Annals of Pure and Applied Logic 163 (2012) 1163–1182

Contents lists available at SciVerse ScienceDirect

Annals of Pure and Applied Logic

journal homepage: www.elsevier.com/locate/apal

Propositional proofs and reductions between NP search problems

Samuel R. Buss*, Alan S. Johnson

Department of Mathematics, University of California, San Diego, La Jolla, CA 92093-0112, United States

ARTICLE INFO

Article history:
Received 9 January 2011
Received in revised form 9 July 2011
Accepted 9 July 2011
Available online 3 March 2012
Communicated by Jan Krajicek

MSC:
03B05
03F07
68Q15

Keywords:
Search problems
Nullstellensatz
Propositional proofs
Polynomial time reducibility

ABSTRACT

Total NP search problems (TFNP problems) typically have their totality guaranteed by some combinatorial property. This paper proves that, if there is a polynomial time Turing reduction between two TFNP problems, then there are quasipolynomial size, polylogarithmic height, constant depth, free-cut free propositional (Frege) proofs of the combinatorial property associated with the first TFNP problem from the property associated with the second problem. In addition, they have Nullstellensatz derivations of polylogarithmic degree. These results extend the previous work of Buresh-Oppenheim and Morioka first by applying to Turing reductions in place of many-one reductions and second by giving tight bounds on the height of the Frege proofs. As a corollary, PLS-complete problems are not polynomial time Turing reducible to PPA problems relative to a generic oracle. We establish a converse construction as well, by showing that a polynomial time Turing reduction can be obtained from a family of quasipolynomial size, polylogarithmic depth, propositional proofs which are based on "decision tree" substitutions. This establishes the optimality of our constructions, and partially resolves an open question posed by Buresh-Oppenheim and Morioka.

We observe that the classes PPA, PPAD, PPADS, and PLS are closed under Turing reductions, and give an example of a TFNP class that is not closed under Turing reductions.

1. Introduction

Total NP search problems, called TFNP problems as an acronym for total function NP problems, are multi-valued functions (relations) which are of polynomial growth rate, are total, and have graph in NP. There are a number of commonly studied classes of total NP search problems. In particular, Johnson et al. [17] introduced the class PLS of polynomial local search problems, and Papadimitriou [22] introduced a wider range of classes, notably the parity principles PPP, PPA, and PPAD. These are all subclasses of the total NP multifunctions (TFNP) as defined by Megiddo and Papadimitriou [19].

A key property of TFNP functions is that they are total; namely, a solution (or answer) exists for all possible inputs. In most cases, the totality follows from a combinatorial property. For example, the class PPP contains search problems that are total by virtue of the pigeonhole principle. Similarly, the class PPA is based on the combinatorial principle that any graph of degree two or less has an even number of nodes of degree one. Another example is the class PLS, which is based on the existence of a local minimum for search problems which have a local search heuristic that respects a positive integer valued cost function.

Many of the interesting TFNP classes have multiple equivalent characterizations. For example, PLS can equivalently be characterized in terms of finding a local optimum for the Lin–Kernighan heuristic for the traveling salesman problem, or alternately in terms of solving the problem, FLIP, of finding an input to a Boolean circuit that provides locally minimal output value [22]. The class PPAD is defined in terms of the principle that a *directed* graph in which all nodes have total degree ≤ 2

* Corresponding author.
E-mail addresses: sbuss@math.ucsd.edu (S.R. Buss), asj002@math.ucsd.edu (A.S. Johnson).

0168-0072/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.apal.2012.01.015

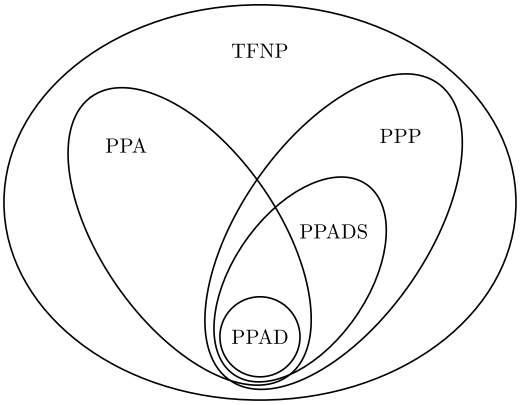

Fig. 1. The known relationships for some TFNP search classes. The class PLS (not shown) does not contain PPAD and is not contained in PPA; it is open whether PLS is contained in PPADS or PPP. This figure is adapted from [1].

cannot contain exactly one node of total degree 1. This is known to be equivalent to a problem SPERNER based on Sperner's lemma about the existence of a panchromatic complex in a triangulation of a trichromatic triangle [22]. Surprisingly, the class PPAD can equivalently be defined in terms of the problem NASH of finding a Nash equilibrium; this was established for three or more players by [14], and for two players by [11].

The various TFNP classes were originally defined explicitly in terms of their associated combinatorial principle; see [22]. For instance, PPP was defined as the class of functions which could be computed by finding inputs for which a polynomial time Turing machine M fails to compute a counterexample to the pigeonhole principle. Beame et al. [1] formulated these definitions somewhat differently. For example, for PPP they defined a "generic" or "type-2" version of the pigeonhole principle. They then defined the class PPP to be the set of total NP problems which are many-one reducible to the generic PPP problem. (Henceforth, "reducible" or "reduction" always means in polynomial time. We define all these concepts more precisely below.) Of course, this is essentially equivalent to the original definition of [22]. But [1] also considered a (potentially larger) version of the class PPP, namely the set of total NP search problems that are *Turing* reducible to the generic PPP problem.

The exact computational complexity of these classes is unknown. For instance, if $\mathrm{P} = \mathrm{NP}$, then all TFNP problems are solvable in polynomial time. Thus, the different TFNP classes cannot be separated without solving the $\mathrm{P} = \mathrm{NP}$ problem. Conversely, however, various inclusions between TFNP classes are known to hold. Some of these are shown in Fig. 1.

It is commonly conjectured that no other reductions exist between the classes shown in Fig. 1. This cannot be established without showing $\mathrm{P} \neq \mathrm{NP}$, but one can instead consider oracle separations. It was for this purpose that Beame et al. [1] reformulated the TFNP classes in terms of type-2 search problems. By [13], a type-2 search problem Q_1 is (many-one or Turing) reducible another type-2 search problem Q_2 if and only if Q_1 is similarly reducible to Q_2 relative to a generic oracle.

In the type-2/generic oracle setting, [1] proved that there are no reductions between the classes pictured in Fig. 1 beyond those already shown. Specifically, they proved that $\mathrm{PPADS}^G \not\leq_{\mathrm{T}} \mathrm{PPA}^G$ and $\mathrm{PPP}^G \not\leq_{\mathrm{T}} \mathrm{PPADS}^G$ and $\mathrm{PPA}^G \not\leq_{\mathrm{T}} \mathrm{PPP}^G$, where $\leq_{\mathrm{T}}$ denotes Turing reducibility and the superscript G indicates the class relativized by a generic oracle.

Morioka [20] extended the separation results of [1] by showing that PPAD^G is not Turing reducible to PLS^G.[1] This implies also that none of PPA^G, PPADS^G, or PPP^G is Turing reducible to PLS^G. This left open the question of whether there are any reductions in the reverse directions. As a partial answer to this question, Buresh-Oppenheim and Morioka showed that PLS^G is not many-one reducible to PPA^G. An interesting aspect of their proof is that, building on constructions from [2,1], it uses known separation results from proof complexity. Broadly speaking, the idea is that type-2 TFNP search problems can be coded as purely existential, first-order formulas. From this, particular instances of TFNP search problems become constant depth propositional formulas. Then, polynomial time reductions between TFNP search problems can be translated into constant depth propositional proofs. In particular, [7,21] proved that, if there is a many-one reduction between two TFNP search problems expressed by first-order existential formulas, then there are quasipolynomial size, constant depth Frege (LK) proofs relating the underlying combinatorial principles which justify the totality properties of the TFNP search problems. In addition, they established similar results for polylogarithmic degree Nullstellensatz proofs in place of constant depth propositional proofs. From these results, previous known separations in proof complexity implied that there can be no polynomial time many-one reductions between the corresponding TFNP search problems.

The main goal of the present paper is to extend the constructions of Buresh-Oppenheim and Morioka to apply to Turing reductions instead of only to many-one reductions. We are able to carry this out completely. In particular, Theorem 3.3 establishes that, if a type-2 TFNP problem Q_1 is Turing reducible to another type-2 TFNP problem Q_2, and if the totality of

[1] Another way to prove this is by noting that Theorem 21 of [7] (also reported as Theorem 4.1 in [21]) holds for Turing reducibility as well as for many-one reducibility. This theorem states that any existential first-order sentence without built-in function or relation symbols which is true in some infinite structure cannot be reduced to PLS.

Q_1 and Q_2 are expressible by purely existential first-order formulas, then there are quasipolynomial size, polylogarithmic height, constant depth, free-cut free LK proofs of the propositional formulas expressing the totality of Q_1 from those expressing the totality of Q_2. Theorem 3.5 sharpens this construction by limiting the kinds of substitution instances of formulas expressing the totality of Q_2 that may be used in the propositional proofs. It allows only "decision tree" substitution instances to be used: decision tree substitutions are ones that can be computed by low-depth decision trees and thus are in non-uniform polynomial time.

Buresh-Oppenheim and Morioka further show that a many-one reduction gives rise to low-degree Nullstellensatz refutations. Theorem 4.5 extends this by showing that it is sufficient to have a Turing reduction instead of a many-one reduction. As a corollary, we establish the new result that there is no *Turing* reduction of PLS^G to PPA^G. The results of Section 6 (discussed below) give an alternate proof of this corollary; indeed, Theorem 6.1 shows that many-one and Turing reducibility are *equivalent* for reductions to several classes, including to PPA^G. It is still open, however, whether PLS^G is Turing reducible to PPP^G.

As an additional new result, we can partially reverse our constructions of LK proofs. Namely, we prove that, if there are sufficiently uniform, quasipolynomial size, polylogarithmic height, cut free, LK proofs of the propositional formulas expressing the totality of Q_1 from decision tree substitutions of the formulas expressing the totality of Q_2, then there is a Turing reduction from Q_1 to Q_2. In effect, this says that constant depth, cut free reducibility in the propositional proof setting is the exact (non-uniform) analogue of Turing reducibility between TFNP search problems. The condition that there are no cuts can be somewhat relaxed to allow cuts on polylogarithmic size formulas.

We have been unable to establish an analogous reversal for polylogarithmic degree Nullstellensatz proofs. In fact, we would not expect to be able to do so since it would entail putting restrictions on Nullstellensatz refutations that would make them identical to quasipolynomial size, polylogarithmic height, free-cut free Frege proofs.

The outline of the rest of the paper is as follows. Section 2 includes the detailed definitions of type-2 TFNP and related concepts, as well as some conventions that will be used throughout the paper. Section 3 establishes the translation from a Turing reduction between TFNP problems to constant depth, free-cut free LK proofs. The main results of this section are Theorems 3.3 and 3.5; we refer to these as the "forward direction". Section 4 proves the forward direction in the Nullstellensatz setting. Section 5 introduces the notion of "decision tree substitution" and then proves the reverse direction for LK proofs; namely, that a Turing reduction can be obtained from sufficiently restricted LK proofs.

In Section 6, we consider more carefully the distinction between many-one and Turing reducibility. First, we prove that PPA, PPAD, PPADS, and PLS are all closed under Turing reductions. On the other hand, we conjecture that PPP is not closed under Turing reductions. We also give an example of two TFNP search problems Q_1 and Q_2 such that Q_1 is Turing reducible to Q_2 but not many-one reducible. Q_2 is the problem of searching for a solution to either a PPP or a PPA problem, and Q_1 is the problem of searching for solutions to both a PPP and a PPA problem.

The relation between many-one and Turing reductions has been studied by Hanika in [15], where it is shown there are problems A and B such that A is Turing reducible to B, but if A is many-one reducible to B then $\mathrm{P} = \mathrm{NP}$. This result does not fit into our context, since A and B are not in TFNP (in particular solutions to A and B are not polynomial time verifiable unless $\mathrm{P} = \mathrm{NP}$).

2. Preliminaries

An NP search problem is any problem for which solutions are recognizable in polynomial time. It is total provided that, for each input, there is at least one solution.

Definition 2.1. Let $R(x, y)$ be a polynomial time predicate such that $R(x, y)$ implies $|y| \leq p(|x|)$, for some polynomial p. Problem Q_R is as follows: Given x, find y such that $R(x, y)$. This is called an NP *search problem*. Let $Q_R(x)$ be the set $\{y|R(x, y)\}$. Problem Q_R is *total* if, for all x, $Q_R(x) \neq \emptyset$. TFNP is the set of all total NP search problems.

By convention $x, y, u, v, \ldots$ denote binary strings (type-0 objects); we only consider binary strings, so "string" always means "binary string". As defined above, TFNP problems take only a string as input; we call these "unrelativized" or "type-1" because they take as arguments type-0 objects. A Turing machine that only takes string inputs is also called a type-1 Turing machine. It is useful to work with a "relativized" version of TFNP, denoted TFNP^2. The superscript 2 indicates that the class contains type-2 problems, namely, problems that also take functions (type-1 objects) as inputs. Similarly, a Turing machine that takes strings and functions as inputs is called a type-2 Turing machine.

We shall only consider type-1 inputs f whose arguments are strings of (the same) length n, and output strings of length n. Let U_n denote the set $\{0, 1\}^n$ of strings of length n. The integer n serves as a size parameter: a type-2 Turing machine M with type-0 inputs (strings) and type-1 inputs (functions) will be invoked with type-0 inputs all of length n and constrained to only query its functions with values from U_n. Machine M is said to run in *polynomial time* if there is a polynomial p such that, for all type-0 inputs of length n and all type-1 functions f as above, M runs in time $\leq p(n)$, where calls to the function oracles count as a single time step.

Definition 2.2. A type-2 search problem Q of input signature $(\ell, k_1, \ldots, k_r)$ and *output arity* s assigns a set $Q(f_1, \ldots, f_r, x_1, \ldots, x_\ell) \subseteq U_n^s$ to each choice of functions $f_i : U_n^{k_i} \to U_n$ and strings $x_j \in U_n$. It is required that $\ell \geq 1, r \geq 0$, and $k_i \geq 1$. Then Q is the following search problem: Given $f_1, \ldots, f_r$ and $x_1, \ldots, x_\ell$, find $y_1, \ldots, y_s$ such that $\vec{y} \in Q(\vec{f}, \vec{x})$.

If the property $\vec{y} \in Q(\vec{f}, \vec{x})$ is decided by a type-2 polynomial time Turing machine, then Q is a *type-2* NP *search problem*. If, in addition, $Q(\vec{f}, \vec{x}) \neq \emptyset$ for all $n > 0$, all $f_i : U_n^{k_i} \to U_n$, and all $x_j \in U_n$, then Q is *total*. The set of total type-2 NP search problems is denoted TFNP^2.

The reason for requiring $\ell \geq 1$ is so that there is at least one string input, as otherwise M would have no way of knowing the value of n. If there is no need for a particular string input, the input 0^n can be used as a placeholder.

Our definition of type-2 NP search problems effectively requires the search problems to have linear growth rate; that is, the length of the output $\vec{y}$ is linear in the length of the string inputs. It is more common to allow NP search problems to have polynomial growth rate. We use the linear growth rate convention, however, since it fits the formalizations into first-order formulas better. In addition, the restriction to linear growth rate makes no essential difference, since polynomial growth rate search problems can be simulated by appropriate polynomial padding.

We next define many-one and Turing reductions between two type-2 problems [1]. (Recall our convention that reductions are always assumed to be polynomial time.) These reductions allow a Turing machine to use a search problem Q as a subroutine. The key is to define how a Turing machine M can use a search problem as an oracle; we first give the definition for the case where M is a type-1 Turing machine.

Definition 2.3. A type-2 search problem $Q(\vec{g}, \vec{y})$ is used as an oracle by a type-1 Turing machine M in the following manner: M has special query tapes for the string inputs to Q and for each of the input functions g_i to Q. M presents a query $(g_1, \ldots, g_r, y_1, \ldots, y_\ell)$ to Q by writing $\vec{y}$ on the query tapes for the string inputs and writing a Turing machine description of each function g_i on its designated query tape. Each g_i must be given an explicit polynomial time clock p_i as part of its description. In the next step, M receives an answer $\vec{z}$ to $Q(\vec{g}, \vec{y})$.

Letting m be the common length of the string values y_i, this call to Q counts as $\max_i p_i(m)$ steps for the runtime of M. Machine M runs in *polynomial time relative to* Q provided that M always halts within $p(n)$ steps for some fixed polynomial p, where n is the length of the input to M.

If M is a type-2 instead of type-1 Turing machine, then M may use an oracle $Q(\vec{g}, \vec{y})$ in much the same way; however, now the functions $g_1, \ldots, g_r$ are described by *oracle* Turing machines that are allowed to query the functions input to M. Specifically, suppose that the inputs of the Turing machine M include functions $f_1, \ldots, f_{r'}$, each f_i being k_i'-ary. Then, when M invokes Q with functions $g_1, \ldots, g_r$, each g_i is described by a type-2 Turing machine M_i that has r' function oracles. The runtime of M is defined as before. Note that when a Turing machine g_i invokes a function f_j this counts as a single time step; this reflects the fact that f_j is an oracle and does not have a runtime.

We can now define the notions of many-one and Turing reductions.

Definition 2.4. Let $Q_1(\vec{f}, \vec{x})$ and $Q_2(\vec{g}, \vec{y})$ be type-2 search problems. A type-2 oracle Turing machine M is a *Turing reduction* from Q_1 to Q_2 if for any input $(\vec{f}, \vec{x})$ to Q_1, M outputs some $\vec{z} \in Q_1(\vec{f}, \vec{x})$ in polynomial time relative to Q_2 (and $\vec{f}$). In this case, Q_1 is said to be *Turing reducible* to Q_2, and we write $Q_1 \leq_{\mathrm{T}} Q_2$. If $Q_1 \leq_{\mathrm{T}} Q_2$ and $Q_2 \leq_{\mathrm{T}} Q_1$, then Q_1 and Q_2 are *Turing equivalent*, $Q_1 \equiv_{\mathrm{T}} Q_2$.

If M makes at most one query to Q_2, then M is a *many-one reduction*, and Q_1 is *many-one reducible* to Q_2, written $Q_1 \leq_{\mathrm{m}} Q_2$. If $Q_1 \leq_{\mathrm{m}} Q_2$ and $Q_2 \leq_{\mathrm{m}} Q_1$, then Q_1 and Q_2 are *many-one equivalent*, $Q_1 \equiv_{\mathrm{m}} Q_2$.

The definition of reductions applies immediately also to type-1 search problems Q_1; the only change is that the oracle machine M is a type-1 instead of a type-2 machine. This lets us define the standard classes of (type-1) TFNP problems in terms of reductions to type-2 problems.

Definition 2.5. Let Q_2 be a type-2 search problem. The class $\mathrm{C}_{\mathrm{m}}(Q_2)$, respectively $\mathrm{C}_{\mathrm{T}}(Q_2)$, is the set of type-1 search problems which are many-one, respectively Turing, reducible to Q_2.

With these definitions in place, we now introduce Buresh-Oppenheim and Morioka's method [7,21] of defining TFNP^2 search problems and TFNP classes in terms of first-order formulas. As a example, consider the following formula from [12,10]:

$$0 < f(0) \wedge \forall x[x \leq f(x)] \to \exists x[x < f(x) \wedge f(f(x)) = f(x)]. \tag{1}$$

The formula characterizes the iteration principle, ITER, and thereby the class PLS. It is intended that this formula will be interpreted in the structure U_n with $<$ and $\leq$ corresponding to lexicographic ordering on n bit strings and with the constant symbol 0 denoting the string 0^n. It is clear that (1) is valid in all such structures. We think of this formula as expressing the totality of the type-2 NP search problem ITER: Given 0^n as a size parameter and $f : U_n \to U_n$, find $u \in U_n$ such that of the following holds: (a) $f(0^n) = 0^n$, or (b) $f(u) < u$, or (c) $u < f(u)$ and $f(f(u)) = f(u)$. As is well known, ITER is many-one complete for PLS, and thus PLS $= \mathrm{C}_{\mathrm{m}}(\mathrm{ITER})$. (In fact, also PLS $= \mathrm{C}_{\mathrm{T}}(\mathrm{ITER})$, as we show in Section 6.)

The example of ITER generalizes to other existential sentences valid in U_n. These first-order sentences will generally use both *uninterpreted* function symbols (e.g., the function f above) and *interpreted* constant symbols, relation symbols, and function symbols (e.g., 0, $<$, and $\leq$). The intent is that the interpreted symbols depend only on n and thus have a fixed meaning in U_n, while the uninterpreted symbols will be the type-1 inputs.

We may assume without loss of generality that there are no relation symbols and no uninterpreted constant symbols. Each uninterpreted constant symbol c can be replaced a new unary function symbol f_c, using the term $f_c(0)$ in place of c.

Relation symbols R, whether interpreted or uninterpreted, may be replaced with a new function symbol f_R for the graph of R and then using $f_R(\vec{t}) = 0$ in place of $R(\vec{t})$. This works since, as usual, we disallow empty structures, so $n \geq 1$, and the universe has at least two members. The advantage of removing interpreted relation symbols is that the only atomic formulas are equalities of the form $s = t$ for terms s and t. The equality sign, $=$, always denotes true equality. The purpose of eliminating uninterpreted constant symbols is to avoid the oddity of having 0-ary functions as inputs to a predicate; the type-0 inputs play the same role and could be viewed as 0-ary function symbols. (We could have also eliminated interpreted constant symbols, but this yields no advantage, so we refrain from this.)

Following [7], interpreted constants and function symbols are called “built-in” symbols. The interpretation of built-in symbols depends only on n. Commonly used built-in symbols include 0, 1, $f_\leq$, and $f_<$, and they are interpreted by 0^n, 1^n, and the graphs of $\leq$ and $<$, respectively. Any use of $\leq$ and $<$ is to be understood as shorthand notation for formulas using $f_\leq$ and $f_<$.

Definition 2.6. A language L is *basic* if it is finite and contains only the following symbols: built-in constant symbols and function symbols, equality (=), and the non-built-in function symbols $f_1, \ldots, f_r$.

Definition 2.7. An $\exists$-*formula* is a formula of the form $\exists\vec{x}\phi(\vec{x})$ over a basic language, where ϕ is quantifier free. If $\exists\vec{x}\phi(\vec{x})$ has no free variables, then it is an $\exists$-*sentence*. An $\exists$-sentence is *total* if it is true in all U_n under arbitrary interpretations for the function symbols f_i.

$\exists$-sentences give rise to NP search problems in the obvious way.

Definition 2.8. Let Φ be the $\exists$-sentence $\exists\vec{x}\phi(\vec{x})$. The type-2 search problem Q_Φ is as follows: Given the string 0^n and interpretations for the f_i's, find $\vec{u} \in U_n$ such that $\phi(\vec{u})$ holds.

If Φ is total, then clearly Q_Φ is in TFNP^2. The string input 0^n to Q_Φ serves only as a size parameter.

We already defined PLS above as $\mathrm{C_m}$(ITER). The four classes that are shown in Fig. 1 are defined as follows.

PPAD. Let Φ be the prenex form of the formula

$$g(0) = 0 \wedge f(0) \neq 0 \rightarrow \exists x[x \neq g(f(x)) \vee (x \neq 0 \wedge x \neq f(g(x)))].$$

This gives the *onto pigeonhole principle*, OntoPIGEON $= Q_\Phi$. Note that g acts as the inverse of f. The class PPAD is $\mathrm{C_m}$(OntoPIGEON).

PPADS. Let Φ be the prenex form of the formula

$$g(0) = 0 \wedge f(0) \neq 0 \rightarrow \exists x[x \neq g(f(x))]. \tag{2}$$

Define LeftPIGEON to be Q_Φ. Note that g is a left inverse of f. Then PPADS is $\mathrm{C_m}$(LeftPIGEON).

PPA. Let Φ be the prenex form of the formula

$$f(0) = 0 \rightarrow \exists x[x \neq f(f(x)) \vee (x \neq 0 \wedge x = f(x))].$$

Define LONELY to be Q_Φ. Then LONELY expresses the following parity principle, or mod 2 counting principle. If f pairs elements, and pairs 0 with itself, then there is another node paired with itself. Since the universes U_n have even cardinality, LONELY is total. Let PPA be $\mathrm{C_m}$(LONELY).

PPP. Let Φ be a prenex form of the formula

$$\forall x[f(x) \neq 0] \rightarrow \exists x, \quad y[x \neq y \wedge f(x) = f(y)].$$

This expresses the standard pigeonhole principle. Let PIGEON be Q_Φ and PPP be $\mathrm{C_m}$(PIGEON).

The above definitions of PPAD, PPADS and PPA are slightly different from the previous definitions by [22,1,7]. It is more common to define them in terms of the search problems SourceOrSink, SINK, and LEAF, respectively. It is easy, however, to see that the above definitions are equivalent.

We next establish some definitions and conventions that will simplify the technical details of working with $\exists$-sentences. The first simplification is to require that atomic formulas be “basic”.

Definition 2.9. Let Φ be an $\exists$-sentence $\exists\vec{x}\phi(\vec{x})$. Then Φ is *basic* if $\phi(\vec{x})$ is in disjunctive normal form (DNF) $\bigvee_{j\in J}\phi_j(\vec{x})$, and, for each $j \in J$, $\phi_j(\vec{x})$ is a conjunction of formulas of the form $g(\vec{u}) = v$, $v = w$, or $v \neq w$, where $\vec{u}$, v, w are either variables or constant symbols.

It is easy to see that every $\exists$-sentence is equivalent to a basic formula. Complex terms may be eliminated by introducing new existentially quantified variables: for example, $f(g(x)) = h(y)$ can be replaced with $\exists u, v[g(x) = u \wedge h(y) = v \wedge f(u) = v]$. Formulas of the form $g(\vec{u}) \neq v$ may be replaced with $\exists x[g(\vec{u}) = x \wedge x \neq v]$.

In a formula $g(\vec{u}) = v$, the function g may be either built in or one of the uninterpreted (input) functions f_i. If g is built in, and $\vec{u}, v \in U_n$ are fixed, then $g(\vec{u}) = v$ has a fixed truth value that depends only on n and not on the choice of the f_i's. Similarly, if $v, w \in U_n$ are fixed, then $v = w$ will be either true or false, independently of the choice of the f_i's.

$$\frac{\Gamma, A, B \longrightarrow \Delta}{\Gamma, A \wedge B \longrightarrow \Delta}\ \wedge : \text{left} \qquad \frac{\Gamma \longrightarrow \Delta, A \quad \Gamma \longrightarrow \Delta, B}{\Gamma \longrightarrow \Delta, A \wedge B}\ \wedge : \text{right}$$

$$\frac{\Gamma, A \longrightarrow \Delta \quad \Gamma, B \longrightarrow \Delta}{\Gamma, A \vee B \longrightarrow \Delta}\ \vee : \text{left} \qquad \frac{\Gamma \longrightarrow \Delta, A, B}{\Gamma \longrightarrow \Delta, A \vee B}\ \vee : \text{right}$$

$$\frac{\Gamma \longrightarrow \Delta, A}{\Gamma, \neg A \longrightarrow \Delta}\ \neg : \text{left} \qquad \frac{\Gamma, A \longrightarrow \Delta}{\Gamma \longrightarrow \Delta, \neg A}\ \neg : \text{right}$$

$$\frac{\Gamma \longrightarrow \Delta, A \quad A, \Gamma \longrightarrow \Delta}{\Gamma \longrightarrow \Delta}\ \text{Cut} \qquad \frac{}{p \longrightarrow p}\ \text{Logical Axiom}$$

$$\frac{}{\longrightarrow \top}\ \top \qquad \frac{}{\bot \longrightarrow}\ \bot$$

$$\frac{\Gamma, A, A \longrightarrow \Delta}{\Gamma, A \longrightarrow \Delta}\ \text{Contract:left} \qquad \frac{\Gamma \longrightarrow \Delta, A, A}{\Gamma \longrightarrow \Delta, A}\ \text{Contract:right}$$

$$\frac{\Gamma \longrightarrow \Delta}{\Gamma' \longrightarrow \Delta'}\ \text{Weakening}$$

Fig. 2. Γ' and Δ' are sets of formulas such that $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$.

Our second simplification is to assume that there is only one uninterpreted (non-built-in) function symbol f in the language L of Φ. If there are multiple function symbols $f_1, \dots, f_r$ in Φ, then let $\underline{i}$ be a built-in constant symbol with value equal to the binary expansion of i in all sufficiently large U_n. (We will not consider the case when $i > 2^n$, and it is irrelevant as we are only interested in asymptotics anyway.) Let f have arity equal to one plus the maximum arity of the f_i's. Then we can replace each occurrence of $f_i(\vec{x})$ in Φ with $f(\underline{i}, \vec{x}, \vec{0})$, where $\vec{0}$ represents extra inputs that pad out to the arity of f. Letting Φ' be the resulting formula; we clearly have $Q_\Phi \equiv_{\mathrm{m}} Q_{\Phi'}$.

As an example, applying these simplifications to formula (2) defining LeftPIGEON yields

$$\exists x, y, z, w[(f(\underline{2}, 0){=}w \wedge w{\neq}0) \vee f(\underline{1}, 0){=}0 \vee (f(\underline{1}, x){=}y \wedge f(\underline{2}, y){=}z \wedge z{\neq}x)],$$

where f is now a binary function. The occurrences of $f(t)$ and $g(t)$ in (2) have been replaced by $f(\underline{1}, t)$ and $f(\underline{2}, t)$, respectively, where $\underline{1}$ and $\underline{2}$ are to be interpreted as $0^{n-1}1$ and $0^{n-2}10$ in U_n, respectively. Also, note that the quantifier-free subformula is in disjunctive normal form.

Propositional proofs. Sections 3 and 4 deal with two types of propositional proof systems: constant depth LK (Frege) proofs formalized with the sequent calculus, and the Nullstellensatz proof system. Our conventions for both systems are standard. We presume that the reader is already familiar with these proof systems, but quickly review some terminology.

Our results are not particularly sensitive to the precise formulation used for LK proofs, but we will assume that the rules of inference are as given in Fig. 2. The connectives are $\wedge, \vee, \neg, \top$, and $\bot$. Negations are allowed only on propositional variables, and $\neg x$ is usually denoted $\bar{x}$. LK proofs can be measured by their size, their depth, their height, and their cut complexity; see [5] or [18] for instance. The *depth* of a formula is based on counting the alternations of $\wedge$'s and $\vee$'s. Depth 0 formulas are literals. Fix a size parameter S. Then depth 0.5 formulas are conjunctions (or disjunctions) of at most $\log S$ many literals. Depth $d + 1$ formulas are the depth d formulas together with conjunctions (or disjunctions) of at most S many depth d formulas. The *depth*, $d(\mathcal{P})$, of a proof $\mathcal{P}$ is the maximum depth of any formula in any sequent in the proof. The *size*, $s(\mathcal{P})$, of $\mathcal{P}$ is the total number of symbols in P, and it is required that $s(\mathcal{P}) \leq S$. The *height*, $h(\mathcal{P})$, of $\mathcal{P}$ is the height of the tree representation of $\mathcal{P}$, i.e., the maximum number of sequents along any branch from an initial sequent to the conclusion.

When determining the depth of a proof, it is useful to note that the cut inferences are the only inferences for which a formula in the lower sequent might not appear as a (sub)formula in an upper sequent of the inference. Consequently, every formula appearing in a proof $\mathcal{P}$ must be a subformula of either a formula appearing in an initial sequent or the endsequent, or a cut formula. It is thus useful to bound the depth of cut formulas, and we will mostly work with proofs that contain no free-cuts. The notion of "free-cut" is defined in the general way of [6]: in the present paper, it means any cut for which the cut formula is not a (descendant of a) formula that appears in an initial sequent.

We are interested in the asymptotic growth rates of families of constant depth proofs $\mathcal{P}_n$. The size of $\mathcal{P}_n$ will be bounded by $S(n) = 2^{n^{O(1)}}$, and its height will be bounded by $n^{O(1)}$. Letting $N = 2^n$, we have the size of $\mathcal{P}_n$ bounded by $2^{(\log N)^{O(1)}}$ and its height bounded by $(\log N)^{O(1)}$; the formulas proved by these proofs will be constant depth and will have size quasipolynomially bounded by N. We measure the complexity of proofs with respect to the size of the sequent it proves. Thus, $(\mathcal{P}_n)_n$ will be a family of quasipolynomial size, polylogarithmic height, free-cut free proofs. It follows that the $\mathcal{P}_n$'s are constant depth.

For information on the Nullstellensatz system, see [2,9]. A Nullstellensatz proof consists of a polynomial over a field F, with variables that are intended to range over the field elements 0 and 1 representing the values "False" and "True", respectively. Starting from a set of initial polynomials q_i, intended to express a set of propositional conditions ϕ, a Nullstellensatz refutation is a set of polynomials p_i such that

$$p_1 q_1 + p_2 q_2 + \cdots p_m q_m = 1.$$

This serves to prove that the q_i's cannot be simultaneously all equal to zero; thus the propositional formulas ϕ cannot be simultaneously satisfied. The polynomials $x^2 - x$ are included among the q_i's to enforce the condition that the variables x

are 0/1-valued. It is common to measure the complexity of a Nullstellensatz refutation in terms of its total degree, i.e., the maximum total degree of the polynomials $p_i q_i$. Our Nullstellensatz refutations will have degree $(\log N)^{O(1)}$, that is, polylogarithmic degree.

3. Forward direction, propositional LK

Suppose that Q_1 and Q_2 are TFNP problems and that $Q_1 \le_{\mathrm{T}} Q_2$. The goal of the present section is to use the Turing reduction M from Q_1 to Q_2 to build constant depth LK proofs which prove that the totality of Q_2 implies the totality of Q_1. As discussed above, this was carried out already by [7] for many-one reductions. Our constructions extend this by allowing M to be a Turing reduction and by giving a more careful analysis of the height and cut complexity of the LK proofs. In addition, we do not need the model extension property that was assumed by [7]. In Section 5, we reverse the construction.

We begin by describing the propositional formulas that express the totality of an NP search problem Q_Φ, where Φ is the $\exists$-sentence $\exists\vec{x}\phi(\vec{x})$. Let Φ be a basic formula over a basic language with one uninterpreted function symbol f. For $n \ge 1$, we define a propositional sequent $F_{\Phi,n} \longrightarrow G_{\Phi,n}$ which expresses the totality of the search problem Q_Φ on inputs of length n. The sequent $F_{\Phi,n} \longrightarrow G_{\Phi,n}$ uses propositional variables $x_{\vec{u},v}$, for all $\vec{u}$, $v \in U_n$, which define the graph of f: the intended meaning of $x_{\vec{u},v}$ is that $f(\vec{u}) = v$. For the $x_{\vec{u},v}$'s to properly define a function, there should be exactly one value $v \in U_n$ for each $\vec{u} \in U_n$ such that $x_{\vec{u},v}$ holds. This condition is expressed by the following formulas.

Definition 3.1. $Tot_{\Phi,n}$ is the set of formulas

$$\left\{ \bigvee\nolimits_{v \in U_n} x_{\vec{u},v} \ : \ \vec{u} \in U_n \right\}.$$

The notation $\bigvee$ is shorthand for $2^n - 1$ many binary disjunctions: these disjunctions are to be applied in a balanced fashion.

$Func_{\Phi,n}$ is the set of formulas

$$\left\{ \overline{x}_{\vec{u},v} \vee \overline{x}_{\vec{u},v'} \ : \ \vec{u}, v, v' \in U_n, \ v \ne v' \right\}.$$

The sets $Tot_{\Phi,n}$ and $Func_{\Phi,n}$ both contain $2^{n^{O(1)}}$ many formulas. Specifically, if the function f has arity k, then $Tot_{\Phi,n}$ contains 2^{kn} many formulas, each of size $O(2^n)$, and $Func_{\Phi,n}$ contains $2^{(k+2)n} - 2^{(k+1)n}$ many formulas, each a disjunction of two literals.

The formulas $\bigwedge Tot_{\Phi,n}$ and $\bigwedge Func_{\Phi,n}$ are formed as balanced conjunctions. The antecedent $F_{\Phi,n}$ is the cedent $\bigwedge Tot_{\Phi,n}, \bigwedge Func_{\Phi,n}$.

$F_{\Phi,n}$ expresses the well-definedness of the interpreted function symbol f; note it depends only on n and the arity k of f. The formula $G_{\Phi,n}$, defined below, expresses the totality of the search problem Q_Φ.

Definition 3.2. Let Φ be in disjunctive normal form $\exists\vec{x} \bigvee_{j=1}^{J} \phi_j(\vec{x})$, with each ϕ_j a conjunction of literals $\ell_{j,1}, \ldots, \ell_{j,i_j}$. Let $\vec{x}$ be a vector of s variables $x_1, \ldots, x_s$; fix $n \ge 1$; and let $\vec{u} = u_1, \ldots, u_s$ be a vector of fixed values in the universe U_n. The intent is that $\vec{u}$ assigns values to the x_i's, namely, $\mathrm{val}_{\vec{u}}(x_i) = u_i$. We extend this notation to built-in constant symbols c by letting $\mathrm{val}_{\vec{u}}(c)$ equal the interpretation of c in U_n.

Each literal $\ell \in \{\ell_{j,i}\}_{i,j}$ is of the form $g(\vec{a}) = b$ or $b = c$ or $b \ne c$, where each $\vec{a}$, b, c is either a variable x_i or a built-in constant symbol. Note that g may be either a built-in function or the uninterpreted function symbol f. The propositional translation $[\![\ell]\!]_{\vec{u}}$ of ℓ under the assignment $\vec{u}$ is defined as follows.

a. If ℓ is $f(\vec{a}) = b$, then $[\![\ell]\!]_{\vec{u}}$ is $x_{\mathrm{val}_{\vec{u}}(\vec{a}),\mathrm{val}_{\vec{u}}(b)}$. The subscript $\mathrm{val}_{\vec{u}}(\vec{a})$ denotes the vector of the values of the members of the vector $\vec{a}$.
b. Otherwise ℓ involves only built-in symbols and variables that have been assigned values by $\vec{u}$. In this case, $[\![\ell]\!]_{\vec{u}}$ is either $\top$ or $\bot$, depending on whether ℓ is true or false in U_n with these assigned values.

The propositional translation $[\![\phi_j]\!]_{\vec{u}}$ of the disjunct ϕ_j in ϕ is defined to be the conjunction of the translations of the literal in ϕ_j, namely,

$$[\![\ell_{j,1}]\!]_{\vec{u}} \wedge [\![\ell_{j,2}]\!]_{\vec{u}} \wedge \cdots \wedge [\![\ell_{j,i_j}]\!]_{\vec{u}}.$$

The formula $Soln_{\Phi,n}$ is defined to be the set of formulas $[\![\phi_j]\!]_{\vec{u}}$, where $j = 1, \ldots, J$, and $\vec{u}$ ranges over all values from U_n. The formula $G_{\Phi,n}$ is defined to be $\bigvee Soln_{\Phi,n}$, namely, a balanced disjunction of the formulas in $Soln_{\Phi,n}$.

That completes the definition of the sequent $F_{\Phi,n} \longrightarrow G_{\Phi,n}$, which expresses the totality of the search problem Q_Φ.

We next state the main theorem of this section. It states that, if $Q_\Phi \le_{\mathrm{T}} Q_\Psi$, then there are "well-behaved" LK-proofs of the sequents $F_{\Phi,n} \longrightarrow G_{\Phi,n}$ from substitution instances of sequents $F_{\Psi,m} \longrightarrow G_{\Psi,m}$, where $m = n^{O(1)}$. A "substitution instance" of $F_{\Psi,m} \longrightarrow G_{\Psi,m}$ means any sequent obtained by uniformly substituting arbitrary propositional formulas $\lambda_{\vec{u},v}$ for the variables $x_{\vec{u},v}$ of $F_{\Psi,m} \longrightarrow G_{\Psi,m}$. The substitution instances will required to be of depth 1.5; that is, the formulas $\lambda_{\vec{u},v}$ will be depth 1.5. (In fact, the substitutions defined below by (4) will be "decision tree substitutions" as defined in Section 5.)

Theorem 3.3. *Suppose that $Q_\Phi \le_{\mathrm{T}} Q_\Psi$. Then there are proofs $\mathcal{P}_n$ of the sequents*

$$F_{\Phi,n} \longrightarrow G_{\Phi,n}$$

such that the following hold.

(a) *The initial sequents of $\mathscr{P}_n$ are either logical axioms or are depth 1.5 substitution instances of sequents $F_{\Psi,m} \longrightarrow G_{\Psi,m}$;*
(b) *$\mathscr{P}_n$ has size $2^{n^{O(1)}}$, height $n^{O(1)}$, and constant depth; and*
(c) *there are no free-cuts in $\mathscr{P}_n$.*

The values m are implicitly bounded by $m = n^{O(1)}$ since the entire proof P_n, and hence each substitution instance of a sequent $F_{\Psi,m} \longrightarrow G_{\Psi,m}$, has size bounded by $2^{n^{O(1)}}$. Recall that a non-free-cut is one whose cut formula is a descendant of a formula in an initial sequent, in this case, a formula from a substitution instance of a sequent $F_{\Psi,m} \longrightarrow G_{\Psi,m}$.

Corollary 3.4. *Suppose that $Q_\Phi \le_{\mathrm{T}} Q_\Psi$ and that the sequents $F_{\Phi,n} \longrightarrow G_{\Phi,n}$ do not have quasipolynomial size, constant depth proofs, where size is measured in terms of the size of the endsequent. Then the sequents $F_{\Psi,n} \longrightarrow G_{\Psi,n}$ do not have polynomial size, constant depth proofs.*

The corollary follows immediately from Theorem 3.3, since the proofs from Theorem 3.3 can be combined with substitution instances of proofs of $F_{\Psi,m} \longrightarrow G_{\Psi,m}$ to obtain proofs of $F_{\Phi,n} \longrightarrow G_{\Phi,n}$.

The proof of Theorem 3.3 will use the possible executions of a Turing reduction M to construct the proofs $\mathscr{P}_n$. For each n, we will construct a set of depth 1.5 substitutions $\sigma_1, \ldots, \sigma_r$, for $r = 2^{n^{O(1)}}$. A substitution will substitute a big (size $2^{n^{O(1)}}$) disjunction of small (size $n^{O(1)}$) conjunctions of variables. In order to enforce the condition that negations apply only to variables, when substituting for a negated atom, De Morgan rules are used to push the negations down to atoms; that is to say, a negated atom is replaced by a big conjunction of small disjunctions of negated variables.

Theorem 3.5. *Suppose that $Q_\Phi \le_{\mathrm{T}} Q_\Psi$. Then there is a set of depth 1.5 substitutions $\sigma_1, \ldots, \sigma_r$, and a set of sizes $m_1, \ldots, m_r$ such that the following sequents have cut free* LK *proofs of size $2^{n^{O(1)}}$ and height $n^{O(1)}$:*

(a) $\left(\bigwedge Tot_{\Phi,n}\right) \longrightarrow \left(\bigwedge Tot_{\Psi,m_i}\right)\sigma_i$, *for each* $i = 1, \ldots, r$,
(b) $\left(\bigwedge Func_{\Phi,n}\right) \longrightarrow \left(\bigwedge Func_{\Psi,m_i}\right)\sigma_i$, *for each* $i = 1, \ldots, r$,
(c) $\bigwedge_{i=1}^{r} \left(\bigvee Soln_{\Psi,m_i}\right)\sigma_i, F_{\Phi,n} \longrightarrow G_{\Phi,n}$.

As Section 5 discusses, conditions (a) and (b) imply that the σ_i are decision tree substitutions; indeed, conditions (a) and (b) are a little stronger than what is required for decision tree substitutions.

Theorem 3.3 follows immediately from Theorem 3.5. To see this, first use cuts with the substitution instances $F_{\Psi,m_i}\sigma_i \longrightarrow G_{\Psi,m_i}\sigma_i$ and the proofs from parts (a) and (b) to derive the sequents

$$F_{\Phi,n} \longrightarrow G_{\Psi,m_i}\sigma_i.$$

Since $G_{\Psi,m_i}\sigma_i$ is $(\bigvee Soln_{\Psi,m_i})\sigma_i$, applying $\wedge$:right in a balanced manner derives the sequent

$$F_{\Phi,n} \longrightarrow \bigwedge_{i=1}^{r} \left(\bigvee Soln_{\Psi,m_i}\right)\sigma_i. \tag{3}$$

Finally, cut this against the sequent (c).

The rest of this section gives the proof of Theorem 3.5. After some work to describe the substitutions, Lemma 3.8 will prove parts (a) and (b); Lemma 3.10 will prove part (c).

Suppose that M is a Turing reduction of Q_Φ to Q_Ψ. Machine M takes as input $(\alpha, 0^n)$, where $\alpha : U_n^k \to U_n$, and $\vec{x} \in U_n$. (The type-0 input is just 0^n since without loss of generality (w.l.o.g.) Φ is an $\exists$-sentence and has no free variables.) M makes queries to the input function α and to the search problem Q_Ψ. A call to $Q_\Psi(\beta, 0^m)$ passes a function $\beta : U_m^{k'} \to U_m$; the function β is specified by describing a polynomial time oracle Turing machine M' such that, for all $\vec{v} \in U_m$, $M'(\vec{v})$ computes the value of $\beta(\vec{v})$. Machine M' is allowed to make oracle calls to α but otherwise runs deterministically. Accordingly, for particular $\vec{v} \in U_m$, the computation of $M'(\vec{v})$ can be described by a (α, n)-decision tree $T_{M',\vec{v}}$ such that the internal nodes of $T_{M',\vec{v}}$ represent queries to α and each leaf of $T_{M',\vec{v}}$ is labeled with an output value:

Definition 3.6. An (α, n)-*decision tree* is a tree in which each internal node is labeled $\alpha(\vec{w})$ for some $\vec{w} \in U_n$. Each internal node has 2^n children, with the edges to its children labeled with values y, one edge for each $y \in U_n$. Leaves may be labeled by any member z of U_m.

In $T_{M',\vec{v}}$, an internal node labeled $\alpha(\vec{w})$ corresponds to a query to $\alpha(\vec{w})$, an outgoing edge labeled y corresponds to an oracle response that $\alpha(\vec{w}) = y$, and a leaf labeled with z indicates that $M'(\vec{v})$ outputs $z = \beta(\vec{v})$. It is clear that, if $M'(\vec{v})$ has runtime bounded by r, then there is a canonical way to build a corresponding (α, n)-decision tree of height $\le r$ which faithfully represents the computation of $M'(\vec{v})$. (Our use of decision trees to represent oracle computations repeats many previous works, including [2,1,7].)

To simplify some aspects of our proofs, we use the convention that all nodes have 2^n children, even if a particular value $\alpha(\vec{v})$ is queried twice on the same branch in the tree. Paths that contain a contradictory pair of answers do not apply to any actual computation of M'.

A *branch* in a tree is any path that starts at the root and descends to a leaf. The set of branches in T is denoted $\mathrm{br}(T)$, and $\mathrm{br}_z(T)$ is the set of branches that end at a leaf with label z. The complementary set of branches is $\mathrm{br}_z(T)^c = \mathrm{br}(T) \setminus \mathrm{br}_z(T)$, namely the set of branches that end with leaf labels $z' \neq z$. The length $|P|$ of a branch P is the number of edges in P. The height $h(T)$ of T is the maximum length of any branch. The size $s(T)$ of T is the number of nodes in T. The oracle queries to Q_Ψ invoked by the Turing reduction $M : Q_\Phi \leq_{\mathrm{T}} Q_\Psi$ give rise to decision trees $T_{M',\vec{v}}$ which have height $n^{O(1)}$, and therefore size $(2^n)^{n^{O(1)}} = 2^{n^{O(1)}}$.

Definition 3.7. If P is a branch in an (α, n)-decision tree, then we identify P with the conjunction

$$\bigwedge\nolimits_{i\in I} x_{\vec{w}_i,y_i},$$

where $\{\alpha(\vec{w}_i) = y_i\}_{i\in I}$ is the set of α values set by the edge labels of P. The formula $\overline{P}$ is defined to be $\bigvee_{i\in I} \overline{x}_{\vec{w}_i,y_i}$: this expresses the negation of P.

When using P in an antecedent, it is convenient to replace the $\wedge$'s with commas to control the complexity of formulas appearing in proofs. Accordingly, $\widehat{P}$ is used to denote the cedent containing the literals $x_{\vec{w}_i,y_i}$, for $i \in I$.

The proofs that will be constructed for Theorem 3.5 contain substitution instances of $F_{\Psi,m} \longrightarrow G_{\Psi,m}$. These substitution instances will be defined from families of (α, n)-decision trees. Namely, suppose that $\mathcal{T}$ is a set of (α, n)-decision trees, $\mathcal{T} = \{T_{\vec{w}} : \vec{w} \in U_m\}$. The intent is that the decision tree $T_{\vec{w}}$ computes the value of $\beta(\vec{w}) \in U_m$. The substitution $\sigma_{\mathcal{T}}$ uses formulas $\lambda_{\mathcal{T},\vec{w},y}$ defined by

$$\lambda_{\mathcal{T},\vec{w},y} = \bigvee_{P\in \mathrm{br}_y(T_{\vec{w}})} P. \tag{4}$$

Note that this formula is a "big" disjunction of "small" conjunctions, since P is identified with the conjunction of literals along a branch of length $n^{O(1)}$, and there are potentially $2^{n^{O(1)}}$ many branches P in $\mathrm{br}_y(T_{\vec{w}})$. The substitution $\sigma_{\mathcal{T}}$ acts by replacing any positively occurring variable $x_{\vec{w},y}$ with the formula $\lambda_{\mathcal{T},\vec{w},y}$. Negatively occurring variables are replaced with the negation $\overline{\lambda_{\mathcal{T},\vec{w},y}}$, namely, the formula $\bigwedge_{P\in \mathrm{br}_y(T_{\vec{w}})} \overline{P}$.

Substitutions are denoted with postfix notation, so $A\sigma_{\mathcal{T}}$ indicates the result of applying the substitution to the propositional formula A.

Lemma 3.8. *Let $\mathcal{T}$ and $\sigma_{\mathcal{T}}$ be as above, and suppose that $m = n^{O(1)}$. Then the following two sequents have* LK *proofs which have size $2^{n^{O(1)}}$, have depth $n^{O(1)}$, and are cut free:*

$$\left(\bigwedge Tot_{\Phi,n}\right) \longrightarrow \left(\bigwedge Tot_{\Psi,m}\right)\sigma_{\mathcal{T}}, \tag{5}$$

$$\left(\bigwedge Func_{\Phi,n}\right) \longrightarrow \left(\bigwedge Func_{\Psi,m}\right)\sigma_{\mathcal{T}}. \tag{6}$$

Proof of Lemma 3.8. We first prove (5). $\left(\bigwedge Tot_{\Psi,m}\right)\sigma_{\mathcal{T}}$ is a balanced conjunction of the formulas $Tot_{\mathcal{T},\vec{v}}$ defined as

$$\bigvee_{z\in U_m} \bigvee_{P\in \mathrm{br}_z(T_{\vec{v}})} P,$$

with one such formula for each choice of $\vec{v} \in U_m$. Together, the two big disjunctions range over all branches P in $T_{\vec{v}}$. For each branch P, it is trivial that $\widehat{P} \longrightarrow P$ is provable with a cut free proof. Therefore, using $\vee$:right introductions, we obtain a cut free LK proof $\mathscr{P}_P$ of $\widehat{P} \longrightarrow Tot_{\mathcal{T},\vec{v}}$.

Fix $\vec{v} \in U_n$. Let $\widehat{P}_{\upharpoonright i}$ be the cedent containing the first i members of $\widehat{P}$. We construct LK proofs $\mathscr{P}_{P,i}$ of the sequents $\bigwedge Tot_{\Phi,n}, \widehat{P}_{\upharpoonright i} \longrightarrow Tot_{\mathcal{T},\vec{v}}$ by induction, with i varying from $|P|$ down to 0. For $i = |P|$, $\mathscr{P}_{P,i}$ is obtained from $\mathscr{P}_P$ by a weakening inference. For the induction step, let the $(i + 1)$st member of $\widehat{P}$ be $x_{\vec{w}_{i+1},y_{i+1}}$. Since the P come from the decision tree $T_{\vec{v}}$, there are paths $P_{y'}$ such that $(\widehat{P}_{y'})_{\upharpoonright i+1} = \widehat{P}_{\upharpoonright i}, x_{\vec{w}_{i+1},y'}$ for every $y' \in U_n$. The induction hypothesis gives proofs $\mathscr{P}_{P_{y'},i+1}$ for the sequents $\bigwedge Tot_{\Phi,n}, \widehat{P}_{\upharpoonright i}, x_{\vec{w}_{i+1},y'} \longrightarrow Tot_{\mathcal{T},\vec{v}}$. Combining these with $\vee$:left inferences in a balanced fashion gives a proof of $\bigwedge Tot_{\Phi,n}, \widehat{P}_{\upharpoonright i}, \bigvee_{y'}(x_{\vec{w}_i,y'}) \longrightarrow Tot_{\mathcal{T},\vec{v}}$. The introduced disjunction is a member of $Tot_{\Phi,n}$. The desired proof $\mathscr{P}_{P,i}$ is obtained by weakening and then applying $\wedge$:left inferences and a contraction. If $i = 0$, then $\widehat{P}_{\upharpoonright 0}$ is empty, and $\mathscr{P}_{P,0}$ is a proof of the sequent $\bigwedge Tot_{\Phi,n} \longrightarrow Tot_{\mathcal{T},\vec{v}}$.

Combining all these proofs with $\wedge$:right inferences gives the desired LK proof of (5). The size bounds and the cut free property are easy to verify.

Now we prove (6). The formula $\left(\bigwedge Func_{\Psi,m}\right)\sigma_{\mathcal{T}}$ is a big conjunction of the formulas $Func_{\mathcal{T},\vec{v},z,z'}$ defined as

$$\left(\bigwedge_{P\in \mathrm{br}_z(T_{\vec{v}})} \overline{P}\right) \vee \left(\bigwedge_{P'\in \mathrm{br}_{z'}(T_{\vec{v}})} \overline{P}'\right), \tag{7}$$

with $\vec{v}, z, z' \in U_m$ and $z \neq z'$. Fix $\vec{v}, z, z'$. Any pair of branches $P \in \mathrm{br}_z(T_{\vec{v}})$ and $P' \in \mathrm{br}_{z'}(T_{\vec{v}})$, as distinct paths in a single decision tree, must contain a clashing assignment to α. That is to say, there is some pair of literals $x_{\vec{w},y}$ on P and $x_{\vec{w},y'}$ on P' with $y \neq y'$. This means that $\overline{P}$ and $\overline{P}'$ contain $\bar{x}_{\vec{w},y}$ and $\bar{x}_{\vec{w},y'}$ as disjuncts (respectively). We thus get simple cut free proofs of the sequent $\bigwedge Func_{\Phi,n} \longrightarrow \overline{P}, \overline{P}'$. Combining these proofs with $\wedge$:right inferences (as usual, in a balanced fashion), we obtain proofs of the sequents $\bigwedge Func_{\Phi,n} \longrightarrow \overline{P}, \bigwedge_{P' \in \mathrm{br}_y(T_{\vec{w}})} \overline{P}'$. Combining these proofs with more balanced $\wedge$:right inferences, and then a single $\vee$:right inference, gives a proof of the sequent $\bigwedge Func_{\Phi,n} \longrightarrow (7)$.

Finally, letting $\vec{w}, y, y'$ vary, using $\wedge$:right inferences gives the desired proof of $\bigwedge Func_{\Phi,n} \longrightarrow \left(\bigwedge Func_{\Psi,m}\right) \sigma_{\mathcal{T}}$. The size bounds and the cut free property are left to the reader to verify. □

We now define decision trees for the computation of the machine M which computes a Turing reduction from Q_Φ to Q_Ψ. This is more complicated than the (α, n)-decision trees used for M' above, since M makes queries to instances of Ψ as well as to α. For $n \in \mathbb{N}$, the decision tree $T_{M,n}$ for the computation of $M(\alpha, 0^n)$ is defined as follows. Each internal node of the decision tree is labeled either (a) with label $\alpha(\vec{w})$ for some $\vec{w} \in U_n$, or (b) with label $\Psi(\beta, 0^m)$ with β described as a polynomial time oracle Turing machine M'. A node with label $\alpha(\vec{w})$ has 2^n children and outgoing edges labeled with values y, one outgoing edge per $y \in U_n$. As before, traversing the outgoing edge labeled y indicates that $\alpha(\vec{w}) = y$. The nodes labeled with $\Psi(\beta, 0^m)$ have $J' \cdot 2^{s'm}$ children, where s' is the output arity of Q_Ψ; the outgoing edges are labeled with all possible values $(j, \vec{a})$ with $1 \leq j \leq J'$ and $\vec{a} \in U_m^{s'}$. The intuition is that the edge with label $(j, \vec{a})$ may be traversed if $\vec{a}$ is a solution to the search problem $\Psi(\beta, 0^m)$ and the jth disjunct of Ψ is satisfied by setting the existentially quantified variables of Ψ equal to $\vec{a}$. The leaf nodes of $T_{M,n}$ are labeled with the value output by $M(\alpha, 0^n)$ after the computation leading to that leaf; thus, each leaf node is to be labeled with a tuple of values $\vec{b}$ which provides a solution to the search problem Q_Φ (with the exception that an arbitrary value may be given for a contradictory path through $T_{M,\alpha}$). Since M has polynomial runtime $n^{O(1)}$, the height of $T_{M,n}$ is $n^{O(1)}$. Each node has $2^{n^{O(1)}}$ children, and hence the tree has size $2^{n^{O(1)}}$.

The nodes labeled $\Psi(\beta, 0^m)$ in $T_{M,n}$ require more explanation. First, note that the outgoing edges are labeled with values $(j, \vec{a})$, that is to say, the outgoing edge label specifies both the solution to the NP search problem $\Psi(\beta, 0^m)$ and the index j of the disjunct of Ψ which is satisfied by $\vec{a}$. On the other hand, when M calls the oracle $\Psi(\beta, 0^m)$, only a value $\vec{a}$ is returned, not the value of j. Nonetheless, we can assume w.l.o.g. that the oracle call also returns j since M can quickly determine a value for j by evaluating Ψ with its existentially quantified variables set equal to $\vec{a}$. Second, a path in the decision tree $T_{M,n}$ may contain contradictions not only in the values given for α, but may also contain “implicit” contradictions if the edge labels $(j, \vec{a})$ on the path specify solutions that are incompatible with the values of α specified elsewhere on the path. Of course, M could recognize such contradictions quickly by evaluating the jth disjunct of Ψ with the values $\vec{a}$. Machine M is not required to check for these contradictions, but the LK proofs constructed below will be constructed as if M does immediately verify the correctness of edge labels $(j, \vec{a})$. Indeed, we next define the notion of a “trace” of a path in $T_{M,n}$ for exactly this purpose.

Definition 3.9. Let μ be a node in $T_{M,n}$. Let P_μ denote the path from the root of $T_{M,n}$ to the node μ. We define the set of traces to μ by induction on the length of P_μ. Any trace to μ will consist of a sequence of literals $x_{\vec{w},y}$ intended to indicate the conditions that $\alpha(\vec{w}) = y$. The following constructions may be used to form traces.

(a) For the base case, if μ is the root node and P_μ has length zero, then the only trace to μ is the empty sequence.

(b) Suppose that node μ is labeled with a query $\alpha(\vec{w})$, and let μ' be a child of μ with the edge (μ, μ') labeled by $y \in U_n$. If Λ is a trace to μ, then $\Lambda, x_{\vec{w},y}$ is a trace to μ'.

(c) Now suppose that μ is labeled with a query $\Psi(\beta, 0^m)$, and let μ' be a child of μ that is reached by an edge labeled with $(j, \vec{a})$. Letting $\psi_j(\vec{x})$ be the jth disjunct of Ψ, consider $\psi_j(\vec{a})$. If $\psi_j(\vec{a})$ contains a conjunct $g(\vec{y}) = y'$ or $y = y'$ or $y \neq y'$ which is false, then there is no trace to μ. (Here, y, y' are uninterpreted constants or members of $\vec{a}$, and g is built in.) Otherwise, let $\mathcal{T} = \{T_{\vec{w}}\}_{\vec{w} \in U_m}$ be the family of (α, n)-decision trees for β, and let $\beta(\vec{w}_i) = y_i$, where $i = 1, \ldots, r$, be the atomic formulas of ψ_j that involve β. In this case, for any trace Λ to μ and any choice of paths P_i in $\mathrm{br}_{y_i}(T_{\vec{w}_i})$, the sequence $\Lambda, \widehat{P}_1, \ldots, \widehat{P}_r$ is a trace to μ'.

The motivation for part (c) of the definition is that, in order for $\psi_j(\vec{a})$ to be true, each of $\beta(\vec{w}_i) = y_i$ must hold. That is to say, there must be some choice for branches $P_1, \ldots, P_k$ such that all literals in the sequence are true.

For μ a node in a tree T, let T_μ be the subtree of T rooted at μ.

Lemma 3.10. *Let there be r nodes in $T_{M,n}$ which are labeled with calls to Ψ. To fix notation, for $i = 1, \ldots, r$, there are integers m_i and calls to $\Psi(\beta_i, 0^{m_i})$ in $T_{M,n}$, where the function β_i is defined by a Turing machine M_i'. Each machine M_i' generates a family $\mathcal{T}^i$ of (α, n)-decision trees $\mathcal{T}^i = \{T^i_{\vec{w}} : \vec{w} \in U_{m_i}\}$. Let σ_i be $\sigma_{\mathcal{T}^i}$.*

Let μ be a node in $T_{M,n}$, and let Λ be a trace to μ. Then there is a cut free proof $\mathcal{P}_\mu$ of the sequent

$$\Lambda, \bigwedge_{i=1}^{r} (\bigvee Soln_{\Psi,m_i})\sigma_i, \qquad F_{\Phi,n} \longrightarrow G_{\Phi,n}, \tag{8}$$

such that $\mathcal{P}_\mu$ has size $2^{n^{O(1)}}$ and height $n^{O(1)}$.

Although it is suppressed in the notation, P_μ depends on both μ and Λ.

Proof. The trace Λ contains variables $x_{\vec{w},y}$ with $\vec{w}, y \in U_n$. We define Λ to be *contradictory* if it contains two literals $x_{\vec{w},y}$ and $x_{\vec{w},y'}$, where $y \neq y'$. If Λ is contradictory, then there is an easy cut free proof of $\bigwedge Func_{\Phi,n}, \Lambda \longrightarrow$. From this, (8) can be inferred by weakening.

For traces which are not contradictory, we prove the existence of the proofs $\mathcal{P}_\mu$ by induction on the depth of μ in the decision tree $T_{M,n}$. To establish the size bounds, we show that each $\mathcal{P}_\mu$ has height $n^{O(1)} \cdot (1 + h_\mu)$, where h_μ is the height of the subtree T_μ. The induction progresses from the leaves to the root.

For the base case, let μ be a leaf, and let Λ be a trace to μ. As a leaf node, μ corresponds to a halting configuration of M. Once the execution of M reaches μ, M is finished querying α and Ψ, and it then runs deterministically to output values $\vec{a}$ in U_n that satisfy some disjunction $\phi_j(\vec{a})$ of $\phi(\vec{a})$. The propositional translation $[\![\phi_j]\!]_{\vec{a}}$ of $\phi_j(\vec{a})$ is a conjunction of variables $x_{\vec{w},y}$ plus possibly occurrences of $\bot$ and $\top$.

We claim that $\bot$ does not occur in $[\![\phi_j]\!]_{\vec{a}}$, and that every variable in $[\![\phi_j]\!]_{\vec{a}}$ also appears in Λ. To prove this claim, it is sufficient to note that the trace Λ contains enough information to ensure that there is a correct computation of M that reaches node ν in the decision tree. And, since M is, in actuality, a Turing reduction from Q_Φ to Q_Ψ, it follows that the output $\vec{a}$ is valid and thus makes some $\phi_j(\vec{a})$ true. This implies that no conjunct in $\phi_j(\vec{a})$ is false. Therefore, $\bot$ does not appear in $[\![\phi_j]\!]_{\vec{a}}$. In addition, every variable $x_{\vec{w},y}$ in $[\![\phi_j]\!]_{\vec{a}}$ must appear in the trace Λ, since, if not, the value of $\alpha(\vec{w})$ could be set to some value $y' \neq y$ and still be consistent with the values specified by Λ. This cannot happen, by virtue of M's being a correct Turing reduction.

It follows that the sequent $\Lambda \longrightarrow [\![\phi_j]\!]_{\vec{a}}$ has a cut free proof (containing only $\wedge$:right inferences). Adding weakening inferences and $\vee$:right inferences yields the desired proof $\mathcal{P}_\mu$ of the sequent (8). It is clear that this proof has height $n^{O(1)}$.

Now, suppose that μ is an internal node labeled with an α query, $\alpha(\vec{w})$. For each $y \in U_n$, μ has a child μ_y. By the definition of a trace, $\Lambda, x_{\vec{w},y}$ is a trace to μ_y. Consider the proofs $\mathcal{P}_{\mu_y}$ given by the induction hypothesis of the sequents

$$\Lambda, x_{\vec{w},y}, \bigwedge_{i=1}^{r} \left(\bigvee Soln_{\Psi,m_i}\right)\sigma_i, F_{\Phi,n} \longrightarrow G_{\Phi,n}.$$

Combining these with $\vee$:left inferences eliminates the $x_{\vec{w},y}$'s in the antecedent and gives a member of $Tot_{\Phi,n}$. The desired proof $\mathcal{P}_\mu$ is obtained by weakening and adding $\wedge$:left inferences and a contraction on $\bigwedge Tot_{\Phi,n}$. The height of $\mathcal{P}_\mu$ is at most $n^{O(1)}$ greater than the maximum height of any $\mathcal{P}_{\mu_y}$.

Finally, suppose that μ is an internal node labeled $\Psi(\beta, 0^m)$ with the function β described by Turing machine M' and induced family $\widehat{\mathcal{T}} = \{\widehat{T}_{\vec{w}}\}_{\vec{w}}$ of decision trees. Each child μ' of μ is reached by an edge labeled with $(j, \vec{a})$; any trace to μ' has the form $\Lambda, \widehat{P}_{\mu',1}, \ldots, \widehat{P}_{\mu',k_{\mu'}}$, where the cedents $\widehat{P}_{\mu',i}$ are in $\mathrm{br}_{y_i}(T_{\vec{w}_i})$ for $\beta(\vec{w}_i) = y_i$ the ith conjunct of $\psi_j(\vec{a})$ that involves β.

The induction hypothesis, together with weakening, gives proofs of the sequents

$$\Lambda, \widehat{P}_{\mu',1}, \ldots, \widehat{P}_{\mu',k_{\mu'}}, \bigwedge_{i=1}^{r} \left(\bigvee Soln_{\Psi,m_i}\right)\sigma_i, F_{\Phi,n} \longrightarrow G_{\Phi,n}.$$

To form the proof $\mathcal{P}_\mu$, it will suffice to show the sequent

$$\left(\bigvee Soln_{\Psi,m}\right)\sigma_i \longrightarrow \tag{9}$$

can be proved from the set of sequents

$$P_{\mu',1}, \ldots, P_{\mu',k_{\mu'}} \longrightarrow \tag{10}$$

with a cut free proof of height $n^{O(1)}$.

For the moment, we fix μ', and hence j and $\vec{a}$. Let $\bigvee P_{\mu',i}$ denote the balanced disjunction taken over all members of $\mathrm{br}_{y_i}(T_{\vec{w}_i})$. For fixed $P_{\mu',1}, \ldots, P_{\mu',k_{\mu'}-1}$, we use balanced $\vee$:left inferences to combine sequents (10) to obtain

$$P_{\mu',1}, \ldots, P_{\mu',k_{\mu'}-1}, \bigvee P_{\mu',k_{\mu'}} \longrightarrow .$$

Continuing to apply $\vee$:left inferences in the same way on $k_{\mu'} - 1$ down to 1 yields

$$\bigvee P_{\mu',1}, \ldots, \bigvee P_{\mu',k_{\mu'}} \longrightarrow .$$

Note that each $\bigvee P_{\mu',i}$ is one of the conjuncts of $([\![\psi_j]\!]_{\vec{a}})\sigma_i$. Thus, applying a constant number of $\wedge$:inferences (and possibly a weakening to introduce $\top$), gives a proof of the sequent $([\![\psi_j]\!]_{\vec{a}})\sigma_i \longrightarrow$.

(A different construction is needed for the case where $\bot$ is in $[\![\psi_j]\!]_{\vec{a}}$. In this case, there are no traces to μ'. However, it is easy to derive $([\![\psi_j]\!]_{\vec{a}})\sigma_i \longrightarrow$ from the logical initial sequent $\bot \longrightarrow$.)

Finally, applying $\vee$:left inferences to these last sequents, letting j and $\vec{a}$ vary over all possible values, gives a cut free proof of (9) from the sequents (10) as desired. It is easy to verify that this cut free proof has height $n^{O(1)}$. □

Part (c) of Theorem 3.5 follows immediately from Lemma 3.10 by taking μ to be the root of $T_{M,n}$, since the trace to the root is empty. Parts (a) and (b) of the theorem were already established in Lemma 3.8. This completes the proof of Theorem 3.5 and hence of Theorem 3.3.

Existing upper and lower bounds on the complexity of proofs of $F_{\Phi,n} \longrightarrow G_{\Phi,n}$ in conjunction with Corollary 3.4 give Turing separations between various TFNP^2 problems. The separations (a)–(c) of Corollary 3.11 were already proved in [1]. However, Corollary 3.11 gives new proofs of these facts as well as emphasizing the connection to proof complexity theory. Previously, [7] used proof complexity results to recreate these separations only with respect to many-one reductions, not Turing reducibility.

Corollary 3.11. (a) PIGEON $\not\leq_{\mathrm{T}}$ ITER. *[20]*
(b) LONELY $\not\leq_{\mathrm{T}}$ ITER. *[20]*
(c) LONELY $\not\leq_{\mathrm{T}}$ PIGEON. *[1]*

Proof. [3] shows that $F_{\Phi,n} \longrightarrow G_{\Phi,n}$ require exponential size bounded depth proofs when Φ is LONELY or PIGEON. By [7], $F_{\Phi,n} \longrightarrow G_{\Phi,n}$ have quasipolynomial size bounded depth proofs when Φ is ITER. The separations (a) and (b) therefore follow from Corollary 3.4. It is shown in [4] that proving $F_{\mathrm{LONELY},n} \longrightarrow G_{\mathrm{LONELY},n}$ from substitution instances of $F_{\mathrm{PIGEON},n} \longrightarrow G_{\mathrm{PIGEON},n}$ requires exponential size bounded depth proofs. Therefore (c) holds by Theorem 3.3. □

4. Nullstellensatz

This section shows that, if $Q_1 \leq_{\mathrm{T}} Q_2$ and there are polylogarithmic degree Nullstellensatz proofs that Q_2 is total, then there are polylogarithmic degree Nullstellensatz proofs that Q_1 is total. This extends the results of [7] which proved this result with respect to many-one reductions.

Definition 4.1. Suppose that p is a polynomial and that $F = \{f_i\}$ is a set of polynomials. Write $p \in \langle F, d\rangle$ if and only if there exist polynomials g_i such that $p = \sum_i f_i g_i$ and the maximum degree of the $f_i g_i$'s is at most d. The g_i's are a *Nullstellensatz proof* of p from F. If p is 1, the g_i's are a *Nullstellensatz refutation* of F. If q is a polynomial, then $p \in q + \langle F, d\rangle$ if there is an $r \in \langle F, d\rangle$ such that $p = q + r$.

Let Q_Φ be the NP search problem in which Φ is the $\exists$-sentence $\exists\vec{x}\phi(\vec{x})$, such that Φ is a basic formula over a basic language. A Nullstellensatz proof that Q_Φ is total will actually be a Nullstellensatz refutation of the fact that Q_Φ is not total. We define a set of polynomials $F_{\Phi,n}$, which, when simultaneously set to 0, encode that Φ fails to be total on inputs of length n. The polynomials in $F_{\Phi,n}$ use variables $x_{\vec{u},v}$ to define the graph of f. The intended meaning is that $x_{\vec{u},v}$ is 1 if $f(\vec{u}) = v$ and $x_{\vec{u},v}$ is 0 if $f(\vec{u}) \neq v$. To ensure that the $x_{\vec{u},v}$'s are 0/1-valued, we define the following set of formulas.

Definition 4.2. Let $Bool_{\Phi,n}$ be $\{x^2_{\vec{u},v} - x_{\vec{u},v} | \vec{u} \in U_n, v \in U_n\}$.

Polynomials are *satisfied* by an assignment if they are set to 0. It is clear that any assignment that satisfies $Bool_{\Phi,n}$ sets each $x_{\vec{u},v}$ to either 0 or 1.

The following sets ensure that f is a total function.

Definition 4.3. Let $Tot_{\Phi,n}$ be $\{\sum_{v \in U_n} x_{\vec{u},v} - 1 | \vec{u} \in U_n\}$ and let $Func_{\Phi,n}$ be $\{x_{\vec{u},v}x_{\vec{u},v'} | \vec{u} \in U_n, v \neq v' \in U_n\}$.

If $Tot_{\Phi,n}$ is satisfied then f is total, and if $Func_{\Phi,n}$ is satisfied then f is single valued. Further, it is easy to see that $x^2_{\vec{u},v} - x_{\vec{u},v}$ is in $\langle Tot_{\Phi,n} \cup Func_{\Phi,n}, 2\rangle$. Therefore, if S is a set of formulas such that $Tot_{\Phi,n} \cup Func_{\Phi,n} \subseteq S$ and $p \in \langle S, d\rangle$, then $p \in \langle S \backslash Bool_{\Phi,n}, d+2\rangle$. Since in applications d is $n^{O(1)}$, we omit $Bool_{\Phi,n}$ from further arguments.

Finally, to express the fact that Φ is not total, we need to encode the fact that $\phi_j(\vec{u})$ is false, for each choice of j, $\vec{u}$. Recall the propositional translation $[\![\cdot]\!]_{\vec{u}}$ of Section 3.

Definition 4.4. Let Φ be in disjunctive normal form $\exists\vec{x}\bigvee_{j=i}^{J}\phi_j(\vec{x})$, where each ϕ_j is a conjunction of literals $\ell_{j,1}, \ldots, \ell_{j,i_j}$. Let $\vec{x}$ be a vector of s variables $x_1, \ldots, x_s$. Fix $n \geq 1$, and let $\vec{a} = a_1, \ldots, a_s$ be a vector of fixed values from U_n.

The Nullstellensatz translation $p_{\vec{a}}(\neg\phi_j)$ of $\neg\phi_j$ under the assignment $\vec{a}$ is the polynomial $\Pi_{i=1}^{i_j} p_{\vec{a}}(\ell_{j,i})$, where

$$p_{\vec{a}}(\ell_{j,i}) = \begin{cases} 1 & \text{if } [\![\ell_{j,i}]\!]_{\vec{a}} \text{ is } \top \\ 0 & \text{if } [\![\ell_{j,i}]\!]_{\vec{a}} \text{ is } \bot \\ x_{\vec{u},v} & \text{if } [\![\ell_{j,i}]\!]_{\vec{a}} \text{ is } x_{\vec{u},v}. \end{cases}$$

Define $Failure_{\Phi,n}$ to be the set $\{p_{\vec{a}}(\neg\phi_j) | 1 \leq j \leq J, \vec{a} \in U_n\}$. Define $F_{\Phi,n}$ to be the union of $Tot_{\Phi,n}$, $Func_{\Phi,n}$, and $Failure_{\Phi,n}$.

If $p_{\vec{a}}(\neg\phi_j)$ is set to 0, then some factor $p_{\vec{a}}(\ell_{j,i})$ of $p_{\vec{a}}(\neg\phi_j)$ is 0. There are two ways in which $p_{\vec{a}}(\ell_{j,i})$ can be 0. The first is if $[\![\ell_{j,i}]\!]_{\vec{a}}$ is $\bot$ and the second is if $p_{\vec{a}}(\ell_{j,i})$ is $x_{\vec{u},v}$ and $f(\vec{u}) \neq v$. In either case, $\phi_j(\vec{a})$ is false. Therefore, when $Failure_{\Phi,n}$ is satisfied, each solution to Q_Φ is ruled out.

Theorem 4.5 is the main result of this section.

Theorem 4.5. *If M is a Turing reduction from Q_Φ to Q_Ψ and $1 \in \langle F_{\Psi,m}, m^{O(1)}\rangle$, then $1 \in \langle F_{\Phi,n}, n^{O(1)}\rangle$.*

In particular, if there are degree $n^{O(1)}$ Nullstellensatz proofs that Q_Ψ is total, but any Nullstellensatz proof that Q_Φ is total requires degree $2^{n^{O(1)}}$, then Theorem 4.5 implies that $Q_\Phi \not\leq_{\mathrm{T}} Q_\Psi$. This fact is used to obtain the separations in Corollary 4.10 below.

The proof of Theorem 4.5 closely follows the proof of Theorem 3.3. Namely, given a Turing reduction M from Q_Φ to Q_Ψ, we use the (α, n)-decision trees for the β's on which M queries Q_Ψ to define a substitution. Then we prove Lemma 4.7, which involves general traces in $T_{M,n}$. Theorem 4.5 is just the special case of Lemma 4.7 applied to the trace to the root.

Branches in (α, n)-decision trees are now associated with polynomials.

Definition 4.6. If P is a branch in an (α, n)-decision tree, then we identify P with the product

$$\prod_{i\in I} x_{\vec{u}_i, v_i},$$

where $\{\alpha(\vec{u}_i) = v_i\}_{i\in I}$ is the set of α values set by the edge labels in P.

A family of (α, n)-decision trees, $\mathcal{T} = \{T_{\vec{w}} \mid \vec{w} \in U_m\}$, induces a substitution $\sigma_{\mathcal{T}}$ in much the same manner as in Section 3. Let $\lambda_{\mathcal{T},\vec{w},y}$ be

$$\sum_{P\in \mathrm{br}_y(T_{\vec{w}})} P.$$

Given a polynomial $p(x_{\vec{w}_1,y_1}, x_{\vec{w}_2,y_2}, \ldots)$ in the variables $\{x_{\vec{w},y}\}_{\vec{w},y\in U_m}$, the polynomial $p\sigma_{\mathcal{T}}$ is the polynomial $p(\lambda_{\mathcal{T},\vec{w}_1,y_1}, \lambda_{\mathcal{T},\vec{w}_2,y_2}, \ldots)$.

Recall the definition of a trace in Definition 3.9. If Λ is a trace, then identify Λ with the polynomial $\prod_{x_{\vec{u},v}\in\Lambda} x_{\vec{u},v}$.

Lemma 4.7. *Let M be a Turing reduction from Q_Φ to Q_Ψ, and suppose that $1 \in \langle F_{\Psi,m}, m^{O(1)}\rangle$. If Λ is a trace to μ, then $\Lambda \in \langle F_{\Phi,n}, n^{O(1)}\rangle$.*

Before proving Lemma 4.7, we first prove two lemmas that will be needed in the proof.

Lemma 4.8. *Let $\mathcal{T} = \{T_{\vec{w}}\}$ be a family of (α, n)-decision trees such that each $T_{\vec{w}} \in \mathcal{T}$ has height at most d. Then $(Tot_{\Psi,m})\sigma_{\mathcal{T}} \subseteq \langle F_{\Phi,n}, d\rangle$.*

Proof. We begin by proving the following claim:

Claim: For any (α, n)-decision tree T of height at most d, $\sum_{P\in\mathrm{br}(T)} P - 1 \in \langle Tot_{\Phi,n}, d\rangle$.

The proof proceeds by induction on the structure of T. For the base case, if T is a single node, then there is only one empty branch. Since the empty branch corresponds to the polynomial 1, the result is trivial.

Otherwise, pick a node μ of T labeled $\alpha(\vec{u})$ whose children are leaves and let T' be T with μ's children pruned. Then, by the induction hypothesis, $\sum_{P'\in\mathrm{br}(T')} P' - 1 \in \langle Tot_{\Phi,n}, d'\rangle$, where d' is the height of T'. Let P'_μ be the branch in T' that ends at μ. The branches in T and T' coincide, except the branch P'_μ in T' is replaced by the branches $P'_\mu x_{\vec{u},v}$ in T for each $v \in U_n$. Thus

$$\begin{aligned}\sum_{P\in\mathrm{br}(T)} P - 1 &= \sum_{\substack{P'\in\mathrm{br}(T'):\\ P'\neq P'_\mu}} P' + P'_\mu \sum_{v\in U_n} x_{\vec{u},v} - 1\\ &= \sum_{\substack{P'\in\mathrm{br}(T'):\\ P'\neq P'_\mu}} P' + \left(P'_\mu\left(\sum_{v\in U_n} x_{\vec{u},v} - 1\right) + P'_\mu\right) - 1 = \sum_{P'\in\mathrm{br}(T')} P' - 1 + P'_\mu\left(\sum_{v\in U_n} x_{\vec{u},v} - 1\right).\end{aligned}$$

Thus $\sum_{P\in\mathrm{br}(T)} P - 1$ is in

$$\langle Tot_{\Phi,n}, d'\rangle + \langle Tot_{\Phi,n}, |P'_\mu| + 1\rangle \subseteq \langle Tot_{\Phi,n}, d\rangle,$$

since $d' \leq d$ and $|P'_\mu| \leq d - 1$.

To prove the lemma, let $p \in (Tot_{\Phi,m_i})\sigma_{\mathcal{T}}$ be $\sum_{y\in U_n}\sum_{P\in\mathrm{br}_y(T_{\vec{w}})} P - 1$; note that p is equal to $\sum_{P\in\mathrm{br}(T_{\vec{w}})} P - 1$. The claim implies that $p \in \langle Tot_{\Phi,n}, d\rangle$, since the height of $T_{\vec{w}}$ is at most d. □

Lemma 4.9. *Let $\mathcal{T} = \{T_{\vec{w}}\}$ be a family of (α, n)-decision trees, and let m be $n^{O(1)}$. Then $(Func_{\Psi,m})\sigma_{\mathcal{T}} \subseteq \langle F_{\Phi,n}, n^{O(1)}\rangle$.*

Proof. Any polynomial in $(Func_{\Phi,m})\sigma_{\mathcal{T}}$ is a sum of monomials of the form P_1P_2, where $P_1 \in \mathrm{br}_y(T_{\vec{w}})$, $P_2 \in \mathrm{br}_{y'}(T_{\vec{w}})$, and $y \neq y'$. Since P_1 and P_2 have different outputs, there is a first query where they differ; say it is $\alpha(\vec{u})$. Then $x_{\vec{u},v}$ is a factor of P_1 and $x_{\vec{u},v'}$ is a factor of P_2, for some $v \neq v'$. Thus $P_1P_2 \in \langle Func_{\Phi,n}, n^{O(1)}\rangle$, and the result follows. □

Proof of Lemma 4.7. The proof proceeds in a manner similar to the proof of Lemma 3.10 to show that $\Lambda \in \langle F_{\Phi,n}, n^c\rangle$ for a suitably large constant c. The constraints on c will be made apparent during the proof, but c must strictly dominate the constants of Lemmas 4.8 and 4.9, and n^c must strictly dominate the runtime of Q_Φ and the implicit polynomial bounds on the runtimes of the calls to Q_Ψ.

Define Λ to be *contradictory* if it contains two factors $x_{\vec{u},v}$ and $x_{\vec{u},v'}$, where $v \neq v'$. If Λ is contradictory, then $\Lambda \in \langle Func_{\Phi,n}, n^c \rangle \subseteq \langle F_{\Phi,n}, n^c \rangle$, provided that $c > c_1$, where n^{c_1} bounds the length of a trace.

Let h_μ be the height of the subtree of $T_{M,n}$ rooted at μ. For traces that are not contradictory, we will prove that $\Lambda \in \langle F_{\Phi,n}, n^c(h_\mu + 1) \rangle$, by induction on h_μ. For the base case, let μ be a leaf of $T_{M,n}$, and let Λ be a trace to μ. As in the proof of Lemma 3.10, when the execution of M reaches μ, M deterministically produces an output $\vec{a}$ that satisfies a disjunct $\phi_j(\vec{a})$ of $\phi(\vec{a})$. Consider $p = p_{\vec{a}}(\neg\phi_j)$. Since $\phi_j(\vec{a})$ is true, p is not the zero polynomial. Also, each factor $x_{\vec{u},v}$ of p must be a factor of Λ, since otherwise the value of $\alpha(\vec{u})$ could be set to some $v' \neq v$ that is consistent with the other values of α specified by Λ. But this would imply that $\phi_j(\vec{a})$ is not true, which cannot happen, since M is a correct Turing reduction. Since each factor of p is a factor of Λ, we have $\Lambda \in \langle p, n^c \rangle \subseteq \langle F_{\Phi,n}, n^c \rangle$.

Now, let μ be an internal node. There are two cases based on the label of μ. Suppose that μ is labeled $\alpha(\vec{u})$. Let μ' be a child of μ with the edge (μ, μ') labeled $\alpha(\vec{u}) = v$. By the induction hypothesis, $\Lambda x_{\vec{u},v}$ is in

$$\langle F_{\Phi,n}, n^c \cdot (h_{\mu'} + 1) \rangle.$$

Rewrite Λ as

$$\Lambda \sum_{v \in U_n} x_{\vec{u},v} + \left(1 - \sum_{v \in U_n} x_{\vec{u},v}\right) \Lambda.$$

Thus, $\Lambda \in \langle F_{\Phi,n}, S + 1 \rangle + \langle Tot_{\Phi,n}, n^c \rangle \subseteq \langle F_{\Phi,n}, \max\{S + 1, n^c\} \rangle$, where S is the maximum of $n^c \cdot (h_{\mu'} + 1)$ taken over all children μ' of μ. This immediately implies that $\Lambda \in \langle F_{\Phi,n}, n^c \cdot (h_\mu + 1) \rangle$.

Now, let μ be labeled $\Psi(\beta, 0^m)$. Let μ' be a child of μ such that the edge from μ to μ' is labeled $(j, \vec{a})$. Let $p = p_{\vec{a}}(\neg\psi_j)$ and suppose p is $\prod_{i=1}^{I} x_{\vec{w}_i, y_i}$. Then $\Lambda(p\sigma_{\mathcal{T}})$ is

$$\Lambda \prod_{i=1}^{I} \sum_{P_i \in \mathrm{bry}_{y_i}(T_{\vec{w}_i})} P_i = \sum_{P_1 \in \mathrm{bry}_{y_1}(T_{\vec{w}_1}), \ldots, P_I \in \mathrm{bry}_{y_I}(T_{\vec{w}_I})} \Lambda \prod_{i=1}^{I} P_i.$$

By induction, each term of this sum is in $\langle F_{\Phi,n}, n^c \cdot (h_{\mu'} + 1) \rangle$. Therefore

$$\Lambda((p_{\vec{a}}(\neg\psi_j))\sigma_{\mathcal{T}}) \in \langle F_{\Phi,n}, n^c \cdot (h_{\mu'} + 1) \rangle \subseteq \langle F_{\Phi,n}, n^c \cdot h_\mu \rangle \tag{11}$$

for all possible choices of $j, \vec{a}$. (If p is 0 then the induction fails, as then there would be no trace to μ'. However, in this case, $\Lambda(p\sigma_{\mathcal{T}}) = 0$ and is trivially in $\langle F_{\Phi,n}, n^c \cdot h_\mu \rangle$.)

By hypothesis, $1 \in \langle F_{\Psi,m}, m^{c_2} \rangle$ for suitable constant c_2. This implies that $1 \in \langle (F_{\Psi,m})\sigma_{\mathcal{T}}, m^{c_2} + n^{c_3} \rangle$ after the substitution of the $\lambda_{\mathcal{T},\vec{w},y}$'s for variables, where the constant c_3 is chosen so that n^{c_3} bounds the degree of the $\lambda_{\mathcal{T},\vec{w},y}$'s. This can be expressed, with polynomials f_i and q_i, as

$$1 = \sum_{g_i \in Tot_{\Psi,m} \cup Func_{\Psi,m}} f_i(g_i\sigma_{\mathcal{T}}) + \sum_{p_i \in Failure_{\Psi,m}} q_i(p_i\sigma_{\mathcal{T}}). \tag{12}$$

We have $m \leq n^{c_4}$ for some constant c_4. Each term in (12), and therefore each f_i and q_i, has degree bounded by $n^{c_4c_2} + n^{c_3}$. Multiplying by Λ gives

$$\Lambda = \sum_{g_i \in Tot_{\Psi,m} \cup Func_{\Psi,m}} \Lambda f_i(g_i\sigma_{\mathcal{T}}) + \sum_{p_i \in Failure_{\Psi,m}} q_i\Lambda(p_i\sigma_{\mathcal{T}}).$$

Letting c_5 dominate the exponents for the bounds of Lemmas 4.8 and 4.9, each term in the first summation is in $\langle F_{\Phi,n}, n^{c_1} + n^{c_4c_2} + n^{c_3} + n^{c_5} \rangle$. And, by (11), each term in the second summation is in $\langle F_{\Phi,n}, n^{c_4c_2} + n^{c_3} + n^c \cdot h_\mu \rangle$. With c strictly greater than c_1, c_4c_2, c_3, and c_5, this gives $\Lambda \in \langle F_{\Phi,n}, n^c \cdot (h_\mu + 1) \rangle$. This finishes the proof by induction. □

Similar to the propositional LK case, we can use existing upper and lower bounds on the degree of Nullstellensatz refutations of various principles to prove separation results. The separations (a) and (b) were shown by direct methods in [1]. The separations (c) and (d) are new; they were previously known only with respect to many-one reducibility [7].

Corollary 4.10. (a) PIGEON $\not\leq_{\mathrm{T}}$ OntoPIGEON. *[1]*
(b) PIGEON $\not\leq_{\mathrm{T}}$ LONELY. *[1]*
(c) ITER $\not\leq_{\mathrm{T}}$ OntoPIGEON.
(d) ITER $\not\leq_{\mathrm{T}}$ LONELY.

Proof. It is shown in [1] that $F_{\mathrm{PIGEON},n}$ requires polynomial (in 2^n) degree Nullstellensatz refutations, and [8,12] shows the same for $F_{\mathrm{ITER},n}$. On the other hand, [7] shows that $F_{\Phi,n}$ has polylogarithmic degree Nullstellensatz refutations when Φ is OntoPIGEON or LONELY. Thus the separations hold by Theorem 4.5. □

The results of Section 6 below give an alternate proof of (c) and (d). Namely, Theorem 6.1 implies that Turing and many-one reducibility are equivalent for reductions to either OntoPIGEON and LONELY. From this, the results of [7] for many-one reducibility immediately give (c) and (d).

In addition, note that, since OntoPIGEON $\le_{\mathrm{T}}$ LONELY, one could also prove that ITER $\not\le_{\mathrm{T}}$ OntoPIGEON from ITER $\not\le_{\mathrm{T}}$ LONELY. It is still unknown whether ITER is many-one or Turing reducible to LeftPIGEON or PIGEON. A partial result along this line is shown in [7].

5. Propositional LK reversal

This section proves a converse of Theorem 3.5, namely, that a Turing reduction between NP search problems can be extracted from proofs of the sequents (a), (b), and (c) of Theorem 3.5. In fact, we will prove a slightly stronger result by using a weaker condition (a′) in place of (a), and omitting (b) altogether.

It is worth explaining why the reversal requires there to be proofs of sequents (a) and (b) at all. Or, in other words, why we do not reverse Theorem 3.3 instead of Theorem 3.5. The following two facts are relevant. First, the substitutions σ_i are depth 1.5 and, moreover, they map variables $x_{\vec{w},y}$ to big disjunctions of small conjunctions. Second, the validity of the sequents (a) and (b) means that any $x_{\vec{w},y}$ can also be expressed as $\bigwedge_{y'\neq y}\bar{x}_{\vec{w},y'}$. Thus, σ_i maps the variables $x_{\vec{w},y}$ to conditions that are expressible both as disjunctions of small conjunctions and as conjunctions of small disjunctions. As is well known, this in turn implies that there are polynomial height (height $n^{O(1)}$) decision trees for computing the values $\beta_i(\vec{w})$ of the function $\beta_i : U_m^k \to U_m$ coded by the formulas $x_{\vec{w},y}\sigma_i$. Having these decision trees means there are (perhaps non-uniform) algorithms for computing $\beta(\vec{w})$, and this corresponds to the fact that the Turing reduction between the NP search problems must be polynomial time. (The fact that the LK proofs are uniform will enable us to remove the non-uniformity of the algorithm.)

To illustrate this in a more general setting, we define a notion of "decision tree substitution" that applies to arbitrary propositional variables, not just to the variables of the type $x_{\vec{u},v}$ used above which are required to define a single-valued, total function.[2] Suppose we wish to prove $\Gamma\longrightarrow\Delta$ from substitution instances of non-logical sequents $\Pi_j\longrightarrow\Lambda_j$. A *decision tree substitution* σ is a pair (σ', σ'') of substitutions, such that the following hold.

(i) Both σ' and σ'' are depth 1.5 substitutions. σ' maps variables to big disjunctions of small conjunctions, and σ'' maps variables to big conjunctions of small disjunctions.

(ii) For each propositional variable x, there is an LK proof of

$$\Gamma\longrightarrow\Delta, x\sigma', \bar{x}\sigma''. \tag{13}$$

(iii) For each propositional variable x, there is an LK proof of

$$x\sigma', \bar{x}\sigma'', \Gamma\longrightarrow\Delta. \tag{14}$$

Conditions (ii) and (iii) ensure that either $\Gamma\longrightarrow\Delta$ is true, or $x\sigma'$ is equivalent to $\bar{x}\sigma''$. In particular, when proving $\Gamma\longrightarrow\Delta$, it can be assumed that $x\sigma'$ and $\bar{x}\sigma''$ are equivalent. Since $x\sigma'$ and $\bar{x}\sigma''$ are disjunctions of small conjunctions, this means that there is a small height decision tree for $x\sigma'$.

When the above general notion of decision tree substitution is specialized to variables that code the graph of a function β, the sequents (13) and (14), respectively, become

$$F_{\Phi,n}\longrightarrow G_{\Phi,n}, \left(\bigwedge Tot_{\Psi,m_i}\right)\sigma_i \tag{15}$$

and

$$F_{\Phi,n}\longrightarrow G_{\Phi,n}, \left(\bigwedge Func_{\Psi,m_i}\right)\sigma_i. \tag{16}$$

To understand this, consider letting the variable x in (13) and (14) be $x_{\vec{w},y}$, and letting its negation $\bar{x}$ be $\bigvee_{y'\neq y} x_{\vec{w},y'}$. It is not difficult to check that then the set of sequents (13) and (14) are valid iff the set of sequents (15) and (16) are valid.

Note how (15) and (16) are weaker than the sequents (a) and (b) of Theorem 3.5.

We now state the reversal of Theorem 3.5. It is somewhat stronger than just the converse of Theorem 3.5 for several reasons. First, because sequent (a) is replaced by (a′); second, because sequent (b) is omitted; and, third, because the substitutions σ_i may map variables to disjunctions of conjunctions of literals (as compared to disjunctions of conjunctions of unnegated variables). We require that the LK proofs be uniform: this has the usual meaning that there is a polynomial time algorithm which can calculate the structure and content of any formula in any sequent of the proof based on the path taken in the proof tree from the conclusion of the proof to the sequent.

Theorem 5.1. *Let Φ and Ψ be total $\exists$-sentences. Suppose that there are substitutions $\sigma_1, \ldots, \sigma_r$ such that each σ_i is a depth 1.5 substitution that sends each variable to a disjunction of conjunctions of literals. Further, suppose that there are polynomial time*

[2] We never use decision tree substitutions in this general setting, but it should help motivate the formulation of Theorem 5.1.

uniform LK *proofs of size* $2^{n^{O(1)}}$ *and height* $n^{O(1)}$ *of the following sequents:*

(a′) $F_{\Phi,n} \longrightarrow G_{\Phi,n}, \left(\bigwedge Tot_{\Psi,m_i}\right)\sigma_i$, *for each* $i = 1, \ldots, r$, *and*
(c) $\bigwedge_{i=1}^{r}\left(\bigvee Soln_{\Psi,m_i}\right)\sigma_i, F_{\Phi,n} \longrightarrow G_{\Phi,n}$.

Finally, suppose that the LK *proofs are either cut free, or involve cuts only on formulas of size* $n^{O(1)}$. *Then there is a Turing reduction* M *from* Q_Φ *to* Q_Ψ.

We fix the conventions for the proof of Theorem 5.1. The input to Q_Φ is $(\alpha, 0^n)$. Assume that the arity of α is k and that the arity of solutions to Q_Φ is s, so $Q_\Phi(\alpha, 0^n) \subseteq U_n^s$ and $\alpha : U_n^k \to U_n$. Assume that the arity of functions β input to Q_Ψ is k' and that the arity of solutions to Q_Ψ is s', so that $Q_\Psi(\beta, 0^m) \subseteq U_m^{s'}$ and $\beta : U_n^{k'} \to U_n$. Let $\mathcal{D}_{i,\vec{w},y}$ be the set of disjuncts of $x_{\vec{w},y}\sigma_i$, so that $x_{\vec{w},y}\sigma_i$ is $\bigvee_{P \in \mathcal{D}_{i,\vec{w},y}} P$. Note how $\mathcal{D}_{i,\vec{w},y}$ is playing the role that $\mathrm{br}_y(T^i_{\vec{w}})$ played earlier; however, now the P's can be arbitrary conjunctions and may no longer explicitly correspond to paths in a decision tree.

We will prove Theorem 5.1 by traversing backwards through the proof of sequent (c). When the traversal reaches an inference that introduces a $(\bigvee Soln_{\Psi,m})\sigma_i$ it queries Q_Ψ. The query to Q_Ψ requires a description of a function β_i computed by a polynomial time Turing machine with access to α; for this, Lemma 5.2 shows how to construct β_i by traversing the proof of the sequent (a′). Given that we can define β_i, the traversal of the proof of sequent (c) makes queries to α and $Q_\Psi(\beta_i, 0^{m_i})$ that make formulas on the left true. Therefore, it must eventually make a formula on the right true, which must be a member of $G_{\Phi,n}$, which solves $Q_\Phi(\alpha, 0^n)$ and finishes the reduction.

The proof of Lemma 5.2 defines β_i in terms of an algorithm that traverses the proof of sequent (a′), while making queries to α. These queries will make formulas in the antecedent true. Since the proof is correct, the traversal must eventually visit a sequent in which a formula A in the succedent is made true. If A is a disjunct of $\bigvee_{\vec{u} \in U_{m_i}} \bigvee_{P \in \mathcal{D}_{i,\vec{u},v}} P$, a member of $Tot_{\Psi,m_i}\sigma_i$, then $\beta_i(\vec{u})$ is defined to be v. If A is a disjunct of $[\![\phi_j]\!]_{\vec{a}}$, a member of $Soln_{\Phi,n}$, then the computation of $\beta_i(\vec{u})$ has failed; however, this is not a problem, since in this case we have already solved $Q_\Phi(\alpha, 0^n)$. Lemma 5.2 formalizes this intuition by constructing two functions β_i and γ_i from the proof of sequent (a′) such that if the computation of $\beta_i(\vec{u})$ fails then $\gamma_i(\vec{u})$ computes a solution $Q_\Phi(\alpha, 0^n)$.

Lemma 5.2. *Assume that there are proofs of* (a′) *as in Theorem* 5.1. *Then, for each* $i = 1, \ldots, r$, *there are functions* $\beta_i : U_{m_i}^{k'} \to U_{m_i}$ *and* $\gamma_i : U_{m_i}^{k'} \to U_n^s$ *computed by polynomial time Turing machines with access to* α *with the following properties. Let* $\vec{u} \in U_{m_i}^{k'}$, *and suppose that* $\gamma_i(\vec{u}) = \vec{a} \notin Q_\Phi(\alpha, 0^n)$. *Then, if* $\beta_i(\vec{u}) = v$, *the computation of* $\beta_i(\vec{u})$ *specifies enough of* α *to satisfy some* $P \in \mathcal{D}_{i,\vec{u},v}$.

Proof. We describe an algorithm that calculates the values $\beta_i(\vec{u})$ and $\gamma_i(\vec{u})$ simultaneously. The algorithm traverses a branch in proof of (a′) starting at the endsequent and without backtracking. We construct the traversal mentioned in the preceding paragraph. Let α_t be the partial function defined by the answers to all the α queries the traversal has made after visiting t sequents. Then α_t defines a partial assignment τ_t, where $\tau_t \vDash x_{\vec{w},y}$ (respectively, $\tau_t \nvDash x_{\vec{w},y}$) if $\alpha_t(\vec{w}) = y$ (respectively, $\alpha_t(\vec{w}) = y' \neq y$). As the traversal proceeds, the partial assignment defined by α_t will make certain formulas (called p.s.-settable) in the tth sequent true or false.

Definition 5.3. A formula appearing in the proof of (a′) is *p.s.-settable* provided that it is a subformula of one of the following formulas.

- $\bigvee_{v \in U_n} x_{\vec{u},v}$, for $\vec{u} \in U_n$. (These are subformulas of $F_{\Phi,n}$.)
- $\overline{x}_{\vec{u},v} \vee \overline{x}_{\vec{u},v'}$, for $\vec{u}$, $v \neq v' \in U_n$. (These are also subformulas of $F_{\Phi,n}$.)
- $[\![\phi_j]\!]_{\vec{a}}$, for $j \geq 0$ and $\vec{a} \in U_n$. (These are subformulas of $G_{\Phi,n}$.)
- $P \in \mathcal{D}_{i,\vec{w},y}$, for $\vec{w}, y \in U_{m_i}$. (These are subformulas of $\left(\bigwedge Tot_{\Psi,m_i}\right)\sigma_i$.)
- Any ancestor of a cut formula.

The phrase p.s.-settable means that the formula is settable by specifying a polynomial size (p.s.) part of α. A p.s.-settable formula A is set true (respectively, false) by a partial assignment τ provided each variable $x_{\vec{w},y}$ appearing in A is in the domain of τ, and under this assignment the value of A is true (respectively, false).

The traversal will be defined so that, at the tth sequent, one of the following holds.

(1) Every p.s.-settable formula in the tth antecedent (respectively, succedent) is set true (respectively, false) by τ_t. Furthermore, if $\bigvee_j A_j$ is a p.s.-settable formula in the tth antecedent, then the traversal knows a j such that A_j is set true by τ_t.
(2) The traversal has found j, $\vec{a}$ by step t such that $[\![\phi_j]\!]_{\vec{a}}$ is set true by τ_t. The algorithm then halts, and the value of $\gamma_i(\vec{u})$ is defined to be $\vec{a}$, and $\beta_i(\vec{u})$ is defined arbitrarily.
(3) The traversal has found v by step t such that $P \in \mathcal{D}_{i,\vec{u},v}$ is set true by τ_t. The algorithm then halts, and the value of $\beta_i(\vec{u})$ is defined to be v, and $\gamma_i(\vec{u})$ is defined arbitrarily.

The second sentence of (1) applies to subformulas of $\bigvee_{v\in U_n} x_{\vec{u},v}$; for these formulas, the traversal must know the value v of $\alpha(\vec{u})$ and thus know that $x_{\vec{u},v}$ is true. Note that (1) cannot hold for the initial sequents of the proof; thus the traversal ends at either case (2) or (3), which defines $\beta_i(\vec{u})$ and $\gamma_i(\vec{u})$. Since the proof is polynomial height, the traversal will be polynomial time, and hence β_i and γ_i are polynomial time. From (2) and (3) it is clear that β_i and γ_i have the required properties.

The algorithm breaks into cases on the type of inference. We show the most interesting cases.

$\wedge$**: right**: Let S be $\Gamma \longrightarrow \Delta, A\wedge B$, let S_0 be $\Gamma \longrightarrow \Delta, A$, let S_1 be $\Gamma \longrightarrow \Delta, B$, and let S be derived from S_0 and S_1 by $\wedge$:right. If $A\wedge B$ is not p.s.-settable, then it is a subformula of $\left(\bigwedge Tot_{\Psi,m_i}\right)\sigma_i$ of the form

$$\bigwedge\nolimits_{\vec{w}\in W}\left(\bigvee\nolimits_{y\in U_{m_i}} x_{\vec{w},y}\sigma_i\right),$$

where $W\subseteq U_{m_i}^{k'}$. Since we are attempting to define $\beta_i(\vec{u})$, the traversal moves to S_0 or S_1 according to which of A or B contains the conjunct

$$\bigvee\nolimits_{y\in U_{m_i}} x_{\vec{u},y}\sigma_i.$$

Otherwise, $A\wedge B$ is p.s.-settable. By (1), $A\wedge B$ is set false by τ_t, so the traversal moves to S_0 (respectively, S_1) if A (respectively, B) is set false, and (1) still holds.

$\vee$**: left**: Let S be $\Gamma, A\vee B \longrightarrow \Delta$, let S_0 be $\Gamma, A \longrightarrow \Delta$, let S_1 be $\Gamma, B \longrightarrow \Delta$, and let S be derived from S_0 and S_1 by $\vee$:left. Then $A\vee B$ must be p.s.-settable; therefore the traversal knows that A is set true or B is set true. The traversal moves to S_0 in the former case and S_1 in the latter. Condition (1) still holds.

$\wedge$**: left**: Let S be $\Gamma, A\wedge B \longrightarrow \Delta$, let S_0 be $\Gamma, A, B \longrightarrow \Delta$, and let S be derived from S_0 by $\wedge$:left. The only case with something to show is when A and/or B is p.s.-settable but $A\wedge B$ is not (we only show the case when A is p.s.-settable and B is not; the other cases are similar). This can only arise when A is $\bigvee_{y\in U_n} x_{\vec{w},y}$ or $\overline{x}_{\vec{w},y}\vee\overline{x}_{\vec{w},y'}$, for some $\vec{w}, y, y'\in U_n$. In either case, the traversal queries $\alpha(\vec{w})$, and keeps track of the literal that sets A true. Then (1) holds for S_0.

$\vee$**: right**: Let S be $\Gamma \longrightarrow \Delta, A\vee B$, let S_0 be $\Gamma \longrightarrow \Delta, A, B$, and let S be derived from S_0 by $\vee$:right. Again, the only case with something to show is when A and/or B is p.s.-settable but $A\vee B$ is not (we only show the case when A is p.s.-settable and B is not;, the other cases are similar). This only arises when A is $P\in\mathcal{D}_{i,\vec{u},y}$ or is $[\![\phi_j]\!]_{\vec{a}}$. Either way, A is polynomial size, and the traversal queries the variables in A. If A is set true and A is $[\![\phi_j]\!]_{\vec{a}}$, then case (2) holds. If A is set true and A is $P\in\mathcal{D}_{i,\vec{u},y}$, then case (3) holds. Otherwise, A is set false, and case (1) holds for S_0.

cut: Let S be $\Gamma \longrightarrow \Delta$, let S_0 be $\Gamma \longrightarrow \Delta, A$, let S_1 be $A, \Gamma \longrightarrow \Delta$, and let S be derived from S_0 and S_1 by a cut. Since A is a cut formula it is polynomial size, and the traversal queries all the variables in A. If A is set true (respectively, false) the traversal proceeds to S_1 (respectively, S_0). Then (1) still holds. □

We now prove Theorem 5.1.

Proof of Theorem 5.1. The algorithm for the Turing reduction from Q_Φ to Q_Ψ traverses the proof of the sequent (c) in a manner similar to the traversal of Lemma 5.2. Let τ_t be as in the proof of Lemma 5.2. Expand the notion of p.s.-settable to include subformulas of $(\bigvee Soln_{\Psi,m_i})\sigma_i$. Note that $(\bigvee Soln_{\Psi,m_i})\sigma_i$ is of the form

$$\bigvee\nolimits_{j,\vec{a}}([\![\psi_j]\!]_{\vec{a}})\sigma_i = \bigvee\nolimits_{j,\vec{a}}\bigwedge\nolimits_{h=1}^{H}\bigvee\nolimits_{P\in\mathcal{D}_{i,\vec{w}_h,y_h}} P,$$

where H depends on j and $\vec{a}$. We extend the notion of setting a p.s.-settable formula true to include these new types of p.s.-settable formulas. (We do not need to update the notion of setting a p.s.-settable formula false, since the new types of p.s.-settable formulas appear only in the antecedent.) A p.s.-settable formula A is set true by a partial assignment τ provided that the following hold. If A is an $x_{\vec{w},y}$ (respectively, $\overline{x}_{\vec{w},y}$), then A is set true if $\tau\vDash A$ (respectively, $\tau\nvDash A$). If A is $\bigvee_j A_j$, then A is set true by τ if there is a *known* A_j set true by τ. If A is $\bigwedge_j A_j$, then A is set true by τ if each A_j is set true by τ. For example, $\left(\bigvee Soln_{\Psi,m_i}\right)\sigma_i$ is set true by τ if and only if the traversal knows values $j, \vec{a}$ and knows disjuncts $P_1,\ldots,P_H$ such that, for each h, P_h is in $\mathcal{D}_{i,\vec{w}_h,y_h}$ and τ sets P_h true.

The traversal does a polynomial amount of work at the tth sequent (relative to α and Q_Ψ) and at each step one of the following holds.

(1) If A is in the tth antecedent (respectively, succedent) and is p.s.-settable, then A is set true (respectively, false) by τ_t.
(2) The traversal has found values $j, \vec{a}$ by step t such that $[\![\phi_j]\!]_{\vec{a}}$ is set true by τ_t. The algorithm then halts, and the reduction outputs $\vec{a}$.

The second sentence of (1) requires that the traversal keeps track of both α and Q_Ψ queries. Then when the traversal visits an $\vee$:left inference that introduces a subformula of either $\bigvee_{v\in U_n} x_{\vec{u},v}$ or $\bigvee_{j,\vec{a}}([\![\psi_j]\!]_{\vec{a}})\sigma_i$, it knows how to proceed. It is clear that (1) cannot hold for the initial sequents of the proof, so the correctness of the proof implies that case (2) must eventually hold, at which point the reduction outputs $\vec{a}$.

It remains to show that the traversal preserves (1) and (2). We show only the $\wedge$:left case, as it is the only case that differs significantly from the cases in the proof of Lemma 5.2.

$\wedge$**: left**: Let S be $\Gamma, A\wedge B \longrightarrow \Delta$, let S_0 be $\Gamma, A, B \longrightarrow \Delta$, and let S be derived from S_0 by $\wedge$:left. The new case to consider is when (w.l.o.g.) A is $\left(\bigvee Soln_{\Psi,m_i}\right)\sigma_i$. Use Lemma 5.2 to obtain β_i and γ_i. The traversal queries $Q_\Psi(\beta_i, 0^{m_i})$ and receives answer $\vec{a}\in U_{m_i}$. Let ψ_j be a conjunct of ψ that is made true by the function β_i with the existential variables of Ψ set equal to $\vec{a}$. Let $\llbracket\psi_j\rrbracket_{\vec{a}}$ be

$$\bigwedge_{h=1}^{H} x_{\vec{w}_h, y_h}$$

so that $(\llbracket\psi_j\rrbracket_{\vec{a}})\sigma_i$ is

$$\bigwedge_{h=1}^{H} \bigvee_{P\in\mathcal{D}_{i,\vec{w}_h,y_h}} P.$$

The traversal algorithm does not know the correct value of j, but there are only a constant number J' of values for j, so it can try them all. For a given value of j, the traversal calculates $\gamma_i(\vec{w}_h)$ for each $1\le h\le H$. If there is an h such that $\gamma_i(\vec{w}_h)=\vec{a}\in Q_\Phi(\alpha, 0^n)$, then case (2) holds. Otherwise, for each h, $\beta_i(\vec{w}_h)=y_h$ and enough of α has been specified to make a P in $\mathcal{D}_{i,\vec{w}_h,y_h}$ true. Thus $(\llbracket\psi_j\rrbracket_{\vec{a}})\sigma_i$ is set true, so that (1) holds for S_1. $\square$

6. Many-one versus Turing reductions

This section shows that for many common TFNP classes Turing reducibility is equivalent to many-one reducibility. This includes the classes PPAD, PPADS, PPA, and PLS. On the other hand, we give an example where Turing reducibility does not imply many-one reducibility. It is an open question whether Turing reducibility implies many-one reducibility for the class PPP.

Theorem 6.1. *Let Q_1 be a type-1 or type-2* NP *search problem, and let Q_2 be any of* OntoPIGEON, LeftPIGEON, LONELY, *or* ITER. *Then $Q_1\le_{\mathrm{m}} Q_2$ if and only if $Q_1\le_{\mathrm{T}} Q_2$.*

As a consequence of Theorem 6.1, we get the following corollary.

Corollary 6.2. *Let Q be any of* OntoPIGEON, LeftPIGEON, LONELY, *or* ITER*; then* $\mathrm{C}_{\mathrm{m}}(Q)=\mathrm{C}_{\mathrm{T}}(Q)$.

Therefore, the classes PPAD, PPADS, PPA, and PLS could have been equivalently defined with respect to Turing reducibility. On the other hand, Theorem 6.3 constructs problems for which many-one and Turing reducibility are not equivalent. Hanika [15, Thm 3.12] proves a related conditional separation; however, that result does not apply to NP search problems.

Theorem 6.3. *There exist type-2* NP *search problems Q_1, Q_2 such that $Q_1\le_{\mathrm{T}} Q_2$, but $Q_1\not\le_{\mathrm{m}} Q_2$.*

We first prove Theorem 6.1 and then Theorem 6.3.

Proof of Theorem 6.1. We only consider the case when Q_2 is ITER; the others are similar and are left to the reader. Let M be a Turing reduction from Q_1 to ITER. The intuition is that M makes multiple calls ITER$(g, 0^m)$ and that these can be combined into a single call to ITER$(F, 0^{n^{O(1)}})$, for some appropriate F. The rest of the proof defines F and shows how to use it in a many-one reduction. For simplicity, we assume that Q_1 has no type-1 input.

Without loss of generality, each call to ITER by M has the same size parameter m, since ITER has the instance extension property of Buresh-Oppenheim and Morioka [7]. For notational convenience, assume that solutions to Q_1 are vectors of length 1. Finally, let $p(n)$ be a bound on the runtime of M, and let ε be the empty sequence.

Let $\vec{x}$ be the string input to Q_1. The function F depends on $\vec{x}$, and takes as input $\langle u; y_1,\ldots,y_\ell; v\rangle$, where $u\in U_n$, $y_1,\ldots,y_\ell, v, \in U_m$, and $\ell\le p(n)$. Since F must take strings as arguments, we encode F's input as

$$u1y_11y_2\cdots 1y_\ell 0^{(m+1)(p(n)-\ell)}v\in U_{n+(m+1)p(n)+m}.$$

A 1 in the $(n+(m+1)i+1)$th position $(0\le i<p(n))$ indicates that y_{i+1} is an answer to the $(i+1)$st query to ITER. A 0 in the $(n+(m+1)i+1)$th bit indicates that no $(i+1)$st query to ITER has been made yet. The intended meaning for the inputs is as follows: u is either 0^n or equals the output of M; $y_1,\ldots,y_\ell$ is a valid sequence of answers to the first ℓ queries to ITER made by M; and if $y_1,\ldots,y_\ell$ determine an $(\ell+1)$st call ITER$(g, 0^m)$ by $M(\vec{x})$, then v is an element of the domain of g. A sequence $y_1,\ldots,y_\ell$ is a valid sequence of answers to the first ℓ queries to ITER if, for all $1\le i\le\ell$, $y_1,\ldots,y_{i-1}$ determines an ith query to ITER and y_i is a valid solution to that query. It is clear there is a polynomial time procedure to determine if $y_1,\ldots,y_\ell$ is valid.

If $u\ne 0^n$ or $y_1,\ldots,y_\ell$ not valid, then let $F(\langle u; y_1,\ldots,y_\ell; v\rangle)=\langle u; y_1,\ldots,y_\ell; v\rangle$. Otherwise, let $u=0^n$ and $y_1,\ldots,y_\ell$ be valid. Define $F(\langle u; y_1,\ldots,y_\ell; v\rangle)$ as follows.

1. Suppose that answering the first ℓ calls to ITER with $y_1,\ldots,y_\ell$ causes M to halt and output $a\in Q_1(\vec{x})$. Then let

$$F(\langle u; y_1,\ldots,y_\ell; v\rangle)=\langle a;\varepsilon; 0^m\rangle.$$

2. Suppose that answering the first ℓ calls to ITER with $y_1, \ldots, y_\ell$ causes M to make an $(\ell+1)$st query to ITER. Suppose that the query is ITER$(g, 0^m)$.
   a. If $v \in$ ITER$(g, 0^m)$, then let
   $$F(\langle u; y_1, \ldots, y_\ell; v\rangle) = \langle 0^n; y_1, \ldots, y_\ell, v; 0^m\rangle.$$
   b. If $v \notin$ ITER$(g, 0^m)$, then let
   $$F(\langle u; y_1, \ldots, y_\ell; v\rangle) = \langle 0^n; y_1, \ldots, y_\ell; g(v)\rangle.$$

We now give the many-one reduction M' from Q_1 to ITER using F. On input $\vec{x} \in U_n$, M' returns 0^n if 0^n is a solution to $Q_1(\vec{x})$. The reason for this will be apparent below. Otherwise, M' queries ITER$(F, 0^{n+(m+1)p(n)+m})$ and receives answer $\langle u; y_1, \ldots, y_\ell; v\rangle$. We claim that u is a solution to $Q_1(\vec{x})$. Assuming this claim, M' finishes by outputting u. The rest of the proof shows that the claim holds.

In general, if w is a solution to ITER on input g, then there are three possibilities. Either (1) $w = 0$ and $g(0) = 0$, (2) $g(w) < w$, or (3) $g(w) > w$ and $g(g(w)) = g(w)$. If

$$w = \langle u; y_1, \ldots, y_\ell; v\rangle \in \text{ITER}(F, 0^{n+(m+1)p(n)+m}),$$

we show that the first two cases cannot happen and that, in the third case, u is a solution to $Q_1(\vec{x})$.

Consider computing $F(0)$; note that the input 0 codes the empty sequence of oracle calls to ITER. This sequence either leads M to make a call to ITER or causes M to halt and produce an output a. (Here $a \neq 0$, since 0 was immediately ruled out as a solution.) In the first case, the $(n+1)$st bit of $F(0)$ is 1, and in the second case the first n bits of $F(0)$ are not all zero. Therefore $F(0) \neq 0$.

It is straightforward to check that the coding conventions (specifically that a 1 in the $(n+(m+1)i+1)$th bit indicates a query to ITER) and the fact that 0^n is ruled out as a solution imply that $F(w) \geq w$. Thus case (2) is ruled out.

Thus it must be that $F(w) > w$ and $F(F(w)) = F(w)$ for any solution $w = \langle u; y_1, \ldots, y_\ell; v\rangle$ to ITER$(F, 0^m)$. Let $F(\langle u; y_1, \ldots, y_\ell; v\rangle) = \langle a; b_1, \ldots, b_k; c\rangle$. From the definition of F, $F(\langle a; b_1, \ldots, b_k; c\rangle) = \langle a; b_1, \ldots, b_k; c\rangle$ if and only if $a \neq 0^n$ or $b_1, \ldots, b_k$ is not valid. Suppose that $b_1, \ldots, b_k$ is not valid. Then, by definition of F, there is no string t such that $F(t) = \langle a; b_1, \ldots, b_k; c\rangle$, which is a contradiction, since we could take t to be $\langle u; y_1, \ldots, y_\ell; v\rangle$. Now, suppose that $a \neq 0^n$. F was defined so that in this case a is a solution to $Q_1(\vec{x})$. Thus the claim is proved, and this finishes the proof of the theorem.

A similar argument holds if Q_1 has a type-1 input. □

The proof of Theorem 6.3 is based on the fact that LeftPIGEON $\not\leq_{\mathrm{T}}$ LONELY and LONELY $\not\leq_{\mathrm{T}}$ PIGEON [1]. We will let Q_1 be the problem of solving both LeftPIGEON and LONELY, and Q_2 be the problem of solving either PIGEON or LONELY. Since [1] shows that LeftPIGEON $\leq_{\mathrm{m}}$ PIGEON, it will be clear that two calls to Q_2 can solve Q_1, and hence $Q_1 \leq_{\mathrm{T}} Q_2$. However, being able to solve Q_1 with only a single call to Q_2 is tantamount to having either LeftPIGEON $\leq_{\mathrm{m}}$ LONELY or LONELY $\leq_{\mathrm{m}}$ PIGEON, which is a contradiction. The following proof fills in the details.

Proof sketch of Theorem 6.3. Let Q_1 be the following problem: Given 0^n and $f, g, h : U_n \to U_n$, find $u \in$ LeftPIGEON$(f, g, 0^n)$ and $v \in$ LONELY$(h, 0^n)$. Let Q_2 be the following problem: Given 0^n and $f : U_n \to U_n$, if $f(0) \neq 0$, find $u \in$ PIGEON$(f, 0^n)$, and, if $f(0) = 0$, find $u \in$ LONELY$(f, 0^n)$.

Suppose that M is a many-one reduction from Q_1 to Q_2. Let n be sufficiently large. We show how to specify f, g, h at step i of M's computation such that either a polynomial part of f, g has been specified and f, g do not contain a solution to LeftPIGEON, or a polynomial part of h has been specified and h does not contain a solution to LONELY. Assuming this, M is forced to output a solution for both LeftPIGEON and LONELY, even though, for one of them, there is no solution on the polynomial part of the underlying graph which has been set. Therefore, since M returns an answer involving the unspecified part of the input, M's answer is wrong. Thus M is not a correct reduction, which is a contradiction.

We now show how to set f, g, h. It is clear how to do this if M queries f, g, or h at step i, since M can only make polynomially many queries and there is an exponential search space. Now, suppose that M queries $Q_2(F, 0^m)$. Calculate $F(0)$ by arbitrarily answering queries to f, g, h without solving either LeftPIGEON$(f, g, 0^n)$ or LONELY$(h, 0^n)$. Suppose that $F(0) \neq 0$. Arbitrarily fix f, g and shorten the decision tree for each $F(u)$ to only involve h. Lemma 4 of [1] shows how to specify a polynomial portion of h to correctly answer PIGEON$(F, 0^m)$ without solving LONELY$(h, 0^n)$. If $F(0) = 0$, then arbitrarily fix h and shorten the decision tree for each $F(u)$ to only involve f, g. Then Theorem 6 of [1] shows that there is a way to specify a polynomial part of f and g that solves LONELY$(F, 0^m)$ without solving LeftPIGEON$(f, g, 0^n)$. □

We finish by mentioning a recently obtained improvement to Theorem 6.3. Let $Q_1 \leq_k Q_2$ denote that there is a Turing reduction M from Q_1 to Q_2 such that M makes at most k calls to Q_2. Then Theorem 6.3 proves that there exists Q_1, Q_2 such that $Q_1 \leq_2 Q_2$ but $Q_1 \not\leq_1 Q_2$. In an earlier version of this paper, we conjectured that this result extends to all k.

Conjecture 6.4. *For each $k \geq 2$ there exist Q_1, Q_2 such that $Q_1 \leq_{k+1} Q_2$ but $Q_1 \not\leq_k Q_2$.*

To understand the motivation for this conjecture, fix a prime $p > 2$ and let MOD^p be Q_Φ, where Φ is

$$\exists x[f(x) \neq x \to f^p(x) \neq x].$$

This is a total ∃-sentence since $p > 2$, and the universe being partitioned has size a power of 2. If $p = 2$, then MOD^p is LONELY. It seemed natural that Conjecture 6.4 holds for Q_1 and Q_2 which are combinations of the MOD^p principles defined using the $\otimes$ and & operations defined next.

Given $Q_1(f_1, x_1)$ and $Q_2(f_2, x_2)$, let $(Q_1 \otimes Q_2)(f_1, f_2, x_1, x_2)$ be the following NP search problem: Given f_1, f_2, x_1, x_2 find $y_i \in Q_i(f_i, x_i)$ for $i = 1, 2$. Let y be 0 or 1, and let $(Q_1 \& Q_2)(f_1, f_2, x_1, x_2, y)$ be the following NP search problem: Given f_1, f_2, x_1, x_2, y, find $y_1 \in Q_1(f_1, x_1)$ if y is 0 and find $y_2 \in Q_2(f_2, x_2)$ if y is 1. The symbols $\otimes$ and & are motivated by linear logic, where $\Gamma \longrightarrow A \otimes B$ means that Γ has enough resources to solve both A and B and $\Gamma \longrightarrow A \& B$ means that Γ has enough resources to solve either one of A or B. Theorem 6.3 essentially states that LeftPIGEON $\otimes$ LONELY $\leq_2$ PIGEON&LONELY but LeftPIGEON $\otimes$ LONELY $\not\leq_1$ PIGEON&LONELY.

This next theorem extends this intuition to apply to multiple MOD^p principles for multiple distinct primes p. This suffices to prove that Conjecture 6.4 is true.

Theorem 6.5. *Let $p_1, \ldots, p_{k+1}$ be distinct primes. Then*

$$\mathrm{MOD}^{p_1} \otimes \cdots \otimes \mathrm{MOD}^{p_{k+1}} \leq_{k+1} \mathrm{MOD}^{p_1} \& \cdots \& \mathrm{MOD}^{p_{k+1}}$$

but

$$\mathrm{MOD}^{p_1} \otimes \cdots \otimes \mathrm{MOD}^{p_{k+1}} \not\leq_{k} \mathrm{MOD}^{p_1} \& \cdots \& \mathrm{MOD}^{p_{k+1}}.$$

It is clear that $k + 1$ calls suffice for a reduction between these problems. However, with only k calls, the reduction would have to solve some MOD^{p_i} counting principle by only using MOD^{p_j} principles for $i \neq j$. The intuition is that this cannot happen, since [2] have defined a counting principle related to the MOD^p principles and proved that there are no polynomial size proofs of the counting mod p_i principle from the counting mod p_j principle, for distinct primes p_i and p_j.

The proof of Theorem 6.5 is too long to include in the present paper, but can be found in Johnson [16].

References

[1] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, T. Pitassi, The relative complexity of NP search problems, Journal of Computer and System Sciences 57 (1998) 3–19.

[2] P. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, P. Pudlák, Lower bounds on Hilbert's Nullstellensatz and propositional proofs, Proceedings of the London Mathematical Society 73 (1996) 1–26.

[3] P. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, P. Pudlák, A. Woods, Exponential lower bounds for the pigeonhole principle, in: Proceedings of the 24-th Annual ACM Symposium on Theory of Computing, 1992, pp. 200–220.

[4] P. Beame, T. Pitassi, An exponential separation between the parity principle and the pigeonhole principle, Annals of Pure and Applied Logic 80 (1996) 195–228.

[5] A. Beckmann, S.R. Buss, Separation results for the size of constant-depth propositional proofs, Annals of Pure and Applied Logic 136 (2005) 30–55.

[6] A. Beckmann, S.R. Buss, Corrected upper bounds for free-cut elimination, Theoretical Computer Science 412 (39) (2011) 5433–5445.

[7] J. Buresh-Oppenheim, T. Morioka, Relativized NP search problems and propositional proof systems, in: Proc. 19th IEEE Conference on Computational Complexity, CCC, 2004, pp. 54–67.

[8] S.R. Buss, Lower bounds on Nullstellensatz proofs via designs, in: P. Beame, S. Buss (Eds.), Proof Complexity and Feasible Arithmetics, American Mathematical Society, 1998, pp. 59–71.

[9] S.R. Buss, R. Impagliazzo, J. Krajíček, P. Pudlák, A.A. Razborov, J. Sgall, Proof complexity in algebraic systems and bounded depth Frege systems with modular counting, Computational Complexity 6 (1996/1997) 256–298.

[10] S.R. Buss, J. Krajíček, An application of Boolean complexity to separation problems in bounded arithmetic, Proceedings of the London Mathematical Society 69 (1994) 1–21.

[11] X. Chen, X. Deng, Settling the complexity of two-player Nash equilibrium, in: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, FOCS'06, 2006, pp. 261–272.

[12] M. Clegg, J. Edmonds, R. Impagliazzo, Using the Groebner basis algorithm to find proofs of unsatisfiability, in: Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing, 1996, pp. 174–183.

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

[14] C. Daskalakis, P.W. Goldberg, C.H. Papadimitriou, The complexity of computing a Nash equilibrium, in: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, STOC'06, 2006, pp. 71–78.

[15] J. Hanika, Search problems for bounded arithmetic, Ph.D. Thesis, Charles University, February 2004. Available at Electronic Colloquium on Computational Complexity (ECCC), Theses.

[16] A.S. Johnson, Reductions and propositional proofs for total NP search problems, Ph.D. Thesis, Univ. of California, San Diego, August 2011.

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

[18] J. Krajíček, Bounded Arithmetic, Propositional Calculus and Complexity Theory, Cambridge University Press, Heidelberg, 1995.

[19] N. Megiddo, C. Papadimitriou, On total functions, existence theorems and computational complexity, Theoretical Computer Science 81 (1991) 317–324.

[20] T. Morioka, Classification of search problems and their definability in bounded arithmetic, Master's Thesis, University of Toronto, 2001.

[21] T. Morioka, Logical approaches to the complexity of search problems: proof complexity, quantified propositional calculus, and bounded arithmetic, Ph.D. Thesis, University of Toronto, 2005.

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