the CNF formula asserting that every function in L is total. For each n, $\rightarrow disj(T_n)$ has a a bounded-depth LK derivation from clauses of F_{Def} of size quasipolynomial in N.*

Proof. Let $P_1, \ldots, P_m$ be the paths of T_n. Let $\alpha_1(\bar{v}_1), \ldots, \alpha_\ell(\bar{v}_\ell)$ be all the queries to Q_Φ that appear in T_n. Our goal is to derive the sequent S of the form

$$S: \quad def(X_{\alpha_i(\bar{v}_i)}), \ldots, def(X_{\alpha_l(\bar{v}_\ell)}) \longrightarrow disj(T_n),$$

since then $\rightarrow disj(T_n)$ is easily obtained from S and $\rightarrow def(X_{\alpha_i(\bar{v}_i)})$ for each i by weakening and cut.

For every node k of T_n, let S_k be the sequent

$$S_k: \quad R_k, def(X_{\alpha_{i_1}(\bar{v}_{i_1})}), \ldots, def(X_{\alpha_{i_k}(\bar{v}_{i_j})}) \longrightarrow P_1, \ldots, P_k$$

where R_k is the set of variables corresponding to the path from the root to k, and $\alpha_{i_1}(\bar{v}_{i_1}), \ldots, \alpha_{i_j}(\bar{v}_{i_j})$ are all the queries in the subtree of T_n rooted at x.

We argue that the sequent S_k for every node k of T_n has a desired bounded-depth *LK* derivation; in fact, the derivation is essentially an upside-down copy of T_n itself. First, let k be a leaf node. Then S_k essentially contains R_k on both sides, and therefore it has a short derivation from a logical axiom. If k is a nonleaf node labeled with the query $\alpha(\bar{u})$, then S_k contains

$$def(\alpha(\bar{u})) =_{syn} X_{\alpha(\bar{u}),0^n} \vee \ldots \vee X_{\alpha(\bar{u}),1^n}$$

in the left side. If k' is a child node of k by the answer '$\alpha(\bar{u}) = w$', then $R_{k'}$ is $\{X_{\alpha(\bar{u}),w}\} \cup R_k$. It is easy to see that S_k is derived from $S_{k'}$ for all child nodes k' by weakening and $\vee$-left-introduction. Since T has quasi-poly size, so will this derivation. $\square$

Lemma 8. *Let L, M, and $\{T_n\}_{n\geq 1}$ be defined as above, and let $F_{SingleDef}$ be the CNF formula asserting that every function in L is well-defined. Let D_1 be a disjunction of some of the terms in $disj(T_n)$, and let D_2 be the disjunction of the remaining terms. Then there is a bounded-depth LK derivation of $D_1, D_2 -\!\!\to$from clauses of $F_{SingleDef}$ of size quasipolynomial in N.*

Proof. Let $D_1 = \bigvee_i D_1^i$, and let $D_2 = \bigvee_j D_2^j$. Any two terms D_1^i and D_2^j must differ on at least one query since they correspond to two different paths in the same tree. Assume that on D_1^i, we have $\alpha_{ij}(\bar{v}_{ij}) = w_{ij}$ and on D_2^j, $\alpha_{ij}(\bar{v}_{ij}) = w'_{ij}$. Begin with the sequents

$$D_1^i, D_2^j -\!\!\to X_{\alpha_{ij}(\bar{v}_{ij}),w_{ij}} \qquad \text{and} \qquad D_1^i, D_2^j -\!\!\to X_{\alpha_{ij}(\bar{v}_{ij}),w'_{ij}},$$

for each i, j. By $\neg$- and $\vee$- introduction, we get

$$\neg X_{\alpha_{ij}(\bar{v}_{ij}),w_{ij}} \vee \neg X_{\alpha_{ij}(\bar{v}_{ij}),w'_{ij}}, D_1^i, D_2^j -\!\!\to.$$

By weakenings and $\vee$-introductions, we get

$$\{\neg X_{\alpha_{ij}(\bar{v}_{ij}),w_{ij}} \vee \neg X_{\alpha_{ij}(\bar{v}_{ij}),w'_{ij}}\}_{ij}, D_1, D_2 -\!\!\to.$$

But cutting with $F_{SingleDef}$, we get $D_1, D_2 -\!\!\to$ This derivation again has quasi-polynomial size. □

We adopt the following definition for comparing the proof complexity of different families of propositional formulas.

Definition 9. *Let $\mathcal{S}, \mathcal{R}$ be two families of unsatisfiable CNF formulas. We say $\mathcal{S}$ has a bounded-depth LK reduction to $\mathcal{R}$, written $\mathcal{S} \leq_{bd-LK} \mathcal{R}$, if we can derive a substitution instance $\mathcal{R}'$ of $\mathcal{R}$ from $\mathcal{S}$ in quasi-polynomial-size bounded-depth LK.*

Let $\mathcal{S}, \mathcal{R}$ be families of unsatisfiable CNF formulas such that $\mathcal{S} \leq_{bd-LK} \mathcal{R}$. Obviously the substitution instances of $\mathcal{R}$ derived from $\mathcal{S}$ have size quasipolynomial in N. Thus, if every substitution instance $\mathcal{R}'$ of $\mathcal{R}$ has bounded-depth LK refuations of size quasipolynomial in $|\mathcal{R}'|$, then so does $\mathcal{S}$.

Theorem 10. *Let Φ and Ψ be two first-order $\exists$-sentences and Q_Φ, Q_Ψ be the corresponding type-2 NP search problems. Assume that Q_Ψ has the instance extension property. If $Q_\Phi \leq_m Q_\Psi$, then $CNF(Q_\Phi) \leq_{bd-LK} CNF(Q_\Psi)$. In fact, the derivations are tree-like and have depth-*1.5.

Proof. We assume that the type-1 arguments of Q_Φ and Q_Ψ are $\alpha_1, \ldots, \alpha_k$ and and $\beta_1, \ldots, \beta_{k'}$, respectively. We use X with subscripts to denote the variables of $CNF(Q_\Phi)$ and Y with subscripts for the variables of $CNF(Q_\Psi)$.

Since Ψ satisfies the model extension property, $Q_\Phi \leq_m Q_\Psi$ by a polynomial-time oracle Turing machine M whose query $q = (\beta_1, \ldots, \beta_{k'}, 1^{n'})$ only depends on 1^n, the string argument of Q_Φ. Fix n arbitrary. Recall the definition in Section 2.4 of the search tree $T_{n'}^{\beta_i(\bar{v})}$ computing the value of $\beta_i(\bar{v})$. For each $w \in V_{n'}$, define $DNF(Y_{\beta(\bar{v}),w})$ to be the disjunction

$$\bigvee_{P\in S} P,$$

where S is the set of paths of $T_{n'}^{\beta_i(\bar{v})}$ that end with a leaf stating that $\beta_i(\bar{v}) = w$. Let $DNF(\neg Y_{\beta(\bar{v}),w})$ be

$$\bigvee_{P\in S} P,$$

where S is the set of paths of $T_{n'}^{\beta_i(\bar{v})}$ that end with a leaf stating that $\beta_i(\bar{v}) \neq w$. Because of Lemmas 7 and 8, there is an equivalence in LK between $\neg DNF(Y_{\beta(\bar{v}),w})$ and $DNF(\neg Y_{\beta(\bar{v}),w})$. Therefore, we will ignore the syntactic difference between the two. Define $CNF(Q_\Psi, n')[T/\overline{Y}]$ as the result of substituting for each variable $Y_{\beta_i(\bar{v}),w}$ the formula $DNF(Y_{\beta_i(\bar{v}),w})$.

Below we show that bounded-depth *LK* derives from $CNF(Q_\Phi, n)$ the substitution instance $CNF(Q_\Psi, n')[T/\overline{Y}]$, the derivation is of size quasipolynomial in N. It suffices to show that, for each clause C of $CNF(Q_\Psi, n')$, its substitution instance $C[T/\overline{Y}]$ has quasipolysize derivation from the clauses of $CNF(Q_\Phi, n)$. Three cases arise.

(i) *If all of Q_Φ's functions are total, then so are all of Q_Ψ's:* If C is $def(\beta_i(\bar{v}))$, then $C[T/\overline{Y}]$ is just $disj(T_{n'}^{\beta_i(\bar{v})})$, which has a desired derivation by Lemma 7.

(ii) *If all of Q_Φ's functions are well-defined, then so are all of Q_Ψ's:* Let C be a clause of $F_{SingleDef}(\Psi)$: $\neg(Y_{\beta_i(\bar{v}),w}) \vee \neg(Y_{\beta_i(\bar{v}),w'})$ for some $w \neq w'$. The formula $Y_{\beta_i(\bar{v}),w}[T/\overline{Y}]$, is a disjunction of all paths in $T_{n'}^{\beta_i(\bar{v})}$ that lead to something other than w with all paths that lead to something other than w'. This disjunction includes all paths in $disj(T_{n'}^{\beta_i(\bar{v})})$, so it is easily derivable by Lemma 7.

(iii) *If we find a solution to Q_Ψ, then we can find one for Q_Φ:* Consider a clause C of F_Ψ. Assume C mentions variables $Y_{\bar{v}_1,w_1}^{\beta_1}, \ldots, Y_{\bar{v}_k,w_k}^{\beta_k}$. For each $1 \leq i \leq k$, let $T_i = T_{n'}^{\beta_i(\bar{v}_i)}$. Let $\mathcal{P}$ be the set of conjunctions $\{(P_1 \wedge \ldots \wedge P_k) \mid P_1 \in T_1, \ldots, P_k \in T_k\}$, where $P_i \in T_i$ means that P_i is a path in T_i. $\mathcal{P}$ is just $disj(T)$ where T is the tree that starts with T_1 and puts copies of T_2 at each leaf of T_1, etc. Therefore, by Lemma 7, the sequent $-\!\!\to\mathcal{P}$ has a quasipolysize derivation. First, we can easily remove from $-\!\!\to\mathcal{P}$ the conjunctions containing inconsistent paths. Next, we want to remove the conjunctions that falsify C.

Assume that $P \in \mathcal{P}$ falsifies C, where $P = (P_1 \wedge \ldots \wedge P_k)$. Then P defines a solution w for Q_Ψ. Recall from Section 2.4 that $T_n^{M_2}[q,w]$ encodes all possible computations of M after it receives w as the answer to the Q_2-query q. By Lemma 7, $-\!\!\to disj(T_n^{M_2}[q,w])$ has a quasipolysize derivation. Let R be an arbitrary path in $T_n^{M_2}[q,w]$. If R is inconsistent with

P, then the sequent $P,R \longrightarrow$ has a quasipolysize derivation. If R is consistent with P, then, by the definition of M_2, R contains a solution for Q_Φ and therefore it falsifies some F_Φ-clause B. Starting from $\longrightarrow B$, we get the sequent $B' \longrightarrow$, where B' is a set of literals that are the negations of those literals in B. Since R extends B', we can derive $R \longrightarrow$. By cutting with these sequents, we derive $P \longrightarrow$ from $\longrightarrow disj(T_n^{M_2}[q,w])$. Using this sequent, we remove P from $\longrightarrow \mathcal{P}$.

Now we have simply

$$\longrightarrow \mathcal{P}',$$

where $\mathcal{P}'$ is the set of all $P \in \mathcal{P}$ that satisfies C. Write C as $l_1 \vee \ldots, \vee l_k$, where each l_i is either $Y_{\beta_i(\bar{v}_i),w_i}$ or its negation. For each path $P \in \mathcal{P}'$, there is an i such that P contains one of the paths in $DNF(l_i)$. Therefore, it is easy to derive

$$\longrightarrow DNF(l_1), \ldots, DNF(l_k).$$

Again this derivation has quasi-polynomial size. □

4.2. Nullstellensatz

We saw above that a poly-time reduction between search problems yields a quasi-poly-size bounded-depth LK reduction between the corresponding propositional formulas. Here we show a similar connection to the Nullstellensatz system. The structure of the presentation is analogous to that above. Throughout this section, we work in an arbitrary field F.

Let L, M, $\{T_n\}$ be as in section 4.1. For each path P in T_n, we form a monomial as follows: if the query $\alpha(\bar{v}) = w$ appears in P, then the variable $X_{\alpha(\bar{v}),w}$ appears in the monomial. Clearly, the degree of the monomial is the length of P. In general, we will abuse notation and use P to refer to both the path and the monomial. We associate a polynomial, $poly(T_n)$ with the tree T_n—namely, the sum of the monomials for each path in T_n. The polynomial $poly(T_n)$ has degree equal to the height of the tree T_n.

Lemma 11. *Let L, M, $\{T_n\}$ be as above. For each n, the polynomial $poly(T_n) - 1$, has a Nullstellensatz derivation from $poly_{Def}$ of degree polylogarithmic in N.*

Proof. Fix n and let $T = T_n$. We prove induction on the height of T that $poly(T)$ has a derivation of degree $height(T)$. If $height(T) = 0$, then we consider T to have one path p of length 0 and let $poly(T) = 1$. Otherwise, let T have height $k > 0$. Consider the tree T', the subtree of T where every path from the root is truncated at length $k-1$. By induction, $poly(T') - 1$ has a Nullstellensatz derivation of degree $k-1$. Consider any leaf l of T' that is not a leaf of T and assume it queries α on $\bar{u}$ in T. Let T'_l be the tree T' with every T-child of l added on. Then $poly(T'_l) - poly(T') = poly_{Def}(X_{\alpha(\bar{u})})p_l$, where p_l is the (monomial of the) path from the root to l. We know $poly(T) - poly(T')$ is just the sum of $poly(T'_l) - poly(T')$ for all such leaves l. Hence $poly(T)$ has a degree k Nullstellensatz derivation. □

Definition 12. *Let F be a field and let $\bar{X}$ and $\bar{Y}$ be infinite sets of variables. Let P_1 be an infinite family of finite subsets of $F[\bar{X}]$ and let P_2 be an infinite family of finite subsets of $F[\bar{Y}]$. We say that P_1 has a degree-d Nullstellensatz reduction to P_2 ($P_1 \leq_{HN(d)} P_2$), if, for any $A \in P_1$ there is a $B \in P_2$ and a set of polynomials $\bar{f}_Y = \{f_Y\}_{Y \in \bar{Y}} \subset F[\bar{X}]$ such that each polynomial in $B(\bar{f}_Y/\bar{Y})$ has a degree-d Nullstellensatz derivation from A. $B(\bar{f}_Y/\bar{Y})$ is just the result of replacing each variable Y in each polynomial of B by f_Y.*

Theorem 13. *Let Φ and Ψ be two first-order $\exists$-sentences and Q_Φ, Q_Ψ be the corresponding type-2 NP search problems. Assume that Q_Ψ has the instance extension property. If $Q_\Phi \leq_m Q_\Psi$, then $poly(Q_\Phi) \leq_{HN(d)} poly(Q_\Psi)$ over any field for some d that is polylogarithmic in N.*

Proof. Again, we assume that the type-1 arguments of Q_Φ and Q_Ψ are $\alpha_1, \ldots, \alpha_k$ and and $\beta_1, \ldots, \beta_{k'}$, respectively. We use X with subscripts to denote the variables of $poly(Q_\Phi)$ and Y with subscripts for the variables of $poly(Q_\Psi)$.

Since Ψ satisfies the model extension property, $Q_\Phi \leq_m Q_\Psi$ by a polynomial-time oracle Turing machine M whose query $q = (\beta_1, \ldots, \beta_{k'}, 1^{n'})$ only depends on 1^n, the string argument of Q_Φ. Fix n arbitrary. Recall the definition in Section 2.4 of the search tree $T_{n'}^{\beta_i(\bar{v})}$ computing the value of $\beta_i(\bar{v})$. For each $w \in V_{n'}$, define $poly(Y_{\beta(\bar{v}),w})$ to be the sum

$$\sum_{P \in S} P,$$

where S is the set of paths of $T_{n'}^{\beta_i(\bar{v})}$ that end with a leaf stating that $\beta_i(\bar{v}) = w$. Let $poly(1 - Y_{\beta(\bar{v}),w})$ be

$$\sum_{P \in S} P,$$

where S is the set of paths of $T_{n'}^{\beta_i(\bar{v})}$ that end with a leaf stating that $\beta_i(\bar{v}) \neq w$. Because of Lemma 11, there is an equivalence in Nullstellensatz between $1 - poly(Y_{\beta(\bar{v}),w})$ and $poly(1 - Y_{\beta(\bar{v}),w})$. Therefore, we will ignore the syntactic difference between the two. Define $poly(Q_\Psi, n')[T/\overline{Y}]$ as the result of substituting for each variable $Y_{\beta_i(\bar{v}),w}$ the polynomial $poly(Y_{\beta_i(\bar{v}),w})$.

Below we show that Nullstellensatz derives from $poly_\Phi$ the substitution instance $poly(Q_\Psi, n')[T/\overline{Y}]$ and that the derivation is of degree polylogarithmic in N. It suffices to show that, for each polynomial τ in $poly(Q_\Psi, n')$, its substitution instance $\tau[T/\overline{Y}]$ has a polylogarithmic degree derivation from the polynomials of $poly(Q_\Phi, n)$. Four cases arise.

(i) *If all of Q_Φ's functions are total, then so are all of Q_Ψ's:* If τ is $poly_{Def}(\beta_i(\bar{v}))$, then $\tau[T/\bar{Y}]$ is $poly(T_{n'}^{\beta_i(\bar{v})}) - 1$, which has a desired derivation by Lemma 11.

(ii) *If all of Q_Φ's functions are well-defined, then so are all of Q_Ψ's:* If τ is in $poly_{SingleDef}$, then τ is $Y_{\beta_i(\bar{v}),w}Y_{\beta_i(\bar{v}),w'}$ for some $w \neq w'$. Every term t in $\tau[T/\bar{Y}]$ is the product of two different paths in $T_{n'}^{\beta_i(\bar{v})}$, so something in $poly_{SingleDef}(Q_\Phi)$ is a factor of this term.

(iii) *If we find a solution to Q_Ψ, then we can find one for Q_Φ:* Now, consider a polynomial τ of $poly_\Psi$. Basically, τ is a clause which rules out a certain solution w to Q_Ψ. As in Section 2.4, let $T_n^{M_2}[q,w]$ be the tree that encodes all possible computations of the reduction M after it receives w as the answer to a Q_Ψ-query q. Therefore, every (noncontradictory) term of the polynomial

$$P = poly(T_n^{M_2}[q,w])\tau[T/\bar{Y}]$$

contains a solution to Q_Φ. Hence, each term in P is easily derivable from something in $poly_\Phi$. Moreover, $poly(T_n^{M_2}[q,w]) - 1$ is derivable by Lemma 11. Then

$$\tau[T/\bar{Y}] = P - \tau[T/\bar{Y}](poly(T_n^{M_2}[q,w]) - 1).$$

(iv) *$poly(Q_\Psi)$'s variables are boolean:* The Nullstellensatz setting requires one more case: $\tau = Y_{\beta_i(\bar{v}),w} - Y^2_{\beta_i(\bar{v}),w})$. But then $\tau[T/\bar{Y}]$ is just $poly(Y_{\beta_i(\bar{v}),w})poly(1 - Y_{\beta_i(\bar{v}),w})$, so every term is the product to two different paths in the same tree. Hence, every term is derivable from something in $poly_{SingleDef}(Q_\Phi)$.

It is clear that none of these four cases involves high degree polynomials in any way.

□

5. Proof Complexity Separations

In this section we show a number of proof complexity separations which, together with Theorems 10 and 13, imply separations of type-2 search problems. Note that the CNF formulas CNF(**PIGEON**), CNF(**WeakPIGEON**), CNF(**LONELY**), and CNF(**WeakPIGEON**) are equivalent to the CNF formulas whose proof complexity have been studied extensively. CNF(**ITERATION**) is equivalent to the *housesitting principle* of [10, 7].

Lemma 14. *The following separations hold in bounded-depth LK:*
*(a) CNF(**PIGEON**) $\not\leq_{bd-LK}$ CNF(**WeakPIGEON**).*
*(b) CNF(**LONELY**) $\not\leq_{bd-LK}$ CNF(**PIGEON**).*
*(c) CNF(**PIGEON**) $\not\leq_{bd-LK}$ CNF(**ITERATION**).*
*(d) CNF(**LONELY**) $\not\leq_{bd-LK}$ CNF(**ITERATION**).*

Proof. [23, 16] show that CNF(**PIGEON**) requires exponential-size refutations in any bounded-depth system. [3] show (b), and hence CNF(**LONELY**) requires exponential-size bounded-depth LK refutations. On the other hand, [17] show that CNF(**WeakPIGEON**) has quasi-poly-size 0.5-depth LK refutations, and Lemma 15 below shows that CNF(**ITERATION**) has poly-size tree-like resolution refutations. □

Lemma 15. *CNF(**ITERATION**) has poly-size tree-like resolution refutation.*

Proof. Fix arbitrary $n \in \mathbb{N}$ and let $N = 2^n$. CNF(**ITERATION**$,n$) consists of the following clauses:
(i) $\neg X_{0,0}$
(ii) $\neg X_{i,j}$ for all i,j such that $j < i$
(iii) $\neg X_{i,j} \vee \neg X_{j,j}$ for all i,j such that $i < j$
(iv) $\bigvee_{0\leq j\leq N-1} X_{i,j}$ for every i
(v) $\neg X_{i,j} \vee \neg X_{i,k}$ for all i,j,k with $j \neq k$
For every $i \geq 1$, define A_i to be the clause $\bigvee_{j\geq i} \neg X_{j,j}$. A_1 is derivable from clauses (i), (iii), and (iv) for $i = 0$. Similarly, for every $i \geq 1$, the clause $X_{i,i} \vee A_{i+1}$ is derived using (ii), (iii), and (iv). Thus, for every $i \geq 2$, A_i is derived by resolving A_{i-1} and $X_{i-1,i-1} \vee A_i$ on $P_{i-1,i-1}$. Finally, the empty clause is derived from $A_n = \neg P_{N,N}$ and $P_{N,N}$, which is derived from (ii) and (iv). □

Lemma 16. *The following separations hold for degree-d Nullstellensatz whenever d is polynomial in n:*
*(a) $poly$(**PIGEON**) $\not\leq_{HN(d)}$ $poly$(**OntoPIGEON**) over any field F.*
*(b) $poly$(**ITERATION**) $\not\leq_{HN(d)}$ $poly$(**OntoPIGEON**) over any field F.*
*(c) $poly$(**PIGEON**) $\not\leq_{HN(d)}$ $poly$(**LONELY**) over any field F of characteristic 2.*
*(d) $poly$(**ITERATION**) $\not\leq_{HN(d)}$ $poly$(**LONELY**) over any field F of characteristic 2.*

Proof. [2, 25] prove that $poly$(**PIGEON**) requires $\Omega(N)$-degree Nullstellensatz refutations over any field. [10, 7] prove the same for $poly$(**ITERATION**) (they call the principle "housesitting").

On the other hand, $poly$(**OntoPIGEON**) has constant-degree Nullstellensatz refutations over any field. We have the following polynomials (let X_{ij} say that pigeon i maps to hole j and let Y_{ij} say that hole i maps to pigeon j for $0 \leq i,j < N$):
(i) $(\sum_{j=0}^{N-1} X_{ij}) - 1$ for all i
(ii) $(\sum_{j=0}^{N-1} Y_{ij}) - 1$ for all $i \neq 0$
(iii) X_{i0} for all i
(iv) $X_{ij}(1 - Y_{ji})$ for any i,j
(v) $Y_{ij}(1 - X_{ji})$ for any i,j
(vi) $X_{ij}X_{ij'}$ for any $i,j \neq j'$
Begin by converting each Y_{ij} in (ii) to X_{ji} using (iv) and (v). Now sum up all polynomials in (i) and subtract all polynomials in (ii). What remains is $(\sum_{i=0}^{N} X_{i0}) + 1$. Now we can simply cancel each X_{i0} using (iii).

Finally, $poly(\textbf{LONELY})$ has constant-degree Nullstellensatz refutations over characteristic 2. We have the following polynomials (let X_{ij} say that node i maps to node j for $0 \leq i, j < N$):
(i) $X_{ij} - X_{ij}X_{ji}$ for all $i \neq j$
(ii) X_{ii} for all $i \neq 0$
(iii) $1 - X_{00}$
(iv) $\sum_{j=0}^{N-1} X_{ij}) - 1$ for any i
(v) $X_{ij}X_{ij'}$ for any $i, j \neq j'$
Begin by summing up all polynomials in (i), (ii) and (iv): this yields $(\sum_{i=1}^{N-1} X_{0i}) + 1$. If we add $X_{0j}X_{00} + X_{0j}(1 - X_{00})$ to this, we get simply 1. □

6. Search Problem Separations

Many of the separations follow directly from Theorems 10 and 13, and Lemmas 14 and 16:

Theorem 17. *The following separations hold (G is any generic oracle):*
(a) ([2]) **PIGEON** $\not\leq_m$ **LONELY** *and* $\textbf{PPP}^G \not\subseteq \textbf{PPA}^G$
(b) ([2]) **LONELY** $\not\leq_m$ **PIGEON** *and* $\textbf{PPA}^G \not\subseteq \textbf{PPP}^G$
(c) **PIGEON** $\not\leq_m$ **WeakPIGEON**
(d) ([2]) **PIGEON** $\not\leq_m$ **OntoPIGEON** *and* $\textbf{PPP}^G \not\subseteq \textbf{PPAD}^G$
(e) **ITERATION** $\not\leq_m$ **LONELY** *and* $\textbf{PLS}^G \not\subseteq \textbf{PPA}^G$
(f) [19] **LONELY** $\not\leq_m$ **ITERATION** *and* $\textbf{PPA}^G \not\subseteq \textbf{PLS}^G$
(g) [19] **PIGEON** $\not\leq_m$ **ITERATION** *and* $\textbf{PPP}^G \not\subseteq \textbf{PLS}^G$

As usual, the oracle separations of the search classes follow from the type-2 separations by Theorem 3.

To (almost) complete the characterization of **PLS**, we prove a slightly weaker separation of **ITERATION** from **PIGEON**:

Definition 18. *We say that Q_1 is nicely reducible to Q_2 if $Q_1 \leq_m Q_2$ and, any instance of Q_1 which contains exactly one solution is reduced to an instance of Q_2 that contains exactly one solution.*

Note that all common examples of reductions are nice reductions. In fact, they almost always preserve the number of solutions in general. Nice reductions are, ostensibly, much less restricted than what [2] call *strong reductions*.

Lemma 19. *If* **ITERATION** *is nicely reducible to* **PIGEON***, then* **ITERATION** *is reducible to* **OntoPIGEON**.

Proof. Consider **ITERATION** on a structure of size $N = 2^n$, defined by the type-1 function $succ(\cdot)$. The corresponding instance of PIGEON has size $N' = 2^{n'}$, and is defined by the function β. Let $\mathcal{T} = \{T_{n'}^{\beta(p_i)}\}_{i=0}^{N'-1}$ be the decision trees arising from the nice reduction. Consider a path π_1 in $T_{n'}^{\beta(p_i)}$ that maps pigeon p_i to hole h and π_2 in $T_{n'}^{\beta(p_j)}$ that maps pigeon p_j to h. Either π_1 and π_2 are contradictory or they specify enough of the **ITERATION** instance so that a polytime machine M_2 can find the solution. The only way π_1 and π_2 can so determine a solution is if one of them (wlog π_1) contains a solution or contains a node v whose index is at least $N - poly(n)$ and such that $succ(v) > v$. In the latter case, we can extend P_1 by $poly(n)$ queries so that it contains a solution. Now prune all branches of $\mathcal{T}$ that contain a solution to the **ITERATION** instance. At this point, given any paths π_1, π_2 that represent a collision of pigeons are contradictory. Lemma 4 of [2] describes how to build a forest of trees $\mathcal{H} = \{H^{\beta(h_i)}\}_{i=1}^{N'-1}$ such that each tree has height at most polynomial in n and $H^{\beta(h_i)}$ determines which pigeon, if any, maps to hole h_i. If we find that no pigeon maps to hole h_i, we label the leaf by pigeon 0.

We now have the appropriate objects, namely $\mathcal{T}$ and $\mathcal{H}$, to pass to an oracle for **OntoPIGEON**. This oracle will return (1) pigeons p_i and p_j that collide, (2) a pigeon p_i that maps to hole h_0, (3) a pigeon p_i that maps to hole h_k, but hole h_k maps to pigeon p_j, or (4) a hole h_i that maps to pigeon p_k, but p_k maps to hole h_j. Cases (1) and (2) have nothing to do with $\mathcal{H}$, so we can find a solution to **ITERATION** by the correctness of $\mathcal{T}$. In case (3), it must be that p_i and p_j collide under $\mathcal{T}$, so again we can find a solution to **ITERATION**. Finally, case (4) can arise only when $k = 0$ and h_i is left empty by $\mathcal{T}$. Assume that the pertinent path, π, in tree $H^{\beta(h_i)}$ does not reveal a solution to the **ITERATION** instance, otherwise we are done. Create an instance of **ITERATION** that is consistent with π and contains only one solution as follows: let v be the node (if there is one) such that, according to π, $succ(v) = u > v$ and u is maximal. Note that $u \neq N - 1$, since that would constitute a solution. For every $w < u$ such that π does not determine $succ(w)$, let $succ(w) = u$. For every undetermined $w > u$, let $succ(w) = w$. Finally, let $succ(u) = N - 1$. This instance of **ITERATION** has the unique solution u. Since $\mathcal{T}$ is a nice reduction, the corresponding instance of **PIGEON** has exactly one solution. Hence, there should be no hole, except perhaps h_0, that remains empty. Therefore $H^{\beta(h_i)}$ must have been incorrect, so $\mathcal{T}$ must have been incorrect. □

Theorem 20. **ITERATION** *is not nicely reducible to* **PIGEON**.

Proof. This follows from Lemmas 19 and 16, and Theorem 13. □

7. A Separation Criterion for PLS

We now present a sufficient condition for separating a search Q from **ITERATION**. The condition generalizes the main result of [19] (presented as Theorem 5 of this paper) and also the relative separations from the *iteration problem* in [9].

Theorem 21. *Let Φ be an $\exists$-sentence over a language L all of whose built-in symbols are constants. If Φ fails in an infinite structure, then*

$$Q_\Phi \not\leq_m \textbf{ITERATION}.$$

Proof. For simplicity, we fix the language L to be $L = \{0, \alpha\}$, where α is a unary function, and assume that Φ is an $\exists$-sentence over L of the form $(\exists x)\phi(x)$. The case with arbitrary language and arbitrary $\exists$-sentence is analogous to the current case. Let $\mathcal{K} = (K, \alpha_K)$ be an infinite structure in which Φ fails.

Recall that a partial function $\rho_n : V_n \mapsto V_n$ is called a *restriction*. Let $\rho = \{\rho_n\}_n$ be a family of restrictions. The size of restriction ρ_n is $|dom(\rho_n)|$ and is written $|\rho_n|$. We say that $\{\rho_n\}_n$ is a *polysize family* if $|\rho_n| \in n^{O(1)}$. We say that ρ_n is *safe for* Φ if there exists a one-one mapping $h : V_n \mapsto K$ such that $\rho_n(v) = u$ implies $\alpha_K(h(v)) = h(u)$. Note that, if ρ_n is safe for Φ, then ρ_n does not contain a solution for Q_Φ.

It is not hard to prove that, if ρ_n is a safe restriction and $m + |\rho_n| << N$, then we can answer m queries to α consistently with ρ_n so that ρ_n augmented with the answers is still safe. We call this the *safety property*, and state it more formally as follows: If $\rho = \{\rho_n\}_n$ is a polysize family of safe restrictions, and if $\{T_n\}_n$ is a family of search trees of height polylogarithmic in N, then, for all sufficiently large n, T_n contains a path P such that ρ_n and π_P are consistent and $\rho_n \cup \pi_P$ is safe for Φ. The safety property has been implicit in separation proofs in [5, 15, 2, 9, 19].

Now assume for the sake of contradiction that $Q_\Phi \leq_m$ **ITERATION**. Since **ITERATION** has the instance extension property, we assume without loss of generality that the reduction M on $(\alpha, 1^n)$ composes a query $q = (\beta, 1^m)$ to **ITERATION** without asking any queries to α.

Claim 22. *There exists a polysize family $\{\rho_n\}_n$ of restrictions such that, for sufficiently large n, the following hold: (1) ρ_n is safe for Φ; and (2) ρ_n contains the answers to all the queries to α and* **ITERATION** *made by M on $(1^n, \alpha)$.*

Suppose Claim 22 holds and consider M on $(\alpha, 1^n)$ for n sufficiently large. We answer all the queries to α and **ITERATION** according to ρ_n asserted to exist by the Claim. At the end of its computation, M needs to output some v as a solution for Q_Φ on this instance, although no solution for Q_Φ has been specified. Hence, after M outputs some v, we can construct a total extension α of ρ_n so that $v \notin Q_\Phi(\alpha, 1^n)$. This completes the proof of Theorem 21 from Claim 22.

It remains to prove Claim 22. Fix n sufficiently large and let $q = (\beta, 1^m)$ be the query that M asks to **ITERATION**. We want to construct a safe restriction μ_1 so that a solution for **ITERATION**$(\beta, 1^m)$ is specified. Recall that, for each $x \in V_m$, $T_m^{\beta(x)}$ is the search tree corresponding to the computations of $\beta(x)$; we denote it as $B(x)$. We say a path P of $B(x)$ is *safe* if the corresponding restriction π_P is safe. For each $x \in V_m$, let $Safe_B(x)$ be the set of all good paths of $B(x)$. Because of the safety property, $Safe_B(x)$ is nonempty for every $x \in V_m$. There are three cases to consider:

Case (i): $Safe_B(0^m)$ contains a path P with leaf label $\beta(0^m) = 0^m$. This path defines a solution for **ITERATION**. We give the solution to M and set $\mu_1 := \pi_P$.

Case (ii): For some $x \in V_m$, $Safe_B(x)$ contains a path P with leaf label $\beta(x) = y$ for some $y < x$. This path also defines a solution for **ITERATION**, so we give the solution to M and set $\mu_1 := \pi_P$.

Case (iii): The above two cases do not hold. Since the first case does not hold, every path in $Safe_B(0^m)$ corresponds to a computation of M_β with $\beta(0^m) > 0^m$. Similarly, since the second case does not hold, every path in $Safe_B(1^m)$ leads to $\beta(1^m) = 1^m$. Hence, by the least number principle, there exists $x \in V_m$ such that (1) $Safe_B(x)$ contains a path P that leads to $\beta(x) = y$ for some $y > x$; and (2) for all $z > x$, every path in $Safe_B(z)$ leads to $\beta(z) = z$. Let $Good_B(y)$ as the set of paths P' of $B(y)$ such that $\pi_{P'}$ is consistent with π_P and $\pi_P \cup \pi_{P'}$ is safe for Φ. By the safety property, $Good_B(y)$ is not empty. Let P^* be any path in $Good_B(y)$. Set μ_1 to be $\pi_{P'} \cup \pi_{P^*}$ and return x to M as a solution for its **ITERATION**-query. Note that x is a solution because $\beta(x) = y$ and $\beta(y) = y$. This concludes the construction of μ_1.

Let w be the answer to the query to **ITERATION** that is constructed as above. Recall that $T_n^{M_2}(q, w)$ is the search tree encoding all possible computations of M_2 in this case. By the safety property, there exists a path R in $T_n^{M_2}(q, w)$ such that π_R and μ_1 are consistent and $\pi_R \cup \mu_1$ is safe. Setting $\rho_n := \pi_R \cup \mu_1$ makes Claim 22 hold. ☐

Note that the conclusion of Theorem 21 implies that $C(Q)^G \nsubseteq \textbf{PLS}^G$ for any generic oracle G by Theorem 3.

The reader may be familiar with the following result of Krajicek (Theorem 11.3.1 of [15]):

Theorem 23. *[15] Let Φ be a $\exists\forall$-sentence over a relational language L without $\leq$. If Φ fails in an infinite structure, then the type-2 problem Q_Φ is not in type-2* $\textbf{FP}^{\textbf{NP}}$.

Theorems 21 and 23 are incomparable. Since **ITERATION** is in type-2 $\textbf{FP}^{\textbf{NP}}$ trivially, the consequence of Theorem 23 is stronger than that of our Theorem 21. However, it does not apply to search problems defined by $\exists$-sentences (which are also $\exists\forall$-sentences) over functional languages, which is the scope of our Theorem 21. For example, Theorem 23 does not say anything about the complexity of **PIGEON**, since the **PIGEON** principle is not over a relational language. In fact, **PIGEON** is in type-2 $\textbf{FP}^{\textbf{NP}}$ trivially: binary search asking 'does there exist $v > k$ witnessing **PIGEON**?' for various k yields a solution in polynomial-time.

8. Concluding Remarks and Open Problems

We have obtained a number of search problem separations from proof complexity separations and our Theorems 10 and 13. Note that our proofs of these separations do not depend on the fact that the substitution instance of $CNF(Q_\Psi)$ and $poly(Q_\Psi)$ are uniformly generated by a Turing machine that reduces Q_Φ to Q_Ψ. Hence, all the search problem separations in this paper hold to exclude reductions by nonuniform poly-size circuits. The same is true for the separations obtained in [2, 19].

One of the properties of the two proof systems we use is that the objects they deal with a fairly expressive. What exactly are the properties of a proof system $\mathcal{P}$ necessary to prove an analog of Theorems 10 and 13? We know that no such theorem can be proven for the cutting planes proof system since the **CLIQUE − OR − COLORING** search problem is reducible to **PIGEON**, but there is no short cutting planes reduction between the corresponding CNFs ([24]).

All the separations we obtained in this paper are with respect to many-one reducibility. Since all the known separations from [2, 19] are known to hold with respect to Turing reducibility, it is an interesting open problem to see if this stronger separation is obtainable directly from proof complexity separation.

We made progress toward resolving the relative complexity of **PLS** by showing **ITERATION** $\not\leq_m$ **LONELY** and **ITERATION** is not nicely reducible to **PIGEON**. We are interested in knowing whether **ITERATION** is many-one reducible to **PIGEON** or not, which still remains open. One difficulty is that the iteration principle is easy for almost all proof systems (except for Nullstellensatz, for which the hardness of the iteration principle allowed us to prove **ITERATION** $\not\leq_m$ **LONELY**) and the pigeonhole principle is hard for almost all proof systems.

From Theorem 13 and the fact that $poly$(**LONELY**) has constant-degree Nullstellensatz refutations, it follows that the totality of every **PPA** problem has low-degree Nullstellensatz proofs. This indicates that the fixed point theorems of Brouwer, Nash, and Kakutani, whose corresponding search problems are in **PPA**, have low-complexity proofs.

Theorems 10 and 13 constructs propositional refutations from reductions. Does the converse hold? Is it true that if the translation of a search problem has a simple LK or Nullstellensatz refutation, then the search problem is reducible to, say, **ITERATION** (which is easy for LK) or **LONELY** (which is easy for Nullstellensatz)?

The theories of bounded arithmetic are introduced by Buss in [5] as fragments of Peano Arithmetic with bounds on their reasoning power. Bounded arithmetic is closely related to computational complexity and proof complexity, and our results connecting these two areas naturally have some consequences on bounded arithmetic as well. For the definitions and relevant results, we refer the reader to [5, 15, 6]. For an $\exists$-sentence Φ, we denote by $\Phi^{<a}$ the formula obtained by bounding all existential quantifiers in Φ by a free variable a.

Theorem 24. *Let Φ be an $\exists$-sentence over an arbitrary language L. If the relativized bounded arithmetic theory $S_2(L)$ proves $\forall x \Phi^{<x}$, then* **PIGEON** $\not\leq_m$ Q_Ψ *and* **LONELY** $\not\leq_m$ Q_Ψ*. In fact, $Q_\Psi \not\leq_m Q_\Phi$ for any Φ such that $CNF(\Phi)$ requires exponential-size refutations in any bounded-depth LK system.*

Proof. The idea is that, if $S_2(L)$ proves $\forall x \Phi^{<x}$, then from the proof we can construct quasi-polysize bounded-depth LK refutations of $CNF(Q_\Phi)$ ([22, 15]). From Theorem 10 it follows that, if $Q_\Psi \leq_m Q_\Phi$, then $CNF(\Psi)$ has subexponential-size LK refutations, which contradicts the assumption. □

Our Theorem 21 implies the following independence criterion for the relativized S_2^2 by Riis [26] in a similar way Krajicek's theorem (Theorem 23) in [15] implies it.

Theorem 25. *[26] Let L be a language that is disjoint with the language of bounded arithmetic, and let $\Phi = \exists x \phi(x)$ be a sentence of arbitrary quantifier-complexity. If Φ fails in an infinite structure, then the relativized bounded arithmetic theory $S_2^2(L)$ does not prove $\Phi^{<a}$.*

Proof. Krajicek has a proof of this theorem based on Theorem 23 in [15]. Since our proof is similar to his, we only sketch the idea. Let Φ be of the form $\exists x_1 \forall y_1 \ldots \exists x_k \forall y_k \phi(x_1, y_1, \ldots, x_k, y_k)$, with ϕ quantifier-free. Define a herbrandization Φ_H of Φ as

$$\exists x_1 \exists x_2 \ldots \exists x_k \phi(x_1, f_1(a, x_1), \ldots, x_k, f_k(a, x_1, \ldots, x_k)),$$

where $f_1, \ldots, f_k$ are new functions. Let $\mathcal{K}$ be an infinite structure in which Φ is false. By defining $f_1, \ldots, f_k$ appropriately, $\mathcal{K}$ can be extended to $\mathcal{K}'$ in which Φ_H fails; thus, Q_{Φ_H} is not reducible to **ITERATION** by Theorem 21.

Let $L' = L \cup \{f_1, \ldots, f_k\}$. Since **ITERATION** characterizes the $\Sigma_1^b(L')$-consequences of $S_2^2(L')$ ([9]), $S_2^2(L')$ does not prove

$$\exists x_1 < a \exists x_2 < a \ldots \exists x_k < a$$
$$[f_1(a, x_1) < a \wedge \ldots \wedge f_k(a, x_1, \ldots, x_k) < a \supset$$
$$\phi(x_1, f_1(a, x_1), \ldots, x_k, f_k(a, x_1, \ldots, x_k))].$$

Let M be a model of $S_2^2(L')$ in which the above formula fails. It is not hard to see that $\Phi^{<a}$ fails in this model. □

Acknowledgment. The second author wishes to thank S. Riis for his suggestion that his independence criterion may follow from the main result of [19], which led to Theorems 21 and 25 of this paper. The first author wishes to thank Toni Pitassi for some helpful discussions.

References

[1] T. P. Baker, J. Gill, and R. Solovay. Relativizations of the *P=?NP* question. *SIAM Journal on Computing*, 4:431–442, 1975.

[2] P. W. 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.

[3] P. W. Beame and T. Pitassi. An exponential separation between the parity principle and the pigeonhole principle. *Annals of Pure and Applied Logic*, 80:197–222, 1996.

[4] M. Blum and R. Impagliazzo. Generic oracles and oracle classes. In *28th Annual Symposium on Foundations of Computer Science*, pages 118–126, Los Angeles, CA, Oct. 1987. IEEE.

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

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

[7] S. R. Buss. Lower bounds on Nullstellensatz proofs via designs. In P. W. Beame and S. R. Buss, editors, *Proof Complexity and Feasible Arithmetics*, volume 39 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 59–71. American Mathematical Society, 1998.

[8] 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.

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

[10] M. Clegg, J. Edmonds, and R. Impagliazzo. Using the Gröbner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 174–183, Philadelphia, PA, May 1996.

[11] 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.

[12] L. Fortnow and M. Sipser. Are there interactive proofs for coNP languages? *Information Processing Letters*, 28:249–251, 1988.

[13] Hartmanis, Chang, Chari, Ranjan, and Rohatgi. Relativization: A revisionistic retrospective. *BEATCS: Bulletin of the European Association for Theoretical Computer Science*, 47, 1992.

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

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

[16] J. Krajíček, P. Pudlák, and A. Woods. Exponential lower bounds to the size of bounded depth Frege proofs of the pigeonhole principle. *Random Structures and Algorithms*, 7(1), 1995.

[17] A. Maciel, T. Pitassi, and A. R. Woods. A new proof of the weak pigeonhole principle. In ACM, editor, *Proceedings of the thirty second annual ACM Symposium on Theory of Computing: Portland, Oregon, May 21–23, [2000]*, pages 368–377, New York, NY, USA, 2000. ACM Press.

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

[19] T. Morioka. Classification of Search Problems and Their Definability in Bounded Arithmetic. Master's thesis, University of Toronto, 2001. Also available as ECCC technical report TR01-82 at http://www.eccc.uni-trier.de.

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

[21] 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.

[22] J. Paris and A. Wilkie. Counting problems in bounded arithmetic. In *Methods in Mathematical Logic: Proceedings of the 6th Latin American Symposium on Mathematical Logic 1983*, volume 1130 of *Lecture notes in Mathematics*, pages 317–340, Berlin, 1985. Springer-Verlag.

[23] T. Pitassi, P. W. Beame, and R. Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational Complexity*, 3(2):97–140, 1993.

[24] P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, Sept. 1997.

[25] A. A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, 1998.

[26] S. Riis. Making infinite structures finite in models of second order bounded arithmetic. In P. Clote and J. Krajicek, editors, *Arithmetic, Proof Theory, and Computational Complexity*, pages 289–319. Oxford University Press, 1993.

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

[28] 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.