UNIVERSITY OF CALIFORNIA, SAN DIEGO

Reductions and Propositional Proofs for Total NP Search Problems

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Mathematics

by

Alan Starz Johnson

Committee in charge:

Professor Samuel R. Buss, Chair
Professor Ery Arias-Castro
Professor Ramamohan Paturi
Professor Jeffrey Remmel
Professor Victor Vianu

2011

The dissertation of Alan Starz Johnson is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____
Chair

University of California, San Diego

2011

EPIGRAPH

Vi veri veniversum vivus vici.

—Doctor Faustus

TABLE OF CONTENTS

Signature Page iii

Epigraph iv

Table of Contents v

List of Figures vii

Acknowledgements viii

Vita and Publications ix

Abstract of the Dissertation x

Chapter 1 Introduction 1

Chapter 2 Preliminaries 20
2.1 Overview 20
2.2 TFNP and TFNP^2 21
2.3 Many-one and Turing Reductions 22
2.4 TFNP^2 Problems and First-order Formulas 24
2.5 The Classes Defined 27
2.6 Propositional LK and Nullstellensatz Proof Systems . . . 29

Chapter 3 Forward Direction, Propositional LK 32
3.1 Overview 32
3.2 Propositional LK Formulation 33
3.3 Main Theorem and Separations 37
3.4 Decision Trees 40
3.5 Proof of Theorem 3.3.1 44

Chapter 4 Nullstellensatz Translation 50
4.1 Overview 50
4.2 Nullstellensatz Formulation 51
4.3 Main Theorem and Separations 55
4.4 Proof of Theorem 4.3.1 58

Chapter 5 Propositional LK Reversal 63
5.1 Discussion 63
5.2 Main Result and Proof 66

Chapter 6 Equivalences Between Many-one and Turing Reductions 75
6.1 Many-one and Turing Equivalences 75

Chapter 7 Separations Between Many-one and Turing Reductions 81
7.1 Overview 81
7.2 k-reducibility and the $\otimes$, &, and $\parr$ Operators 83
7.3 A Separation Between $\leq_{\mathrm{m}}$ and $\leq_{\mathrm{T}}$ 84
7.4 The MOD^d Principles 87
7.5 Infinitely-often Turing Reductions 96
7.6 A $\leq_k$ and $\leq_{k+1}$ Separation 98

Chapter 8 The MOD^d Counting Principles 105
8.1 Overview 105
8.2 Reduction and Separation Proofs 108

Bibliography 120

LIST OF FIGURES

Figure 1.1: The known relationships for some TFNP search classes. This figure is adapted from [2]. 9

Figure 2.1: Γ' and Δ' are sets of formulas such that $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$. . . 31

Figure 6.1: The decision tree $T_{M,n}$ for a Turing reduction M to Q_2. The only solutions to $Q_2(f_1, 0^{m_1})$ that are shown are u and w. 78

Figure 6.2: An illustration of the construction of one ITER instance from all the ITER instances in $T_{M,n}$. The outgoing edges from u and w are crossed out and replaced by the corresponding dotted edges. 79

Figure 6.3: An illustration of the construction of one LONELY instance from all the LONELY instances in $T_{M,n}$. The dotted lines indicate added pairings. 80

Figure 8.1: Some separations and inclusions of TFNP^2 classes in a generic relativized setting. The separation of PPADS from the PMOD^p is conjectured. 107

Figure 8.2: An example of the construction for Theorem 8.2.1 in the case $d = 3$, so there are $2^d - d = 5$ extra copies of U_n. The boxed points are the extra copies of U_n combined with enough copies of 0 to make the number of boxed points divisible by d. The arrows in the first three rows show the behavior of the function F input to MOD^d. 109

ACKNOWLEDGEMENTS

I would like to thank Professor Samuel R. Buss for all the help he has given me as my advisor. His guidance and insight has helped me immensely, and I probably would not be writing this now if not for him. I would also like to thank my friends and family for their constant support through this long and difficult process.

Chapters 2, 3, 4, 5, 6, 7 contain material from the paper "Propositional proofs and reductions between NP search problems" which is currently accepted for publication by the Annals of Pure and Applied Logic. This paper is co-authored by the dissertation author and Samuel R. Buss.

VITA

2006	Bachelor of Arts, University of California, Berkeley
2009	Master of Arts, University of California, San Diego
2011	Doctor of Philosophy, University of California, San Diego

PUBLICATIONS

"Norms of Composition Operators with Rational Symbol", Journal of Mathematical Analysis and Applications, 324 (2006) 1062-1072. Co-authored with Sean Effinger-Dean, Joseph Reed, and Jonathan Shapiro.

"The quantifier complexity of polynomial-size definitions in first-order logic", Mathematical Logic Quarterly, 56, 6 (2010) 573-590. Co-authored with Samuel R. Buss.

"Propositional proofs and reductions between NP search problems", accepted for publication in Annals of Pure and Applied Logic (2011). Co-authored with Samuel R. Buss.

ABSTRACT OF THE DISSERTATION

Reductions and Propositional Proofs for Total NP Search Problems

by

Alan Starz Johnson

Doctor of Philosophy in Mathematics

University of California, San Diego, 2011

Professor Samuel R. Buss, Chair

This dissertation studies TFNP, the class of total NP search problems, and TFNP^2, a relativized version of TFNP. Resolving the exact computational complexities of TFNP and its subclasses is closely tied to the P versus NP question, thus the main focus is on the relativized class TFNP^2. The subclasses of TFNP^2 which we study are "syntactic", meaning that their totality is guaranteed by some combinatorial lemma.

A main result is that there is a strong connection between TFNP^2 and propositional proofs. We show that a Turing reduction from Q_1 to Q_2 gives rise to constant depth, polynomial size propositional (Frege) proofs between the underlying combinatorial lemmas. We show that a similar result holds for polylogarithmic degree Nullstellensatz refutations. These results were shown only for the many-one

case in [9]. These new translations provide new Turing separations by using existing upper and lower bounds from proof complexity. We also solve an open problem stated in [9] by showing that a reverse construction also holds. Namely, we prove that constant depth, polynomial size propositional proofs of the totality of one combinatorial lemma from another combinatorial lemma give rise to a Turing reduction between the corresponding TFNP^2 problems.

Another point of investigation is the relation between many-one and Turing reducibility. We show that for many natural TFNP^2 classes many-one and Turing reducibility are equivalent. To show that this result does not hold in general, we construct a TFNP^2 problem whose Turing closure is strictly larger than its many-one closure. To generalize these results, we introduce a new type of reducibility called k-reducibility, and show that k- and $(k+1)$-reducibility are not equivalent.

This last result makes use of modular counting principles. These have been previously studied as propositional formulas, but we introduce them as TFNP^2 problems called MOD^d. We prove results about the relative complexity of the MOD^d's between themselves and other interesting TFNP^2 classes. By constructing equivalences between the propositional translation of MOD^d and the modular counting principles of [3], we can use the provability results in [3] to prove separations.

Chapter 1

Introduction

Total NP search problems (also called TFNP problems for "Total Functional NP") are search problems such that (1) solutions have polynomial growth rate, (2) solutions can be checked in polynomial time, and (3) a solution is guaranteed to exist [28]. The totality requirement in (3) causes TFNP problems to behave differently than decision problems. As an example, consider the problem FACTOR which is: "Given n, find $1 < d < n$ which divides n, otherwise return n." Note that n is returned when it is prime. By the AKS primality test [1], n is a certificate of its own primality, and thus FACTOR is an NP search problem. Since FACTOR is clearly total, it is a TFNP search problem. However, there does not seem to be a nice way to formulate FACTOR as a decision problem. There is nothing to be decided since it is always true that n is prime or has a prime factor. In contrast to decision problems, the computational complexity of FACTOR arises from the totality of the search problem.

At this point we note that the formal definition of TFNP is given in Chapter 2. The point of this chapter is to give a broad motivation for the remaining chapters as well as building intuition for the concepts. Thus this chapter contains no formal definitions at all, but will instead direct the reader to the appropriate locations.

The search problem FACTOR is based on an $\mathrm{NP} \cap \mathrm{coNP}$ decision problem. In fact, any $\mathrm{NP} \cap \mathrm{coNP}$ decision problem gives rise to a total NP search problem. To see this, let $L \in \mathrm{NP} \cap \mathrm{coNP}$. Then the TFNP search problem corresponding to

L is: "Given x, return y that certifies that $x \in L$, otherwise return z that certifies $x \notin L$." There is also a reverse construction taking a TFNP search problem Q with a *unique* solution and producing an equivalent NP $\cap$ coNP decision problem. The equivalent decision problem is simply the problem which decides whether the i^{th} bit of the unique solution to Q is 0 or 1. Thus NP $\cap$ coNP is equivalent to those TFNP problems with unique solutions.

However, TFNP contains many problems which do not have unique solutions. For example, the general pigeonhole principle PHP_n^m, for $m > n$, can be formulated as a TFNP problem: "Given a (polynomial time) mapping from m pigeons to n holes, find two different pigeons mapping to the same hole." It is clear that PHP_n^m does not have a unique solution. In fact, it is shown in [2] that a relativized version of PHP_n^m is not equivalent to any decision problem. Indeed, one reason to study TFNP at all is that it contains so many natural problems which do not have nice formulations as decision problems. The (relativized) problems that we focus on below (including SOURCE.or.SINK, SINK, MOD^d, PIGEON, ITER) all have the property that they are not equivalent to any decision problem.

Even though TFNP cannot be characterized in terms of NP $\cap$ coNP *decision* problems, it can be characterized as NP $\cap$ coNP *search* problems. To see this, consider $F(\text{NP} \cap \text{coNP})$, the set of NP $\cap$ coNP search problems from [28]. An $F(\text{NP} \cap \text{coNP})$ search problem has two polynomial time predicates P_1, P_2 such that for every x there is a y with $P_1(x, 0y)$ or there is a z with $P_2(x, 1z)$; the extra bit prepended to y or z determines which relation holds. It is easy to show that $F(\text{NP} \cap \text{coNP}) = \text{TFNP}$. For one direction, note that any TFNP problem is in $F(\text{NP} \cap \text{coNP})$ since only one of P_1 and P_2 needs to be used. To show the containment $F(\text{NP} \cap \text{coNP}) \subseteq \text{TFNP}$, let Q be in $F(\text{NP} \cap \text{coNP})$. Then Q is a TFNP problem, where the validity of a solution w is checked by running P_1 or P_2, depending on the first bit of w.

The class TFNP contains FP, the set of search problems solvable in polynomial time, and is contained in FNP, the set of (possibly non-total) NP search problems; that is FP $\subseteq$ TFNP $\subseteq$ FNP. The first inclusion is conjectured to be strict, with FACTOR as a problem in TFNP but not in FP. The second inclusion

is trivially strict, as not all FNP problems are total. However, there is a stronger result due to [28] which states that there is a TFNP problem that is complete for FNP if and only if NP = coNP. This result provides evidence that TFNP does not capture all of the computational complexity of FNP. The class FNP and other subclasses consisting of partial functions have been studied by Selman [35, 36]; Selman's work is expanded to total functions in [19].

The main reason to study TFNP is to understand problems, such as PHP_n^{n+1}, whose complexity cannot formulated by decision problems. Since a decision-like question such as "Does a solution exist?" is always answered "Yes" by a TFNP problem, we are concerned with *why* the answer is always "Yes."

At the heart of any TFNP problem is a principle which justifies its totality, and we can ask questions about the strength of that principle. For example, can we relate PHP_n^{n+1}, the standard pigeonhole principle, to $\mathrm{PHP}_n^{n^2}$, the weak pigeonhole principle? It can be shown that the standard pigeonhole principle is stronger than the weak pigeonhole principle (in a relativized setting, this is expanded on near the end of this chapter). By understanding the relation among TFNP problems we can understand the relative strength of different lines of reasoning.

TFNP is a good setting for understanding the strength of different problems because it contains a vast number of interesting principles. One example is the class PLS, introduced by Johnson, Papadimitriou and Yanakakis [22], of polynomial local search problems. A PLS problem has a (polynomial time) local search heuristic which respects a positive integer valued cost function; a solution to a PLS problem is a local minimum with respect to its cost function. A PLS problem is total since such a local minimum must exist. A typical instance of PLS has a size parameter n that specifies a search space of size 2^n, so that a brute force search is not possible in polynomial time. Interestingly, PLS has other equivalent characterizations, including finding a local optimum for the Lin-Kernighan heuristic for the traveling salesman problem, or in terms of finding an input to a Boolean circuit that produces a locally minimum value [22]. For more early work on PLS, including more PLS complete problems, see [27, 33, 34].

In general, TFNP problems have a nice relationship with combinatorial

lemmas via their totality requirement. For example, the fundamental theorem of arithmetic guarantees the totality of FACTOR, and the pigeonhole principle guarantees the totality of PHP_n^m. In fact, any TFNP problem can be thought as expressing the principle which guarantees its totality. In this way, studying the relative complexity of TFNP is essentially studying the relative computational power of combinatorial lemmas.

The exact complexities of important problems in TFNP, like FACTOR, are still unknown, and this question is not expected to be resolved soon. The next best thing is to study the relative complexity of TFNP problems. The relation between two TFNP problems can be thought of as a statement about the relative power of combinatorial principles. Thus the equivalence between PLS and finding a local optimum for the Lin-Kernighan heuristic shows that the combinatorial lemmas underlying these classes are equivalent.

The most common way to create a TFNP problem is to start with a combinatorial lemma and define a corresponding search problem whose totality is based on that lemma. Papadimitriou introduces many TFNP classes in this manner [31]. For example, Sperner's Lemma guarantees the existence of a panchromatic complex in a triangulation of a trichromatic triangle; the goal of the corresponding TFNP problem SPERNER is to find such a panchromatic complex. Other TFNP problems include NASH, based on the existence of Nash equilibria, and LEAF, based on the fact that a graph with degree at most two has an even number of leaves.

Interestingly, there are equivalences between seemingly very different TFNP classes. For example, the problem SOURCE.or.SINK from [2] is "Given a directed graph where each vertex has in- and out-degree at most one and a given source vertex with out-degree one, find another vertex with total degree one." The many-one closure of SOURCE.or.SINK defines the class PPAD. Surprisingly, SOURCE.or.SINK is equivalent to SPERNER, [31], and NASH, which imply that these latter two problems are also complete for PPAD. The equivalence with NASH was shown in the case with three of more players by [18], and the two player case was shown by [15]. These equivalences show that the underlying computational nature of calculating Nash equilibria, whose computational nature is unclear at first

glance, is equivalent to the easily understandable problem SOURCE.or.SINK. By studying TFNP we hope to come to a similar understanding of other combinatorial lemmas.

There are many other problems complete problems for PPAD that arise from a wide variety of areas, including internet routing and economics [24]. A compendium of PPAD complete problems is maintained by [23].

The definition of equivalence between two TFNP problems is based on a many-one reduction for TFNP problems as defined by Megiddo and Papadimitriou [28]. Let Q_1, Q_2 be TFNP problems. Then Q_1 is many-one reducible to Q_2, written $Q_1 \leq_{\mathrm{m}} Q_2$, provided there are polynomial time functions f, g such that if y is a solution to Q_2 on input $f(x)$, then $g(y)$ is a solution to Q_1 on input x. While not specifically mentioned by Megiddo and Papadimitriou, their concept of many-one reduction can be generalized to allow asking a (polynomially long) sequence of calls to Q_2. Such a reduction is called a *Turing* reduction, denoted $\leq_{\mathrm{T}}$. For example, suppose we are trying to solve Q_1 on input x. Then after getting a solution y_1 to Q_2 on input $f_1(x)$, another instance of Q_2 can be specified by a new function f_2, which depends both on x and y_1. After finding a y_2 solving Q_2 on $f_2(x, y_1)$, the process repeats. Finally the reduction computes (in polynomial time) a solution to Q_1 on input x, based on all the answers to instances of Q_2 that it received.

Megiddo-Papadimitriou reducibility is the natural definition of a many-one reduction in the TFNP setting. In particular, $\leq_{\mathrm{m}}$ is transitive, and if $Q_1 \leq_{\mathrm{m}} Q_2$, and $Q_2 \in$ TFNP, then $Q_1 \in$ TFNP. However, there do not seem to be any TFNP complete problems with respect to $\leq_{\mathrm{m}}$ (or any other reasonable notion of reducibility). The difficulty arises in the totality condition. To better understand this, recall that FNP is the set of all (possibly non-total) NP search problems. The class FNP contains many complete problems. For example, the search problem version of any NP complete problem is FNP complete. One way to interpret this is that we can enumerate all nondeterministic Turing machines. However, for a similar result to hold for TFNP we would need to enumerate the nondeterministic Turing machines such that there is an accepting computation on all inputs. Since no such enumeration is known or expected, we do not expect there to be any TFNP

complete problems.

The lack of TFNP complete problems is another motivation for defining subclasses of TFNP based on combinatorial lemmas, since such classes do have complete problems. As an example, consider the TFNP problem PHP_n^{n+1} where the mapping from pigeons to holes is given by f. The requirement that f map $n+1$ pigeons to n holes can be easily enforced syntactically, and thus the pigeonhole principle guarantees the totality of the search problem. A subclass of TFNP can be created by taking the many-one closure of PHP_n^{n+1}; note that PHP_n^{n+1} is trivially a complete problem for this subclass. In general, classes whose totality is based on a combinatorial lemma (such as PHP_n^m) are sometimes called "syntactic" subclasses of TFNP, since the totality requirement can be enforced through the syntactic form of the input.

A more general notion of reducibility, called "type-2" reducibility, is introduced in [2]. To motivate the phrase "type-2", consider a general TFNP problem. This problem will be described by a string (a type-0 object), which is part of the input, and functions (type-1 objects), which are not part of the input. In the case of PLS the string input encodes the instance, and there are polynomial time functions which take a point and determine its cost, neighbor, and feasibility. The relativized class TFNP^2 generalizes TFNP by allowing problems to take both strings and functions as inputs; these problems are type-2 since they take type-1 objects (functions) as inputs. The function inputs are accessed as oracles. The superscript 2 on "TFNP^2" indicates the class contains type-2 TFNP problems.

The advantage of the type-2 approach is that each problem in an unrelativized TFNP class can be expressed as an instantiation of a single TFNP^2 problem. For the classes we are interested in, this canonical type-2 problem has a clear and simple statement. Another reason to use type-2 reducibility is that separations and reductions using type-2 reducibility give rise to oracle separations and inclusions of the corresponding unrelativized TFNP classes.

As an example of defining a TFNP class from a canonical TFNP^2 problem, consider PLS. The class PLS can be defined from the type-2 problem ITER, which is defined as follows. The inputs to ITER are a size parameter n (which can be taken

as the length of the string input) and a function f mapping n-bit strings to n-bit strings, and the solutions are those strings u such that (1) $f(0) = 0$, (2) $f(u) < u$, or (3) $f(u) > u$ and $f(f(u)) = f(u)$. It is clear that ITER is total, and that each solution to ITER can be verified in polynomial time (in fact, two queries to f suffice), where each call to the oracle f counts as a single step. Let Q be in TFNP. A many-one reduction from Q to ITER sets up the size and function parameters to ITER, solves ITER on those parameters, and then produces a solution to Q in polynomial time. More specifically, Q is many-one reducible to ITER if there are polynomial time functions f, g, h such that $h(x, y)$ is a solution to Q on input x for any y that is a solution to ITER on size parameter $g(x)$ and with function input $z \mapsto f(x, z)$. It can be shown that PLS consists of exactly those TFNP problems many-one reducible to ITER.

This example shows how a TFNP problem can be reduced to a TFNP^2 problem. There is also a notion of reducibility between two TFNP^2 problems. It is essentially the same as before, except each function gets oracle access to the functions input to Q_1. Specifically, suppose Q_1, Q_2 are two TFNP^2 problems, where Q_1 takes function input f_0 and string input x. Then Q_1 is many-one reducible to Q_2 if there are polynomial time functions f, g, h relative to f_0 such that $h(f_0, x, y)$ is a solution to Q_1 on input f_0 and x for any y that is a solution to Q_2 on function input $z \mapsto f(f_0, x, z)$ and string input $g(f_0, x)$. Type-2 reducibility is defined formally in Section 2.3.

Whereas the purpose of reducibility from a TFNP problem to a TFNP^2 problem is to define subclasses of TFNP, reducibility between TFNP^2 problems gives oracle results for subclasses of TFNP by the following three equivalent conditions for $Q_1, Q_2, \in$ TFNP^2 [17]: (1) Q_1 is many-one reducible to Q_2 (2) for any oracle G the many-one closure of Q_1 relative to G is contained in the many-one closure of Q_2 relative to G (3) there is a generic oracle G such the many-one closure of Q_1 relative to G is contained in the many-one closure of Q_2 relative to G. The main advantage of type-2 reducibility is that it allows us to prove separation results without resolving difficult open questions about computational complexity. Namely, if P = NP, then all TFNP problems can be solved in polynomial time.

This means that a separation between two TFNP classes implies P $\neq$ NP. Our separations involve the relativized search problems of TFNP[2], and thus do not resolve P versus NP.

Our separation results are all relative to generic oracles. However, other authors have proved results for different kinds of oracles. As an example, [19] introduces Proposition Q, which states that TFNP is "easy." More precisely, a polynomial time predicate $R(x, y)$ defines a TFNP problem if there is a polynomial p such that $R(x, y)$ implies $|y| \leq p(|x|)$, and for each x there is a y such that $R(x, y)$ holds. Proposition Q says that for all R and p which define a TFNP problem, there is a polynomial time function f such that $R(x, f(x))$ holds. [8] shows there is an oracle A such that Proposition Q is true relative to A, but the polynomial hierarchy is proper relative to A. Furthermore, P $\neq$ UP relative to A, where UP is the class of decision problems such that solutions can be verified in polynomial time and the verifier accepts at most one solution.

Four TFNP classes of particular interest are PPA, PPAD, PPADS, and PPP [2]; these classes are defined in Section 2.5. The complete problems for most of these classes have already been introduced. The class PPA (for "polynomial parity argument") is the many-one closure of the problem LONELY, which is based on the fact that an undirected graph of degree at most 2 has an even number of odd degree nodes. The class PPAD (for "polynomial parity argument, directed version") is the many-one closure of the problem SOURCE.or.SINK. The class PPADS (for "polynomial parity argument, directed version, sink only") is the many-one closure of the problem SINK, which is exactly like SOURCE.or.SINK, except that the solutions are required to be sinks. The class PPP (for "polynomial pigeonhole principle") is the many-one closure of the type-2 problem PIGEON, which is "Given a function f from n-bit strings to n-bit strings, return the 0 string if $f(0) = 0$ otherwise return $x \neq y$ such that $f(x) = f(y)$." Note that if $f(x) \neq 0$ for all x, then the pair $x \neq y$ is guaranteed to exist by the pigeonhole principle.

Figure 1.1 shows the relative complexity of these four classes relative to a generic oracle. In particular, there are no many-one reductions between the classes other than those shown, so that PPA is not contained in PPADS or PPP, and

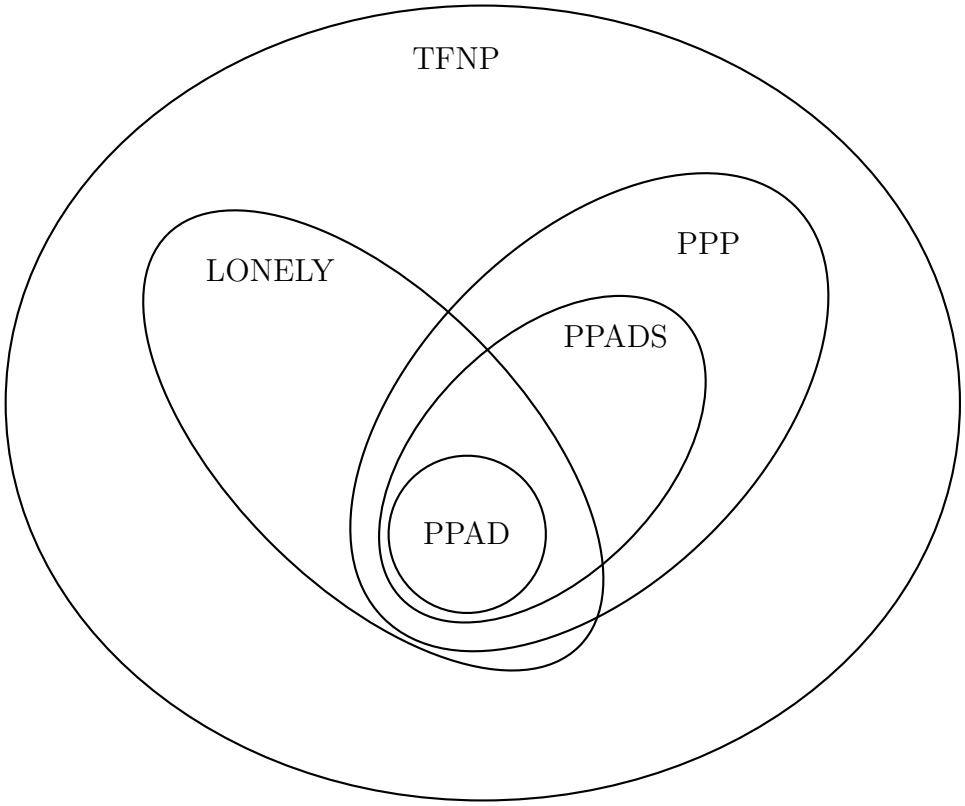

Figure 1.1: The known relationships for some TFNP search classes. This figure is adapted from [2].

PPADS is not contained in PPA. When a reduction exists, it is quite simple. For instance it is trivial that PPAD is contained in PPADS, since any solution to SINK is, in particular, a solution to SOURCE.or.SINK. As another example, it is possible to show that PPAD is contained in PPA by ignoring the directions in the underlying directed graph.

The separations in Figure 1.1 are slightly stronger than stated above because they are proved with respect to Turing reductions, as opposed to many-one reductions. As with Megiddo-Papadimitriou reducibility, the only difference between a type-2 many-one reduction from Q_1 to Q_2 and a type-2 Turing reduction from Q_1 to Q_2 is that in the many-one case the reduction is only allowed to make one query to Q_2, whereas polynomially many calls can be made to Q_2 in the Turing case. Note that there are two kinds of oracle calls that a type-2 reduction (many-one or

Turing) can make: those calls made to the functions input to Q_1 and those calls made to Q_2.

Decision trees are the main tool used to prove the separations in Figure 1.1 [2, 3, 9]. Recall that if $Q_1, Q_2 \in \text{TFNP}^2$ and Q_1 is many-one or Turing reducible to Q_2, then the reduction must specify the string and function inputs for an instance of Q_2. Say Q_1 has one function f as input, and let F be a particular function input to Q_2 set up by the reduction. This means that there is an oracle Turing machine M_F with oracle access to f which computes the values of F. In particular, the computation of M_F on input u can be described by a decision tree T_u, whose internal nodes correspond to f queries during the computation of $F(u)$ and whose leaves correspond to the computed value of $F(u)$; for more details on the construction of decision trees see Section 3.4. A separation $Q_1 \not\leq_{\text{T}} Q_2$ can be shown by assuming such a reduction exists, and then deriving a contradiction by arguing about the decision trees for the functions input to Q_2 queries.

As an example, consider proving the Turing separation LONELY $\not\leq_{\text{T}}$ PIGEON. For a contradiction, suppose that there is a Turing reduction computed by a Turing machine M. We show that it is always possible to answer each query made by M without revealing a solution to LONELY. Once this is shown, M is forced to output an answer to LONELY without explicitly knowing one, which contradicts the correctness of M.

We outline how M can continue its computation without solving LONELY. The problem LONELY takes one function as input, call it f. Consider a particular query to PIGEON made by M, and suppose the function input to PIGEON is F, which is computed in polynomial time relative to f. For M to continue, there must be a way to set the values of f, without solving LONELY, such that there is either a u such that $F(u) = 0$ or a pair $u \neq v$ such that $F(u) = F(v)$. Because we are trying to avoid solutions to LONELY, we prune the branches in all the decision trees for F whose edge labels give a solution to LONELY. Then M can continue if (1) there is some branch labeled 0, or (2) if there are branches P_1, P_2 in $T_u, T_{u'}$, respectively, for $u \neq u'$ such that the previously known values of f combined with the values of f determined by the edge labels of P_1 and P_2 do not solve LONELY. The proof,

which we do not detail further, then derives a contradiction from the failure of (1) and (2) [2]. A similar approach using decision trees is used to prove the separation PIGEON $\not\leq_{\mathrm{T}}$ SINK [2].

The relation of the class PLS to the four classes of Figure 1.1 is studied in [29]. There it is shown that SOURCE.or.SINK is not Turing reducible to ITER, which implies that PLS contains none of PPAD, PPADS, PPA, and PPP in a generic relativized setting. The proof follows the decision tree separation techniques of [2]. The resolution of the reverse inclusions was taken up in [9], where it was shown that ITER is not many-one reducible to LONELY. Thus the only open relation left among the classes PPAD, PPADS, PPA, PPP, and ITER is whether ITER is many-one or Turing reducible to PIGEON.

An interesting aspect of the proof that ITER is not many-one reducible to LONELY is that it uses known results from proof complexity. Broadly speaking, a type-2 TFNP problem can be encoded as a purely existential, first-order formula. A particular instance of this problem then becomes a constant depth propositional formula. A many-one reduction between type-2 TFNP problems becomes a constant depth, polynomial size proof between these propositional formulas. Therefore, the many-one reduction translates into a constant depth, polynomial size proof relating the underlying combinatorial principles which justify the totality of the TFNP^2 problems. The formalization for propositional proofs is given in Section 2.6.

It is also possible to perform this translation instead using polynomials in the Nullstellensatz calculus. The Nullstellensatz calculus is a refutation system involving polynomials, and is described in Section 2.6. Fix a base field F, and consider a set of polynomials $\{f_i\} \subset F[x_1, \ldots, x_k]$ that express, when simultaneously set to 0, an unsatisfiable set of propositional conditions. Assume that $\{f_i\}$ contains $x_i^2 - x_i$ for each i so that any simultaneous solution $a_1, \ldots, a_k$ to the f_i's implies each a_i is 0 or 1, and hence each $a_i \in F$. Then the weak form of Hilbert's Nullstellensatz implies that there are polynomials $\{g_i\} \subset F[x_1, \ldots, x_k]$ such that $\sum_i g_i f_i = 1$. The g_i's are a proof that the f_i's can not be simultaneously set to 0, since otherwise $1 = 0$. The complexity of a Nullstellensatz refutation is measured by the maximum degree of $g_i f_i$. The method of [9] translates a many-one reduction

$Q_1 \leq_{\mathrm{m}} Q_2$ into a low degree Nullstellensatz refutation of Q_1 from Q_2. Upper and lower degree bounds for Nullstellensatz refutations of different principles have been extensively studied [2, 10, 16]. These bounds can be immediately used to show a many-one separation between the corresponding TFNP problems. In particular, [9] uses this idea to prove the separation ITER $\not\leq_{\mathrm{m}}$ LONELY. This idea is also central to the separation SINK $\not\leq_{\mathrm{T}}$ LONELY proved in [2].

A useful feature of Nullstellensatz refutations is that it is often possible to choose a convenient base field. Correctly picking the characteristic of the base field can greatly simplify the structure of the involved polynomials. For example, if the base field has characteristic 2, then pairs of identical terms in a polynomial can be eliminated and any constant term is either 0 or 1. These properties are useful when applied to LONELY, which is a kind of parity argument.

One open question asked by [9] is whether the translations of many-one reductions into constant depth propositional proofs and low degree Nullstellensatz refutations also apply to Turing reductions. We resolve this question in Chapters 3 and 4 where it is shown that this translation carries through for Turing reductions for propositional and Nullstellensatz proofs, respectively. This extended result immediately gives the stronger separation ITER $\not\leq_{\mathrm{T}}$ LONELY. A key construction needed to in the proof of the two translations in the case of a Turing reduction is the decision tree for the machine M calculating a Turing reduction. This decision tree differs from the ones previously mentioned because M can make oracle calls either to a function or a TFNP^2 problem. It is straightforward to extend the definition of a decision tree to include the two different kinds of oracle calls. However, there is a complication with the branching for nodes corresponding to a query to a TFNP^2 problem. For example, suppose $Q_1, Q_2 \in \mathrm{TFNP}^2$ and that $Q_1 \leq_{\mathrm{T}} Q_2$. Suppose the function input to Q_1 is f, and that the reduction makes a call to Q_2 on input F. Let μ be the node corresponding to this Q_2 call in the decision tree. Note that F is polynomial time computable relative to f, and that at the time of the call to Q_2, polynomially many values for f are known. Any solution y to Q_2 on input F can be verified to be correct in polynomial time, and corresponds to a child of μ. Since F is polynomial time computable relative to f, such a verification process reveals

polynomially many more values for f. However, there may be some y for which the verification process conflicts with the earlier known values of f. Paths through such y must be somehow pruned from the tree. Section 3.4 introduces a type of path through a decision tree with f and Q_2 queries called a *trace* that avoids such problematic y's. Traces are the main technical tool used to prove the translations of Turing reductions into propositional and Nullstellensatz proofs.

It is conceivable that more TFNP^2 separations could be proved by attempting a translation into other proof systems. Generally speaking, any translation of the reduction $Q_1 \leq_{\mathrm{T}} Q_2$ into a proof system P should have the property that if the translation of Q_2 is "easy" for P, then the translation of Q_1 is "easy" for P. This is exactly what happens for propositional proofs and Nullstellensatz refutations. However, finding a proof system with the necessary properties to prove a particular separation can have its own problems. For example, it is unknown if ITER is many-one or Turing reducible to PIGEON. The translation technique for propositional proofs and Nullstellensatz refutations fails to resolve this question since PIGEON is hard for both these systems [2, 4]. Thus to prove that ITER $\not\leq_{\mathrm{T}}$ PIGEON via a translation, one needs to find a proof system in which PIGEON is easy but ITER is hard. So far, no such proof system has been found.

Another open question posed by [9] is whether the translations of TFNP^2 problems into propositional and Nullstellensatz proofs can be reversed. That is, given proofs of the totality of one TFNP^2 problem from the totality of another TFNP^2 problem, is it possible to construct a Turing reduction between the two problems? Chapter 5 answers this question in the affirmative, and is an even stronger result because several of the conditions present in the forward translation can be relaxed. Thus there is a kind of equivalence between constant depth propositional proofs and Turing reductions between TFNP^2 problems.

There are a number of issues to overcome when reversing the translation into propositional proofs. The first problem is the precise statement of the reversal. In the forward direction, the propositional proof of the totality of Q_1 on size parameter n from instances of the totality of Q_2 has formulas with size $2^{n^{O(1)}}$ and has $2^{n^{O(1)}}$ many nodes in its treelike representation. Therefore, standard assumptions must

be made on the uniformity of the proofs for the reversal. Namely, we assume there are polynomial time functions describing the proof, including: accessing the root of the proof, finding the parent sequents of a given sequent, finding the principal formula of an inference, determining the syntactic structure of a formula.

Another problem with formulating the reversal is to decide what kinds of substitutions are allowable. Recall that the proofs given by the translation of Chapter 3 use substitution instances of the totality of Q_2. However, these substitutions have a very special form. A query to Q_2 passes a function F as input, where F is polynomial time computable relative to f, the function input to Q_1. The substitutions constructed in the translation replace a variable $x_{\vec{w},y}$ expressing $F(\vec{w}) = y$ with a disjunction of paths from the decision tree for $F(\vec{w})$ that output y. Therefore, we expect that in the reverse direction the substitution instances of the totality of Q_2 arise from decision trees. Chapter 5 defines a "decision tree" substitution intended to capture those substitutions expressible by decision trees. In fact, decision tree substitutions are more general than the substitutions used in the forward translation. This is one way in which the reversal is stronger than just undoing the forward translation.

Once properly formulated, there are still issues to be resolved with the proof of the reversal. A Turing reduction is obtained from propositional proofs by constructing a backwards traversal (with no backtracking) of a treelike representation of propositional proofs. Because of this, the height of the propositional proof on size parameter n must be $n^{O(1)}$ for the traversal to be polynomial time computable. The previous translation of [9] did not give a height bound on the proofs; however, Chapter 3 shows that the proofs do indeed have height $n^{O(1)}$.

This backwards traversal is not enough by itself to construct a Turing reduction from Q_1 to Q_2. The reason is that such a Turing reduction must make queries to Q_2, and these queries require a function F as input. To define F, we require side proofs of the totality of the decision tree substitution. A different traversal of these proofs yields a procedure for F, which will have runtime bounded by the height of the proofs. As before, these proofs were constructed in [9] without bounds on the height. Again, Chapter 3 proves that they actually have height $n^{O(1)}$,

which implies that F is polynomial time.

There are two ways in which these side proofs are more general that what are produced in the forward translation. One is that the side proofs for F only need to define the totality of F, and not the functionality of F. The reason for this is based in the formulation of TFNP^2 problems from Chapter 2. While the specifics follow [9] closely, there is one new convention we adopt that makes functionality generally unimportant. This is explained in detail in Chapter 5. Another difference in the reversal is that the side proofs for F can be weakened so that the computation of $F(u)$ can fail and instead find a solution to Q_1. This sort of behavior is not even allowed in the forward direction.

Once the reversal has been proved for propositional proofs, it is natural to ask whether a similar result holds in the Nullstellensatz setting. However, such a result is not expected since, if there were, propositional and Nullstellensatz proofs would be equivalent through the equivalence between propositional proofs and Turing reductions. Chapters 4 and 5 discuss why it seems unlikely that a Nullstellensatz reversal can even be well formulated.

The prior known reductions between TFNP^2 classes have all been many-one reductions [2]: the extra power of making multiple queries to TFNP^2 problems is never needed. With the extended translations of Chapters 3 and 4 to Turing reductions, all known separations are with respect to Turing reductions. It is then natural to ask whether there is a difference between many-one and Turing reductions. That is, are there Q_1, Q_2 such that $Q_1 \leq_{\mathrm{T}} Q_2$ but $Q_1 \not\leq_{\mathrm{m}} Q_2$? This is equivalent to the existence of a Q whose Turing closure is larger than its many-one closure. Turing reductions are seemingly stronger than many-one reductions, however Chapter 6 shows that the Turing and many-one closures are the same for the classes PPA, PPAD, PPADS, and PLS. It is an open question whether PPP is closed under Turing reductions or not. In fact, it is open whether allowing even two calls to PIGEON is stronger than only allowing one call to PIGEON.

The equivalence of the Turing and many-one closures for the classes mentioned above adds more evidence for the equivalence of many-one and Turing

reductions. However, Chapter 7 produces a TFNP^2 problem whose many-one and Turing closures are distinct. The proof revolves around the binary operations $\otimes, \&, \parr$ on TFNP^2 problems introduced in Section 7.2. Specifically, suppose Q_1, Q_2 are two TFNP^2 problems. Then $Q_1 \otimes Q_2$ is the problem of simultaneously solving both an instance of Q_1 and an instance of Q_2. The problem $Q_1 \& Q_2$ is the problem of solving a user specified choice of one of Q_1 and Q_2. The problem $Q_1 \parr Q_2$ is the problem of solving Q_1 or Q_2, where the user cannot specify which one to solve. The notation for these operators is chosen to mimic properties from linear logic. For example, $\Gamma \longrightarrow A \otimes B$ can be interpreted as the statement that Γ has enough resources to solve both A and B. The sequent $\Gamma \longrightarrow A \& B$ has the interpretation that Γ has enough to solve A or B individually, but may not be able to simultaneously solve A and B. Lastly, $\Gamma \longrightarrow A \parr B$ can be interpreted as the statement that Γ has enough resources to solve A or B individually, but there is no indication as to which one is solvable.

The separation between many-one and Turing reductions in Chapter 7 shows there are problems Q_1 and Q_2 such that Q_1 can be solved with two queries to Q_2 but cannot be solved with only one call to Q_2. This result is shown by applying $\otimes$ and $\&$ to Q_1, Q_2 such that $Q_1 \not\leq_{\mathrm{T}} Q_2$ and $Q_2 \not\leq_{\mathrm{T}} Q_1$; for example, Q_1 and Q_2 can be taken as LONELY and PIGEON, see Figure 1.1. Then $Q_1 \otimes Q_2$ is certainly Turing reducible to $Q_1 \& Q_2$, since a reduction needs only to choose to solve Q_1 out of $Q_1 \& Q_2$ and then choose to solve Q_2 out of $Q_1 \& Q_2$. However, if $M : Q_1 \otimes Q_2 \leq_{\mathrm{m}} Q_1 \& Q_2$, then M's one query to $Q_1 \& Q_2$ becomes either a query to Q_1 or a query to Q_2. Then this one query must solve both Q_1 and Q_2, which we expect to give a many-one reduction from Q_1 to Q_2, or vice versa, which is a contradiction. This is only an intuitive argument, Section 7.3 supplies the details that make it correct.

Chapter 7 goes on to prove new and stronger separations by introducing a kind of reduction called a k-reduction. A k-reduction from Q_1 to Q_2, written $Q_1 \leq_k Q_2$, is a Turing reduction that makes at most k queries to Q_2; thus 1-reducibility is identical to many-one reducibility. An important result of Chapter 7 is that there are Q_1, Q_2 such that $Q_1 \leq_{k+1} Q_2$ but $Q_1 \not\leq_k Q_2$. The problems

Q_1, Q_2 are formed from the mod d counting principles. The mod d principle states that it is impossible to partition a set of size $N \not\equiv 0 \bmod d$ into sets of size d. Chapter 7 formulates this principle as the TFNP^2 problem MOD^d. The separation of $\leq_{k+1}$ and $\leq_k$ shows that $Q_1 = \text{MOD}^{p_1} \otimes \cdots \otimes \text{MOD}^{p_{k+1}}$ is not k-reducible to $Q_2 = \text{MOD}^{p_1} \& \cdots \& \text{MOD}^{p_{k+1}}$. Since it is obvious that $k+1$ calls to Q_2 suffice to solve Q_1, this result separates k calls from $k+1$ calls.

MOD^d is the first formulation of the general mod d counting principle as a TFNP^2 problem; [21] defines an axiom which is equivalent to MOD^2, but does not treat the general case. Previously, modular counting had been studied in the form of propositional formulas, for example the *Count* principles defined in [3]. The main result of [3] characterizes the provability between different *Count* principles via constant depth, polynomial size proofs; in particular, no such proofs exist between the d_1 and d_2 *Count* formulas when d_1, d_2 are relatively prime. In order to make use of this result in our setting, Section 7.4 shows that *Count* and the propositional translation of MOD^d are equivalent via constant depth, polynomial size propositional proofs. Proving the separation $Q_1 \not\leq_k Q_2$, where Q_1 and Q_2 are defined in the preceding paragraph, is then a three step process: (1) translate a supposed k-reduction into constant depth, polynomial size propositional proofs between MOD^d principles (2) use the equivalence between *Count* and MOD^d to obtain constant depth, polynomial size propositional proofs between *Count* formulas (3) use the provability characterization of [3] for the *Count* formulas to derive a contradiction. Chapter 7 contains the proof which fills in the details of this outline.

The MOD^d principles are interesting in their own right. The many-one closure of each MOD^d principle defines its own subclass of TFNP^2, called PMOD^d (for "polynomial mod d argument"), in much the same way that ITER, for example, defines PLS. In fact, MOD^2 is exactly the problem LONELY, so that PMOD^2 is exactly PPA. This is not surprising since LONELY is based on the fact that a graph with degree at most 2 has an even number of odd degree nodes, which is a kind of parity argument. Chapter 8 explores the relative complexity of the classes PMOD^d among themselves and with five other classes: PPAD, PPADS, PPA, PPP, and PLS. The equivalence of MOD^2 and LONELY motivates the search

for separations and reductions involving MOD^d. And in fact, many of the results involving LONELY do generalize to MOD^d, though there are still some relations that are left open.

The relative complexity of PMOD^d discussed in Chapter 8 depends on whether d is prime or not. This is because some of the separations, $\text{ITER} \not\leq_{\text{T}} \text{MOD}^p$ when p is prime for example, are proved using lower and upper bounds on the degrees of Nullstellensatz refutations. The upper bounds are usually proved with respect to a base field whose characteristic simplifies the polynomials involved. For example, Section 8.2 shows that MOD^p has degree p Nullstellensatz refutations over fields with characteristic p. It is an open question whether similar refutations exist for composite d, and so our current proof method gives no information as to whether ITER is Turing reducible to MOD^d when d is composite. Despite this, we still conjecture that $\text{ITER} \not\leq_{\text{T}} \text{MOD}^d$ for composite d. The results of Chapter 8 that do not require Nullstellensatz refutations hold for all d, prime or composite.

There are several possible directions for future research. One such direction is to consider corresponding weak versions of the main TFNP^2 classes. For instance, PIGEON is the TFNP^2 problem characterized by the fact that there is no injection from $n+1$ pigeons to n holes. The corresponding weak problem is WeakPIGEON characterized by the fact that there is no injection from n^2 pigeons to n holes. It is shown in [9] through the translation to propositional proofs that $\text{PIGEON} \not\leq_{\text{m}} \text{WeakPIGEON}$ (the translation of Turing reductions into propositional proofs of Chapter 3 extends this separation to Turing reductions). The problems WeakOntoPIGEON and WeakLeftPIGEON are the weak versions of OntoPIGEON and LeftPIGEON, respectively, and have the property that they can be solved by polynomial time random algorithms. Since this property does not hold for the corresponding strong versions, $\text{OntoPIGEON} \not\leq_{\text{T}} \text{WeakOntoPIGEON}$ and $\text{LeftPIGEON} \not\leq_{\text{T}} \text{WeakLeftPIGEON}$. (This type of argument can be used to give an alternate proof that $\text{PIGEON} \not\leq_{\text{T}} \text{WeakPIGEON}$.) Relatively little is known of the relations between weak TFNP^2 classes. For instance, it is clear that $\text{WeakOntoPIGEON} \leq_{\text{T}} \text{WeakLeftPIGEON} \leq_{\text{T}} \text{WeakPIGEON}$, but it is unknown whether any of the reverse directions hold. Also, it is unknown

whether the weak classes are closed under Turing reductions; for example, does $Q \leq_{\mathrm{T}}$ WeakOntoPIGEON imply that $Q \leq_{\mathrm{m}}$ WeakOntoPIGEON?

Another direction for future work is generalizing TFNP^2 classes. Given a type-2 TFNP^2 problem Q, any relation R that is characterized by universal conditions can be added to Q as an oracle by adding as solutions all $\vec{x}$ such that $R(\vec{x})$ fails. For example, recall that a (type-2) complete problem for PLS is ITER, for which a solution is a local minimum in a graph arising from an integer cost function with respect to a local search heuristic. Implicit in ITER is the standard ordering $<$ on the integers, which is characterized with purely universal conditions. If this ordering is instead described by an oracle which answers queries "$a < b$?", a more general problem called GITER (for "generalized iteration principle") is created. The problem GITER is equivalent to the problem HOP (for "Hebrandized ordering principle") defined in [37], where it is shown that HOP is not reducible to the weak pigeonhole principle. In fact, the separation is shown in the case when $<$ is the successor function. Let $\mathrm{HOP}^{\mathrm{succ}}$ be the problem HOP with added oracle access to a successor function, so that $\mathrm{HOP}^{\mathrm{succ}}$ is not reducible to the weak pigeonhole principle; define $\mathrm{GITER}^{\mathrm{succ}}$ similarly. It is easy to show $\mathrm{HOP}^{\mathrm{succ}} \leq_{\mathrm{m}} \mathrm{GITER}^{\mathrm{succ}} \leq_{\mathrm{m}}$ GITER, but it is unknown whether there are reductions in the reverse directions.

Chapter 2

Preliminaries

2.1 Overview

This chapter contains the definitions that are the basis for all other chapters. Section 2.2 contains the definitions of TFNP (from [28]) and TFNP^2 (from [2]). The difference between the two classes is that a TFNP problem only takes strings (type-0 objects) as input, whereas a TFNP^2 problem also takes functions (type-1 objects) as inputs which are accessed as oracles. The class TFNP^2 is a relativized version of TFNP, and, as discussed in the introduction, it is therefore possible to prove separations in TFNP^2, whereas doing the same for TFNP implies $\text{P} \neq \text{NP}$.

Section 2.3 defines many-one and Turing reductions (1) from a TFNP problem to a TFNP^2 problem and (2) from a TFNP^2 problem to a TFNP^2 problem. Case (1) is important to define important non-relativized TFNP subclasses in terms of one canonical TFNP^2 problem. Case (2) corresponds to TFNP relativized to a generic oracle [2]. All separations are proved with respect to the reductions of case (2). Section 2.4 shows how TFNP^2 can arise from existential first-order sentences, and follows [9, 30] closely. In particular, the classes PPA, PPAD, PPADS, PPP, and PLS are defined in this manner in Section 2.5. The first-order formulation of TFNP^2 problems is an important concept used throughout the remaining chapters.

Section 2.6 defines the propositional LK and Nullstellensatz proof systems, which are the two proof systems needed for the later chapters. Propositional LK is a sequent calculus for propositional formulas. The complexity of a propositional

LK proof can be measured by the number of symbols in the proof, the depth of the formulas in the proof, and the height of a tree representation of the proof. Propositional LK allows cuts, and thus it is also possible to define cut-free or free cut-free proofs. The Nullstellensatz calculus is a refutation system based on polynomials. The complexity measure of a Nullstellensatz proof is the maximum degree of the polynomials involved.

2.2 TFNP and TFNP^2

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. The set of all total NP search problems is denoted TFNP.

Definition 2.2.1. *Let x, y be binary strings, and let $R(x, y)$ be a polynomial-time predicate such that $R(x, y)$ implies $|y| \leq p(|x|)$, for some polynomial p. The search problem Q_R is: "Given x, find y such that $R(x, y)$." This is called an* NP search problem. *Let $Q_R(x)$ be the set $\{y|R(x, y)\}$. The 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); only binary strings are considered, so "string" always means "binary string". Let U_n denote the set $\{0, 1\}^n$ of strings of length n. As defined above, TFNP problems take only a string as input; these problems are called "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.

All type-1 inputs f considered will have arguments and outputs that are strings of (the same) 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 is constrained to only query its

functions with values from U_n. The 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 any call to the function oracles counts as a single time step.

Definition 2.2.2. *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.

Requiring $\ell \geq 1$ guarantees there is at least one string input, so that M always knows the size parameter n. If there is no need for a particular string input, the input 0^n can be used as a placeholder. Note that if there are multiple string inputs they must all be the same length n.

This 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. However, the linear growth rate convention better fits the formalizations into first-order formulas of Section 2.4. 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.

2.3 Many-one and Turing Reductions

This section defines many-one and Turing reductions to a TFNP^2 problem. It is possible to reduce either a TFNP or TFNP^2 problem to a TFNP^2 problem. The first case is used to define the common TFNP classes, such as PPA, PPAD, PPADS, PPP, and PLS. The second case is used to separate TFNP classes with

respect to a generic oracle. The definitions of many-one and Turing reductions to a type-2 NP search problems are due to [2].

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; Definition 2.3.1 considers the case where M is type-1.

Definition 2.3.1. *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. The machine M presents a query $(g_1, \ldots, g_r, y_1, \ldots, y_\ell)$ to Q by writing each string y_i on its designated query tape 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. The machine M runs in* polynomial time relative to Q *provided M always halts within $p(n)$ steps for some fixed polynomial p, where n is the length of the string input to M.*

If M is type-2 instead of type-1, 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 input 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.

Note that we could have equivalently had M write out a circuit descriptions of the functions g_i's. Special oracle gates are used in the case where the g_i's have oracle access to the f_i's.

We use type-2 oracle machines to define the notions of many-one and Turing reducibility.

Definition 2.3.2. *Let $Q_1(\vec{f},\vec{x})$ and $Q_2(\vec{g},\vec{y})$ be type-2* NP *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, denoted $Q_1 \leq_{\mathrm{T}} Q_2$ or $M : Q_1 \leq_{\mathrm{T}} Q_2$. 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, denoted $Q_1 \leq_{\mathrm{m}} Q_2$ or $M : Q_1 \leq_{\mathrm{m}} Q_2$. 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 type-1 instead of type-2. The standard classes of (type-1) TFNP problems are defined in terms of reductions to type-2 problems.

Definition 2.3.3. *Let $Q \in \mathrm{TFNP}^2$ search problem. The class $\mathrm{C}_{\mathrm{m}}(Q)$, respectively $\mathrm{C}_{\mathrm{T}}(Q)$, is the set of* TFNP *problems which are many-one, respectively Turing, reducible to Q.*

If $Q \in \mathrm{TFNP}^2$, then Q is a canonical complete problem for $\mathrm{C}_{\mathrm{m}}(Q)$. Section 2.5 defines the canonical complete problems for the TFNP classes PPA, PPAD, PPADS, PPP, and PLS.

2.4 TFNP^2 Problems and First-order Formulas

This section introduces Buresh-Oppenheim and Morioka's method [9, 30] of defining TFNP^2 search problems and TFNP classes in terms of first-order formulas. As a example, consider the following formula from [16, 13]:

$$0 < f(0) \wedge \forall x[x \leq f(x)] \rightarrow \exists x[x < f(x) \wedge f(f(x)) = f(x)]. \tag{2.1}$$

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 (2.1) is valid in all such structures.

This formula expresses the totality of the TFNP^2 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: (1) $f(0^n) = 0^n$, or (2) $f(u) < u$, or (3) $u < f(u)$ and $f(f(u)) = f(u)$." As is well known, ITER is many-one complete for PLS, and thus $\mathrm{PLS} = \mathrm{C_m}(\mathrm{ITER})$. (In fact, also $\mathrm{PLS} = \mathrm{C_T}(\mathrm{ITER})$ as is shown in Chapter 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.

It is possible to assume without loss of generality that there are no relation symbols and no uninterpreted constant symbols. Uninterpreted constant symbols c can be replaced a new unary function symbol f_c, and 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, empty structures are not allowed, 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. (Interpreted constant symbols could also be eliminated, but this yields no advantage, so it is not done.)

Following [9], interpreted constant and function symbols are called "built-in". The interpretation of built-in symbols depends only on n. Commonly used built-in symbols include 0, 1, $f_{\leq}$, and $f_{<}$, and 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 $f_{<}$.

A "basic" language is one that follows the above simplifications.

Definition 2.4.1. *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 2.4.2. *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.

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

Definition 2.4.3. *Suppose* Φ *is an* $\exists$*-sentence of the form* $\exists\vec{x}\phi(\vec{x})$ *over a basic language with non-built-in functions* $f_1, \ldots, f_r$. *The* TFNP^2 *problem* Q_Φ *is: "Given a string* 0^n *and interpretations for the* f_i*'s in the structure* U_n*, find* $\vec{u} \in U_n$ *such that* $\phi(\vec{u})$ *holds." If* Φ *is total, then* Q_Φ *is called a* first-order TFNP^2 problem.

If Φ is total, it is clear that Q_Φ is in TFNP^2. The string input 0^n to Q_Φ serves only as a size parameter. If a formula $g(\vec{u}) = v$ occurs in an $\exists$-sentence Φ, the function g is either built-in or not. If g is not built-in, it is one of the input functions to Q_Φ. 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 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 f_i's.

The following definition simplifies the technical details of working with first-order TFNP^2 problems in Chapters 3 and 4.

Definition 2.4.4. *Let* Q_Φ *be a first-order* TFNP^2 *problem, where* Φ *is a formula over a basic language with one non-built-in function. Then* Q_Φ *is a* basic *first-order* TFNP^2 *problem if* Φ *is of the form* $\exists\vec{x}\phi(\vec{x})$*, where* $\phi(\vec{x})$ *is in disjunctive normal form (DNF)* $\bigvee_{j=1}^{J}\phi_j(\vec{x})$*, such that, for each* $1 \leq j \leq J$*,* $\phi_j(\vec{x})$ *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.*

By a slight abuse of notation, if Φ is $\exists\vec{x}\phi(\vec{x})$ with ϕ in DNF $\bigvee_{j=1}^{J}\phi_j(\vec{x})$, then $\phi_j(\vec{x})$ is called the j^{th} disjunct of Φ.

It is easy to see that every first-order TFNP^2 problem Q_Φ is equivalent to a basic first-order TFNP^2 problem $Q_{\Phi'}$. If Φ is $\exists\vec{x}\phi(\vec{u})$, then putting ϕ in DNF just increases the size of Φ, and thus Q_Φ is still in TFNP^2. Complex terms in Φ 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 $\exists x[g(\vec{u}) = x \wedge x \neq v]$. The absence of formulas of the form $g(\vec{u}) \neq v$ simplifies the arguments in Chapters 3 and 4. This absence also has some implications for the reversal in Chapter 5 and the degree p Nullstellensatz refutations of Chapter 8; see the discussion after Theorem 5.2.1 and the discussion before Theorem 8.2.4, respectively, for more details.

If there are multiple non-built-in function symbols $f_1, \ldots, f_r$ 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. (The case when $i > 2^n$ is irrelevant, since the results rely on asymptotics). Let f have arity equal to one plus the maximum arity of the f_i's. Then each occurrence of $f_i(\vec{x})$ in Φ can be replaced with $f(\underline{i}, \vec{x}, \vec{0})$, where $\vec{0}$ represents extra inputs that pad out to the arity of f. If Φ' is the resulting formula, then clearly $Q_{\Phi'}$ is a basic first-order TFNP^2 problem and $Q_\Phi \equiv_{\mathrm{m}} Q_{\Phi'}$.

2.5 The Classes Defined

Section 2.4 already defined ITER and PLS, however it is restated here for convenience. The rest of the section defines the four classes that are shown in Figure 1.1.

PLS**.** Let Φ be the prenex form of the formula

$$0 < f(0) \wedge \forall x[x \leq f(x)] \rightarrow \exists x[x < f(x) \wedge f(f(x)) = f(x)].$$

Define ITER to be Q_Φ. Then ITER expresses the iteration principle discussed above which characterizes PLS. In fact, define PLS to be $\mathrm{C_m}$(ITER).

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

$$g(0) = 0 \wedge f(0) \neq 0 \rightarrow \exists x[x \neq g(f(x))]. \tag{2.2}$$

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

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

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 prior definitions by [2, 9, 32]. It is more common to define them in terms of the search problems SOURCE.or.SINK, SINK, and LEAF, respectively. It is easy, however, to see that the above definitions are equivalent.

Theses five classes are defined as first-order TFNP^2 problems. As noted in Section 2.4, any first-order TFNP^2 problem is many-one equivalent to a basic one. The simplifications applied to the formula (2.2) defining LeftPIGEON yields

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

where f is now a binary function. The occurrences of $f(t)$ and $g(t)$ in (2.2) 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. Note that the instance of $f(0) \neq 0$ has

been replaced by an equivalent expression, and that the quantifier-free subformula is in disjunctive normal form.

Since any first-order TFNP^2 problem is equivalent to a basic one, the five first-order TFNP^2 problems above will be assumed to be basic or not depending on context.

2.6 Propositional LK and Nullstellensatz Proof Systems

Chapters 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. The conventions for both systems are standard. This section assumes some familiarity with these proof systems, but quickly reviews some terminology.

The formulation we use for propositional proofs is a propositional version of the sequent calculus called propositional LK. Since the first-order version of LK is never used, the modifier "propositional" will frequently be omitted from "propositional LK." The formulas in LK proofs are formed from propositional variables, $\top$, $\bot$ and the connectives $\wedge$, $\vee$, $\neg$. Negations are allowed only on propositional variables, and $\neg x$ is usually denoted $\overline{x}$. An LK proof consists of *sequents* $\Gamma \longrightarrow \Delta$, where Γ, Δ are finite multisets of propositional formulas. The set Γ is called the *antecedent* and the set Δ is called the *succedent*; collectively, Γ and Δ are called *cedents*. The multisets Γ, Δ are usually considered to be unordered, though occasionally we will put an ordering on them. The intended meaning of the sequent $\Gamma \longrightarrow \Delta$ is that the conjunction of the formulas in Γ implies the disjunction of the formulas in Δ.

There are many equivalent formulations of propositional LK, but we assume the rules of inference are given by Figure 2.6. The sequents above the line in an inference are called the *hypotheses* of the inference and the sequent below the line is called the *conclusion* of the sequent. The absence of the exchange rules

$$\frac{\Gamma, B, A, \Pi \longrightarrow \Delta}{\Gamma, A, B, \Pi \longrightarrow \Delta} \qquad \frac{\Gamma \longrightarrow \Sigma, B, A, \Delta}{\Gamma \longrightarrow \Sigma, A, B, \Delta}$$

corresponds to the fact that the sets of formulas are unordered multisets. An LK proof is a tree of sequents formed by applying the rules of inference such that the leaves are either logical axioms or $\top$ or $\bot$ inferences. The conclusions of these three inferences are called *logical initial* sequents. Occasionally, the leaves of an LK proof will be allowed to be non-logical initial sequents. A family of LK proofs can be measured by their size, their depth, their height and their cut complexity; see [6] or [26] 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.

The separations in Chapters 3 and 4 only rely on 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 $N = 2^n$, the size of $\mathcal{P}_n$ is 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 bounded quasipolynomially in N. The complexity of a proof is measured 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, refer to [3] or [12]. 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

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

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

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

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

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

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

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

Figure 2.1: Γ' and Δ' are sets of formulas such that $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$.

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 identically 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 $p_i q_i$. The Nullstellensatz refutations of Chapter 4 will have degree $(\log N)^{O(1)}$, that is, polylogarithmic degree.

This chapter contains material from the paper "Propositional proofs and reductions between NP search problems" which is currently accepted for publication by the Annals of Pure and Applied Logic. This paper is co-authored by the dissertation author and Samuel R. Buss.

Chapter 3

Forward Direction, Propositional LK

3.1 Overview

The main result of this chapter is Theorem 3.3.1, which states that if $Q_1, Q_2 \in \text{TFNP}^2$ and $M : Q_1 \leq_{\text{T}} Q_2$, then there are constant depth, polynomial size LK proofs of the totality of Q_1 from the totality of Q_2. In particular, M can make polynomially many calls to Q_2. As mentioned in the introduction, this result was proved in [9] for the case when M can only make one call to Q_2, that is when $M : Q_1 \leq_{\text{m}} Q_2$. Using existing upper and lower bounds from proof complexity and the translation from many-one reductions gave new proofs of PIGEON $\not\leq_{\text{m}}$ ITER, LONELY $\not\leq_{\text{m}}$ ITER, and LONELY $\not\leq_{\text{m}}$ PIGEON. Since Theorem 3.3.1 extends the previous translation to Turing reductions, there are new proofs these separations with respect to Turing reductions.

In addition to extending the translation to Turing reductions, Theorem 3.3.1 has other advantages over the results of [9]. The first is that [9] assumes that Q_2 has the instance extension property. This property states that an instance of Q_2 on size n can be extended to an instance of size $n^{O(1)}$ such that solutions to the original, smaller instance of Q_1 can be extracted from solutions to the larger instance in polynomial time. A minor improvement is that Theorem 3.3.1 does not assume

the instance extension property. A more significant fact is that Theorem 3.3.1 gives a more careful analysis of the height and cut complexity of the LK proofs. This is important in formulating the reverse problem of using propositional LK proofs to construct a Turing reduction $M : Q_1 \leq_{\mathrm{T}} Q_2$. Another advantage of using Turing reductions over many-one reductions is that the reverse construction produces Turing reductions. The reverse problem is stated and proved in Chapter 5.

Decision trees are important tools in the proof of Theorem 3.3.1. The first kind of decision tree describes the computation of a Turing machine relative to a function oracle. This type of decision tree has been studied extensively, see [2, 3, 9]. A slightly different type of decision tree is needed for the computation of a Turing machine that uses a TFNP^2 problem as an oracle. The difference is that an edge in such a decision tree may indicate that a query to $Q \in \mathrm{TFNP}^2$ is answered by $\vec{u}$. However, the verification that $\vec{u}$ solves the instance of Q runs in polynomial time relative to some function f, and thus the verification process may reveal values for f that contradict previously known values. Such an edge is said to contain an *implicit* contradiction. Traces, defined in Section 3.4, are analogous to paths in decisions trees for function oracles, and are the technical tool needed to avoid implicit contradictions.

Section 3.2 defines the propositional LK formulas that encode a TFNP^2 problem. Section 3.3 states the translation in Theorem 3.3.1 and related results. Using these, the separations of [9] are proved in the context of Turing reductions; the proof of Theorem 3.3.1 is delayed until Section 3.5. Section 3.4 defines the two types of decision trees mentioned above. Decision trees play an important role throughout the remaining chapters. Traces are also defined in Section 3.4, and are the main technical tool needed to prove Theorem 3.3.1, as well as Theorem 4.3.1, which is the main result of Chapter 4.

3.2 Propositional LK Formulation

This section defines the propositional formulas that express the totality of a TFNP^2 search problem defined by an $\exists$-sentence. Let Q_Φ be a basic first-order

TFNP^2 problem, where the language has one uninterpreted, k-ary function symbol f; see Section 2.4 for details. The goal is to define the propositional sequent $F_{\Phi,n} \longrightarrow G_{\Phi,n}$ that expresses the totality of the search problem Q_Φ on inputs of length n.

As an example, consider the case of the problem LONELY defined in Section 2.5. The problem LONELY is Q_Φ where Φ is the prenex form of

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

Note that Q_Φ is a first-order TFNP^2 problem, but it is not a basic one since there are nested function symbols and subformulas of the form $f(a) \neq b$. As noted in Section 2.4, LONELY is equivalent to a basic first-order TFNP^2 problem $Q_{\Phi'}$. Specifically, Φ' is the formula

$$\exists x, y, z[(f(0) = x \wedge x \neq 0) \vee (y = f(x) \wedge z = f(y) \wedge x \neq z) \vee (x \neq 0 \wedge f(x) = x)].$$

The translation of an instance of $Q_{\Phi'}$ with size parameter n uses the propositional variables $x_{u,v}$, for all $u, v \in U_n$, which define the graph of f: the intended meaning of $x_{u,v}$ is that $f(u) = v$. This is the reason that a basic first-order TFNP^2 problem disallows terms with nested function symbols. For the $x_{u,v}$'s to properly define a function, there should be exactly one value $v \in U_n$ for each $u \in U_n$ such that $x_{u,v}$ holds. The totality is expressed by the formulas $\{\bigvee_v x_{u,v} : v \in U_n\}$, and the functionality is expressed by the formulas $\{\overline{x}_{u,v} \vee \overline{x}_{u,v'} : u, v, v' \in U_n, v \neq v'\}$. The union of these sets is $F_{\text{LONELY},n}$.

The last part of the translation expresses the totality of the search problem $Q_{\Phi'}$ on an instance of size n. One solution is $x_{0^n,u}$ for any $u \neq 0^n$, which corresponds to the first disjunct of Φ'. A triple $u, v, w \in U_n$ is a solution if $x_{u,v} \wedge x_{v,w} \wedge u \neq w$ holds (corresponding to the second disjunct) or if $u \neq 0^n \wedge x_{u,u}$ holds (corresponding to the third disjunct). Therefore the existence of a solution to LONELY on size n is expressed by the formula $G_{\text{LONELY},n}$ defined as

$$\bigvee_{\substack{u \in U_n \\ u \neq 0^n}} x_{0^n,u} \vee \bigvee_{\substack{u,v,w \in U_n \\ u \neq w}} (x_{u,v} \wedge x_{v,w}) \vee \bigvee_{\substack{u \in U_n \\ u \neq 0^n}} x_{u,u}.$$

(This formula differs in unimportant ways from the one produced by Definition 3.2.2 below. This formulation better captures the intuition; see the discussion at the end

of this section.) The search problem LONELY on size n is then expressed as the sequent $F_{\text{LONELY},n} \longrightarrow G_{\text{LONELY},n}$.

The following definitions generalize this translation to any basic first-order TFNP^2 problem.

Definition 3.2.1. *Let Φ be an $\exists$-sentence over a basic language with one uninterpreted, k-ary function symbol. Throughout the definition, $\vec{u}$ is a vector of k elements of U_n. Then $Tot_{\Phi,n}$ is the set of formulas*

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

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

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

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

The notation $\bigvee$ in the definition of $Tot_{\Phi,n}$ is shorthand for $2^n - 1$ many binary disjunctions, and hence formulas in $Tot_{\Phi,n}$ have depth 1. The formulas in $Func_{\Phi,n}$ are each depth 0.5 since they each are a disjunction of two literals. The sets $Tot_{\Phi,n}$ and $Func_{\Phi,n}$ both contain $2^{n^{O(1)}}$ many formulas. Specifically, $Tot_{\Phi,n}$ 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. Therefore, the conjunction $\bigwedge Tot_{\Phi,n}$ is a depth 2 formula and the conjunction $\bigwedge Func_{\Phi,n}$ is a depth 1.5 formula.

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

Definition 3.2.2. *Suppose Q_Φ is a basic first-order* TFNP^2 *problem. That is, the language contains one uninterpreted, k-ary function symbol f and Φ is in disjunctive normal form $\exists \vec{x} \bigvee_{j=1}^{J} \phi_j(\vec{x})$. Furthermore, each ϕ_j is a conjunction of literals $\ell_{j,1}, \ldots, \ell_{j,i_j}$ such that each literal is of the form $g(\vec{a}) = b$ or $b = c$ or $b \neq c$, 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. Let $\vec{x}$ be*

a vector of s variables $x_1, \ldots, x_s$; fix $n \geq 1$; 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, $\mathrm{val}_{\vec{u}}(x_i) = u_i$. This notation is extended to built-in constant symbols c by letting $\mathrm{val}_{\vec{u}}(c)$ equal the interpretation of c in U_n. 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 is defined to be the conjunction of the translations of the literals 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 set $Soln_{\Phi,n}$ is defined to be the set containing the 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 $Soln_{\Phi,n}$.

We abuse notation slightly when Q_Φ is a named problem such as ITER, PIGEON, etc. For example, let Φ be the $\exists$-sentence such that Q_Φ is ITER. To avoid having to write out this specific Φ, we simply write $F_{\mathrm{ITER},n}$ instead of $F_{\Phi,n}$. A similar convention holds for other the other named problems.

Each $[\![\phi_j]\!]_{\vec{u}}$ is $Soln_{\Phi,n}$ is a conjunction of constantly many literals, so that each formula in $Soln_{\Phi,n}$ is depth 0.5. Since $Soln_{\Phi,n}$ contains 2^{kJn} formulas, $G_{\Phi,n}$ is a depth 1.5 formula. Overall, the maximum depth of the formulas in $F_{\Phi,n} \longrightarrow G_{\Phi,n}$ is 2, due to the presence of $\bigwedge Tot_{\Phi,n}$. Also note that, since Q_Φ is a basic first-order TFNP^2 problem, the formulas in $Soln_{\Phi,n}$ contain no negated variables.

As an example of the construction in part (b) in Definition 3.2.2, again consider LONELY. Fix $u, v, w \in U_n$ such that $u \neq w$. Then $[\![y = f(x) \wedge z = f(y) \wedge x \neq z]\!]_{u,v,w}$ is $x_{u,v} \wedge x_{v,w} \wedge \top$; note how $x \neq z$ is

translated to $\top$ since the fixed values u and w are not equal. Similarly, $[\![y = f(x) \wedge z = f(y) \wedge x \neq z]\!]_{u,v,u}$ is $x_{u,v} \wedge x_{v,u} \wedge \bot$, because $[\![x \neq z]\!]_{u,v,u}$ is now $\bot$ since it is false that u does not equal u.

To see how the construction deals with built-in symbols, consider the disjunct $x \neq 0 \wedge f(x) = x$ from LONELY; recall the interpretation of 0 in U_n is 0^n. Then for any $u \in U_n$ which is not 0^n, $[\![x \neq 0 \wedge f(x) = x]\!]_u$ is $\top \wedge x_{u,u}$ and $[\![x \neq 0 \wedge f(x) = x]\!]_{0^n}$ is $\bot \wedge x_{0^n,0^n}$.

In general, $Soln_{\Phi,n}$ contains redundant and unimportant information. Take the example of LONELY. Even though the disjunct $x \neq 0 \wedge f(x) = x$ does not contain y or z the set $Soln_{\text{LONELY},n}$ contains $[\![x \neq 0 \wedge f(x) = x]\!]_{u,v,w}$ for all choices of $u, v, w \in U_n$. Thus for each $u \neq 0^n$ there are 2^{2n} copies of the formula $\top \wedge x_{u,u}$. Note that the $\top$ in this formula can be removed to obtain an equivalent formula, but this is not done. The same holds for $\bot$, in particular $[\![x \neq 0 \wedge f(x) = x]\!]_{0^n,v,w}$ is $\bot \wedge x_{0^n,0^n}$. These simplifications are not used because they would not reduce the number of formulas in $Soln_{\text{LONELY},n}$ below $2^{n^{O(1)}}$, and the formulas in $Soln_{\text{LONELY},n}$ would still be constant size. In the example above, the formula $G_{\text{LONELY},n}$ is worked out using these simplifications, even though they are not present in the definition, because at the time these details would have distracted from the motivation for the example.

3.3 Main Theorem and Separations

This section states Theorem 3.3.1, which is the main result of this chapter; the proof appears in Section 3.5. Theorem 3.3.1 states that if $Q_\Phi \leq_{\text{T}} Q_\Psi$, then there are "well-behaved" propositional 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 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 (3.1) will be "decision tree substitutions" as defined in Chapter 5.) Recall that we only allow negations on variables, thus

when substituting for $\overline{x}_{\vec{u},v}$ De Morgan's laws are used to push the negation in $\overline{\lambda_{\vec{u},v}}$ down to the literals.

Theorem 3.3.1. *Let Q_Φ and Q_Ψ be basic first-order* TFNP^2 *problems, and suppose $Q_\Phi \leq_{\mathrm{T}} Q_\Psi$. Then there are proofs $\mathcal{P}_n$ of the sequents*

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

such that:

(a) *The initial sequents of $\mathcal{P}_n$ are either logical axioms or are depth 1.5 substitution instances of sequents $F_{\Psi,m} \longrightarrow G_{\Psi,m}$;*

(b) *$\mathcal{P}_n$ has size $2^{n^{O(1)}}$, height $n^{O(1)}$, and constant depth; and*

(c) *There are no free cuts in $\mathcal{P}_n$.*

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 3.3.2. *Let Q_Φ and Q_Ψ be basic first-order* TFNP^2 *problems, and suppose that $Q_\Phi \leq_{\mathrm{T}} Q_\Psi$. If 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 quasipolynomial size, constant depth proofs.*

Corollary 3.3.2 follows immediately from Theorem 3.3.1, since the proofs from Theorem 3.3.1 can be combined with substitution instances of proofs of $F_{\Psi,m} \longrightarrow G_{\Psi,m}$ to obtain proofs of $F_{\Phi,n} \longrightarrow G_{\Phi,n}$.

Existing upper and lower bounds on the complexity of proofs of $F_{\Phi,n} \longrightarrow G_{\Phi,n}$ in conjunction with Corollary 3.3.2 give Turing separations between various TFNP^2 problems. The separations (a)-(c) of Corollary 3.3.3 were already proved in [2, 29]. However, Corollary 3.3.3 gives new proofs of these facts as well as emphasizes the

connection to proof complexity theory. Previously, [9] used proof complexity results to recreate these separations only with respect to many-one reductions, not Turing reducibility.

Corollary 3.3.3. (a) PIGEON $\not\leq_{\mathrm{T}}$ ITER. [29]
(b) LONELY $\not\leq_{\mathrm{T}}$ ITER. [29]
(c) LONELY $\not\leq_{\mathrm{T}}$ PIGEON. [2]

Proof. [4] shows that $F_{\Phi,n} \longrightarrow G_{\Phi,n}$ require exponential size bounded depth proofs when Q_Φ is LONELY or PIGEON. By [9], $F_{\mathrm{ITER},n} \longrightarrow G_{\mathrm{ITER},n}$ have quasipolynomial size bounded depth proofs. The separations (a) and (b) follow by invoking Corollary 3.3.2. It is shown in [5] 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 3.3.1. □

The proof of Theorem 3.3.1 will use the possible executions of a Turing reduction M to construct the proofs $\mathcal{P}_n$. Section 3.4 constructs, for each $n \geq 1$, 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.

The next theorem implies Theorem 3.3.1, and better suits the statement of the reversal in Chapter 5. Substitutions are denoted with postfix notation, so $A\sigma$ indicates the result of applying the substitution σ to the propositional formula A.

Theorem 3.3.4. *Let Q_Φ and Q_Ψ be basic first-order* TFNP^2 *problems, and suppose $Q_\Phi \leq_{\mathrm{T}} Q_\Psi$. 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 a constant-depth, 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) $\bigwedge_{i=1}^{r} (\bigvee Soln_{\Psi,m_i})\, \sigma_i, F_{\Phi,n} \longrightarrow G_{\Phi,n}$.

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

To see that Theorem 3.3.1 follows immediately from Theorem 3.3.4, 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

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

for $i = 1, \ldots, r$. 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

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

Finally, cut this against the sequent (c).

The rest of this chapter gives the proof of Theorem 3.3.4. Section 3.4 defines decision trees which are needed to define the substitutions. Lemma 3.5.1 proves parts (a) and (b) and Lemma 3.5.2 proves part (c).

3.4 Decision Trees

Suppose that M is a Turing reduction from Q_Φ to Q_Ψ. The machine M takes as input $(\alpha, 0^n)$ where $\alpha : U_n^k \to U_n$. (The type-0 input is just 0^n since 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 $\vec{w} \in U_m$, $M'(\vec{w})$ computes the value of $\beta(\vec{w})$. The machine M' is allowed to make oracle calls to α but otherwise runs deterministically. Accordingly, for particular $\vec{w} \in U_m$, the computation of $M'(\vec{w})$ can be described by a (α, n)-decision tree $T_{M',\vec{w}}$ such that the internal nodes of $T_{M',\vec{w}}$ represent queries to α and each leaf of $T_{M',\vec{w}}$ is labeled with an output value:

Definition 3.4.1. *An* (α, n)-decision tree *is a tree where each internal node is labeled* $\alpha(\vec{u})$ *for some* $\vec{u} \in U_n$*. Each internal node has* 2^n *children, with the edges to its children labeled with values* v*, one edge for each* $v \in U_n$*. Leaves may be labeled by any member* z *of* U_m*.*

In $T_{M',\vec{w}}$, an internal node labeled $\alpha(\vec{u})$ corresponds to a query to $\alpha(\vec{u})$, an outgoing edge labeled v corresponds to an oracle response that $\alpha(\vec{u}) = v$, and a leaf labeled with z indicates that $M'(\vec{w})$ outputs $z = \beta(\vec{w})$. 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 $\leq r$ which faithfully represents the computation of $M'(\vec{v})$. (This use of decision trees to represent oracle computations is very similar to many prior works, including [2, 3, 9].)

To simplify some aspects of the proofs in Section 3.5, we adopt the convention that all nodes have 2^n children, even if a particular value $\alpha(\vec{u})$ 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 leaf with label 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 in T. 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 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 3.4.2. *A branch* P *in an* (α, n)*-decision tree is identified with the conjunction*

$$\bigwedge\nolimits_{i=1}^{I} x_{\vec{u}_i, v_i},$$

where $\{\alpha(\vec{u}_i) = v_i\}_{i=1}^{I}$ *is the set of* α *values set by the edge labels of* P*. The formula* $\overline{P}$ *is defined to be* $\bigvee_{i=1}^{I} \overline{x}_{\vec{u}_i, v_i}$*: this expresses the negation of* P*.*

When using P *in a cedent, 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 denotes the ordered cedent* $x_{\vec{u}_1, v_1}, \ldots, x_{\vec{u}_I, v_I}$*.*

Recall that cedents in an LK proof are usually considered as *unordered* multisets. However, $\widehat{P}$ is ordered because we will need to pick out the last edge in the branch.

The proofs that will be constructed for Theorem 3.3.4 will contain substitution instances of $F_{\Psi,m} \longrightarrow G_{\Psi,m}$. These substitution instances are defined from families of (α, n)-decision trees. Namely, suppose $\mathcal{T}$ 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

$$\lambda_{\mathcal{T},\vec{w},y} \;=\; \bigvee_{P \in \mathrm{br}_y(T_{\vec{w}})} P. \tag{3.1}$$

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

The machine M which computes a Turing reduction from Q_Φ to Q_Ψ itself has a decision tree. This decision tree is more complicated than the (α, n)-decision trees used for M' above, since M makes queries to instances of Q_Ψ as well as to α. For $n \geq 1$, 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 (1) with label $\alpha(\vec{u})$ for some $\vec{u} \in U_n$, or (2) with label $Q_\Psi(\beta, 0^m)$ with β described as a polynomial time oracle Turing machine M'. A node with label $\alpha(\vec{u})$ has 2^n children and outgoing edges labeled with values v, one outgoing edge per $v \in U_n$. As before, traversing the outgoing edge labeled v indicates that $\alpha(\vec{u}) = v$. The nodes labeled with $Q_\Psi(\beta, 0^m)$ have $J' \cdot 2^{s'm}$ children, where s' is the output arity of Q_Ψ and Ψ has J' disjuncts; the outgoing edges are labeled with all possible values $(j, \vec{a})$ with $1 \leq j \leq J'$ and $\vec{a} \in U_m$. The intuition is that the edge with label $(j, \vec{a})$ may be traversed if $\vec{a}$ is a solution to the search problem $Q_\Psi(\beta, 0^m)$ and the j^{th} 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 solution to the search problem Q_Φ (with the exception that an arbitrary value may be given for a contradictory path through $T_{M,n}$). Since M has 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 $Q_\Psi(\beta, 0^m)$ 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 $Q_\Psi(\beta, 0^m)$ and the index j of the disjunct of Ψ which is satisfied by $\vec{a}$. On the other hand, when M calls the oracle $Q_\Psi(\beta, 0^m)$, only a value $\vec{a}$ is returned, not the value of j. Nonetheless, 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 also contain "implicit" contradictions if the edge labels $(j, \vec{a})$ on 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 j^{th} disjunct of Ψ with the values $\vec{a}$. The 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, the following definition of a "trace" of a path in $T_{M,n}$ serves exactly this purpose. Traces in $T_{M,n}$ are the analogue to paths starting at the root of an (α, n)-decision tree.

Definition 3.4.3. *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. The set of traces to μ is defined by induction on the length of P_μ. Any trace to μ will consist of an ordered multiset of literals $x_{\vec{u},v}$ intended to indicate the conditions that $\alpha(\vec{u}) = v$. 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 node μ is labeled with a query $\alpha(\vec{u})$, and let μ' be a child of μ with the edge (μ, μ') labeled by $v \in U_n$. If Λ is a trace to μ, then $\Lambda, x_{\vec{u},v}$ is a trace to μ'.*

(c) *Now suppose μ is labeled with a query $Q_\Psi(\beta, 0^m)$, 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 j^{th} disjunct of Ψ, consider $\psi_j(\vec{a})$. If $\psi_j(\vec{a})$ contains a conjunct $g(\vec{w}) = y'$ or $y = y'$ or $y \neq y'$ which is false, then there is no trace to μ. (Here, g is built-in, and the members of $\vec{w}$ and y, y' are interpreted constants or members of $\vec{a}$.) 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'.*

We require that traces are ordered multisets because we will need to pick out the last element of a trace. 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 $\psi_j(\vec{a})$ are true.

Suppose μ is an internal node of $T_{M,n}$ labeled $\alpha(\vec{u})$. Then for each child μ' of μ, the number of traces to μ' is the same as the number of traces to μ. Now suppose μ is labeled $Q_\Psi(\beta, 0^m)$. Then for each child μ' of μ the number of traces to μ' is $2^{n^{O(1)}}$ times the number of traces to μ; the factor $2^{n^{O(1)}}$ corresponds to the number of ways of choosing the paths $P_1, \ldots, P_r$. Since the height of $T_{M,n}$ is $n^{O(1)}$ there are $2^{n^{O(1)}}$ traces to any μ in $T_{M,n}$.

Traces play a key role in the proof of Lemma 3.5.2, which in turn proves part (c) of Theorem 3.3.4. Traces also play a role in the proof of Theorem 4.3.1.

3.5 Proof of Theorem 3.3.1

This proves Theorem 3.3.4, which, as noted above, easily implies Theorem 3.3.1. Theorem 3.3.4 has parts (a), (b), and (c). The proof is broken into two lemmas: Lemma 3.5.1 proves parts (a) and (b) and Lemma 3.5.2 proves (c).

Lemma 3.5.1. *Let Q_Φ and Q_Ψ be basic first-order* TFNP^2 *problems. Let $\mathcal{T} = \{T_{\vec{w}} : \vec{w} \in U_m\}$ be a set of (α, n)-decision trees, and let $\sigma_\mathcal{T}$ the corresponding substitution described above. If $m = n^{O(1)}$, then the following two sequents have constant depth*

LK *proofs which have size* $2^{n^{O(1)}}$*, have height* $n^{O(1)}$*, and are cut free:*

$$\Big(\bigwedge Tot_{\Phi,n}\Big) \longrightarrow \Big(\bigwedge Tot_{\Psi,m}\Big)\,\sigma_{\mathcal{T}}, \tag{3.2}$$

$$\Big(\bigwedge Func_{\Phi,n}\Big) \longrightarrow \Big(\bigwedge Func_{\Psi,m}\Big)\,\sigma_{\mathcal{T}}. \tag{3.3}$$

Proof of Lemma 3.5.1. First consider (3.2). Note that $(\bigwedge Tot_{\Psi,m})\,\sigma_{\mathcal{T}}$ is a balanced conjunction of the formulas $Tot_{\mathcal{T},\vec{w}}$ defined as

$$\bigvee_{y\in U_m}\ \bigvee_{P\in \mathrm{br}_y(T_{\vec{w}})} P,$$

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

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

Combining all these proofs with $\wedge$:right inferences gives the desired LK proof of (3.2). The size bounds and the cut-free property are easy to verify.

Now consider (3.3). The formula $(\bigwedge Func_{\Psi,m})\,\sigma_{\mathcal{T}}$ is a big conjunction of the formulas $Func_{\mathcal{T},\vec{w},y,y'}$ defined as

$$\Big(\bigwedge_{P\in \mathrm{br}_y(T_{\vec{w}})} \overline{P}\Big) \vee \Big(\bigwedge_{P'\in \mathrm{br}_{y'}(T_{\vec{w}})} \overline{P}'\Big), \tag{3.4}$$

with $\vec{w}, y, y' \in U_m$ and $y \neq y'$. Fix $\vec{w}, y, y' \in U_m$. Any pair of branches $P \in \mathrm{br}_y(T_{\vec{w}})$ and $P' \in \mathrm{br}_{y'}(T_{\vec{w}})$, 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 $x_{\vec{u},v}$ on P and $x_{\vec{u},v'}$ on P' with $v \neq v'$. This means $\overline{P}$ and $\overline{P}'$ contain $\overline{x}_{\vec{u},v}$ and $\overline{x}_{\vec{u},v'}$ as disjuncts, respectively. Thus there are simple cut free proofs of the sequents $\bigwedge \mathit{Func}_{\Phi,n} \longrightarrow \overline{P}, \overline{P}'$, one for each such choice of P, P'. Combine these proofs with $\wedge$:right inferences (as usual, in a balanced fashion) to obtain proofs of the sequents $\bigwedge \mathit{Func}_{\Phi,n} \longrightarrow \overline{P}, \bigwedge_{P' \in \mathrm{br}_y(T_{\vec{w}})} \overline{P}'$. Applying more balanced $\wedge$:right inferences to these sequents, and then a single $\vee$:right inference, gives a proof of the sequent $\bigwedge \mathit{Func}_{\Phi,n} \longrightarrow (3.4)$.

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

Lemma 3.5.2. *Let Q_Φ and Q_Ψ be basic first-order* TFNP^2 *problems, and suppose $M : Q_\Phi \leq_{\mathrm{T}} Q_\Psi$. Let μ be a node in $T_{M,n}$, and let Λ be a trace to μ. Let there be r nodes in $T_{M,n}$ which are labeled with calls to Q_Ψ. To fix notation, for $i = 1, \ldots, r$, there are integers m_i and calls to $Q_\Psi(\beta_i, 0^{m_i})$ 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 $\sigma_{\mathcal{T}^i}$.*

Then there is a constant depth, cut-free proof $\mathcal{P}_\mu$ of the sequent

$$\Lambda, \bigwedge_{i=1}^{r} (\bigvee \mathit{Soln}_{\Psi,m_i})\sigma_i, F_{\Phi,n} \longrightarrow G_{\Phi,n}, \tag{3.5}$$

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

Although it is suppressed in the notation, $\mathcal{P}_\mu$ depends on both μ and Λ.

Proof. The trace Λ contains variables $x_{\vec{u},v}$ with $\vec{u}, v \in U_n$. Define Λ to be *contradictory* if it contains two literals $x_{\vec{u},v}$, and $x_{\vec{u},v'}$ where $v \neq v'$. If Λ is contradictory, then there is an easy cut free proof of $\bigwedge \mathit{Func}_{\Phi,n}, \Lambda \longrightarrow$. From this, (3.5) can be inferred by weakening.

For traces which are not contradictory, the existence of the proofs $\mathcal{P}_\mu$ is proved by induction on the depth of μ in the decision tree $T_{M,n}$. To establish the size bounds, let T_μ be the subtree $T_{M,n}$ rooted at μ. The induction will show that

each $\mathcal{P}_\mu$ has height $n^{O(1)}(1 + h_\mu)$ 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 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 Q_Ψ, and it then runs deterministically to output values $\vec{a}$ in U_n that satisfy some disjunction $\phi_j(\vec{a})$. The propositional translation $[\![\phi_j]\!]_{\vec{a}}$ of $\phi_j(\vec{a})$ is a conjunction of variables $x_{\vec{u},v}$ plus possibly occurrences of $\bot$ and $\top$.

Claim: $\bot$ does not occur in $[\![\phi_j]\!]_{\vec{a}}$, and every variable in $[\![\phi_j]\!]_{\vec{a}}$ also appears in Λ.

To prove this claim, note that the trace Λ contains enough information to ensure that there is a correct computation of M that reaches the node μ 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. Recall that ϕ_j is a conjunction of positive literals. Since $\phi_j(\vec{a})$ is true, no conjunct in $\phi_j(\vec{a})$ is false. Therefore, $\bot$ does not appear in $[\![\phi_j]\!]_{\vec{a}}$. Further, every variable $x_{\vec{u},v}$ in $[\![\phi_j]\!]_{\vec{a}}$ must appear in the trace Λ, since, if not, the value of $\alpha(\vec{u})$ could be set to some value $v' \neq v$ and still be consistent with the values specified by Λ. But then $\phi_j(\vec{a})$ would be false, which contradicts the fact that M is a correct Turing reduction. The proves the claim.

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 (3.5). It is clear that this proof has height $n^{O(1)}$.

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

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

Combining these with $\vee$:left inferences eliminates the $x_{\vec{u},v}$'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 $\bigwedge Tot_{\Phi,n}$. Also, since the height of $\mathcal{P}_\mu$ is at most $n^{O(1)}$ greater than the maximum height of any $\mathcal{P}_{\mu_v}$, the height of $\mathcal{P}_\mu$ is

$n^{O(1)}(1+h_\mu)$.

Finally suppose that μ is an internal node labeled $Q_\Psi(\beta, 0^m)$ with the function β described by the Turing machine M' and induced family $\mathcal{T} = \{T_{\vec{w}} : \vec{w} \in U_m\}$ of decision trees. Let σ be $\sigma_{\mathcal{T}}$. 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 path $P_{\mu',i}$ is in $\mathrm{br}_{y_i}(T_{\vec{w}_i})$ if $\beta(\vec{w}_i) = y_i$ is the i^{th} conjunct of $\psi_j(\vec{a})$ that involves β.

The induction hypothesis, and weakening, gives proofs of the sequents

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

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

$$(\bigvee \mathit{Soln}_{\Psi,m})\sigma \longrightarrow \tag{3.6}$$

can be proved from the set of sequents

$$P_{\mu',1}, \ldots, P_{\mu',k_{\mu'}} \longrightarrow \tag{3.7}$$

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

For the moment, 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 $P_{\mu',1}, \ldots, P_{\mu',k_{\mu'}-1}$, use balanced $\vee$:left inferences to combine sequents (3.7) 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 $(\llbracket \psi_j \rrbracket_{\vec{a}})\sigma$. Thus, applying a constant number of $\wedge$:inferences (and possibly a weakening to introduce $\top$), gives a proof of the sequent $(\llbracket \psi_j \rrbracket_{\vec{a}})\sigma \longrightarrow$.

(A different construction is needed in case $\bot$ is in $\llbracket \psi_j \rrbracket_{\vec{a}}$. In this case, there are no traces to μ'. However, it is easy to derive $(\llbracket \psi_j \rrbracket_{\vec{a}})\sigma \longrightarrow$ 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 (3.6) from the sequents (3.7) as desired. It is easy to verify that this cut free proof has height $n^{O(1)}$. $\square$

Part (c) of Theorem 3.3.4 follows immediately from Lemma 3.5.2 by taking μ to be the roof of $T_{M,n}$ since the trace to the root is empty. Since parts (a) and (b) were already established in Lemma 3.5.1, this proves Theorem 3.3.4, and hence Theorem 3.3.1.

This chapter contains material from the paper "Propositional proofs and reductions between NP search problems" which is currently accepted for publication by the Annals of Pure and Applied Logic. This paper is co-authored by the dissertation author and Samuel R. Buss.

Chapter 4

Nullstellensatz Translation

4.1 Overview

The main result of this chapter is Theorem 4.3.1, which states 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 [9] which proved this result with respect to many-one reductions. Corollary 4.3.2 uses this result to prove two new separations: ITER $\not\leq_{\mathrm{T}}$ OntoPIGEON and ITER $\not\leq_{\mathrm{T}}$ LONELY. Corollary 4.3.2 also gives new proofs of previously known separations.

The proof of Theorem 4.3.1 is similar in spirit to the proof of Theorem 3.3.1. In particular, (α, n)-decision trees are used to define substitutions, and traces provide the basis for a proof by induction. However, Theorem 4.3.1 is not analogous to Theorem 3.3.1, but rather to Corollary 3.3.2. Theorem 3.3.1 proves that if $Q_\Phi \leq_{\mathrm{T}} Q_\Psi$, then there are LK proofs of the totality of Q_Φ from the totality of Q_Ψ. In particular, the LK proofs are allowed to use (substitution instances of) the non-logical initial sequents $F_{\Psi,m} \longrightarrow G_{\Psi,m}$ expressing the totality of Q_Ψ. Corollary 3.3.2 differs slightly from this because it requires both that $Q_\Phi \leq_{\mathrm{T}} Q_\Psi$ *and* that there are proofs of Q_Ψ while constructing proofs of Q_Φ with the stronger condition that there are no non-logical initial sequents. Theorem 4.3.1 is similar to Corollary 3.3.2 because it requires both a Turing reduction and a Nullstellensatz proof of the totality of Q_Ψ and produces a Nullstellensatz proof of the totality

of Q_Φ. Additionally, the Nullstellensatz proof of the totality of Q_Φ produced by Theorem 4.3.1 corresponds to the proof constructed by Corollary 3.3.2 (which does not allow for any non-logical initial sequents) and not the proof constructed by Theorem 3.3.1 (which allows for some non-logical initial sequents). Indeed, there does not seem to be a good Nullstellensatz version of a "non-logical initial sequent". This leads us to believe that there is no Nullstellensatz theorem that properly corresponds to Theorem 3.3.1. Section 4.3 contains a detailed discussion of these issues.

Theorem 4.3.1 is stronger than the corresponding result with respect to many-one reductions in [9] because it does not assume the model extension property. However, the result for many-one reductions with the model extension property does have the advantage that it gives rise to Nullstellensatz proofs of the totality of Q_Φ from the totality of Q_Ψ, which is analogous to Theorem 3.3.1. The discussion in Section 4.3 addresses this result in the context of the preceding paragraph.

Section 4.2 sets notation for Nullstellensatz proofs, and defines the sets of polynomials that encode a TFNP^2 problem. Section 4.3 states Theorem 4.3.1 and the separations that it implies. Section 4.4 proves Theorem 4.3.1.

4.2 Nullstellensatz Formulation

Fix a base field F and a set of polynomials $\{f_i\}$ such that each f_i is in $F[x_1, \ldots, x_k]$; let $\vec{x}$ be short for $x_1, \ldots, x_k$. Then the weak form of Hilbert's Nullstellensatz states that $\{f_i\}$ has no common root over the algebraic closure $\overline{F}$ or F if and only if 1 is a linear combination of the f_i's, say $1 = \sum_i g_i f_i$ where the g_i's are also in $F[\vec{x}]$. This defines the Nullstellensatz proof system in the following manner: If the f_i's express a set of propositional conditions ϕ, then the g_i's are a refutation of ϕ. The complexity of the refutation is measured with respect to the maximum degree of $g_i f_i$.

As explained below, on size parameter n the polynomials in this chapter have $2^{n^{O(1)}}$ many variables. Each set of polynomials A will contain $x^2 - x$ for each variable x. Therefore any common zero for A is actually in F, since it is a k-tuple

of zeros and ones; thus there is no need to consider $\overline{F}$. Theorem 4.3.1 does not rely on the choice of the base field F and, in particular, does not rely the characteristic of F. It is important to be able to choose the base field to prove the separations in Corollary 4.3.2. As a side note, Theorem 4.3.1 holds when F is only a ring, since the proof never uses division. However, this yields no advantage because the separations in Corollary 4.3.2 follow from existing upper and lower bounds, which require F to be a field.

Definition 4.2.1. *Suppose p is a polynomial and $F = \{f_i\}$ is a set of polynomials. Write $p \in \langle F, d\rangle$ if and only if there exist polynomials g_i such that $p = \sum_i g_i f_i$ and the maximum degree of the $g_i f_i$'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 a $r \in \langle F, d\rangle$ such that $p = q + r$.*

It should be noted that Hilbert's Nullstellensatz is never used in any of the proofs below. The original version of the Nullstellensatz gives no upper bound on the degree of a refutation of a set of unsatisfiable $\{f_i\}$. Much later, several authors proved various general degree upper bounds for Nullstellensatz refutations [7, 14, 25]. When the base field is arbitrary the upper bound is exponential in the number of variables; this bound can be improved when the base field is $\mathbb{C}$, though it is still exponential. However, since the applications of Theorem 4.3.1 have $2^{n^{O(1)}}$ variables on size parameter n, these exponential bounds on the degree are $2^{2^{n^{O(1)}}}$. This not useful in proving the separations below because Theorem 4.3.1 requires $n^{O(1)}$ degree Nullstellensatz refutations.

Let Q_Φ be a basic first-order TFNP^2 problem, so that Φ is the $\exists$-sentence $\exists\vec{x}\phi(\vec{x})$, over a basic language with one uninterpreted function symbol f. A Nullstellensatz proof that Q_Φ is total will actually be a Nullstellensatz refutation of the fact that Q_Φ is not total. The set of polynomials $F_{\Phi,n}$ defined below encodes, when the polynomials are simultaneously set 0, 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 for $\vec{u}, v \in U_n$; in particular, there are $2^{n^{O(1)}}$ many variables. The intended meaning is that $x_{\vec{u},v}$ is 1 if $f(\vec{u}) = v$ and $x_{\vec{u},v}$ is 0 if $f(\vec{u}) \neq v$. The following set of formulas ensure that the $x_{\vec{u},v}$'s are 0/1-valued.

Definition 4.2.2. *Let Φ be an $\exists$-sentence over a basic language with one uninterpreted, k-ary function symbol. 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 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 4.2.3. *Let Φ be an $\exists$-sentence over a basic language with one uninterpreted, k-ary function symbol, let $\vec{u}$ be a vector of k elements of U_n. Then $Tot_{\Phi,n}$ is $\{\sum_{v \in U_n} x_{\vec{u},v} - 1 | \vec{u} \in U_n\}$ and $Func_{\Phi,n}$ is $\{x_{\vec{u},v} x_{\vec{u},v'} | \vec{u} \in U_n, v \neq v' \in U_n\}$.*

Note that the notation $Tot_{\Phi,n}$ and $Func_{\Phi,n}$ is identical to that of the LK formulation of Section 3.2. The notation is overloaded to emphasize the similarities between the proofs in Chapters 3 and 4. There will never be an ambiguity in usage, since context will always make it clear whether the sets contain polynomials or propositional formulas.

If $Tot_{\Phi,n}$ is satisfied then f is total, and if $Func_{\Phi,n}$ is satisfied then f is single-valued. Further, $x^2_{\vec{u},v} - x_{\vec{u},v}$ is in $\langle Tot_{\Phi,n} \cup Func_{\Phi,n}, 2 \rangle$ since

$$x_{\vec{u},v} \left(\sum_{v' \in U_n} x_{\vec{u},v'} - 1 \right) = x^2_{\vec{u},v} - x_{\vec{u},v} + \sum_{\substack{v' \in U_n \\ v \neq v'}} x_{\vec{u},v} x_{\vec{u},v'}.$$

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 $n^{O(1)}$, $Bool_{\Phi,n}$ is omitted from the remaining definitions.

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 3.2.

Definition 4.2.4. *Suppose Q_Φ is a basic first-order* TFNP^2 *problem. That is, the language contains one uninterpreted, k-ary function symbol f and Φ is in disjunctive normal form $\exists \vec{x} \bigvee_{j=1}^{J} \phi_j(\vec{x})$. Furthermore, each ϕ_j is a conjunction of literals $\ell_{j,1}, \ldots, \ell_{j,i_j}$ such that each literal is of the form $g(\vec{a}) = b$ or $b = c$ or $b \neq c$, 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. 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 $\prod_{i=1}^{i_j} p_{\vec{a}}(\ell_{j,i})$, 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 $\mathit{Failure}_{\Phi,n}$ 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 $\mathit{Tot}_{\Phi,n}$, $\mathit{Func}_{\Phi,n}$, and $\mathit{Failure}_{\Phi,n}$.

We abuse notation with respect to $F_{\Phi,n}$ and named problems as in Chapter 3. Namely, if Φ is the $\exists$-sentence such that Q_Φ is ITER, we write $F_{\text{ITER},n}$ instead of $F_{\Phi,n}$. Also note that in the definition of $p_{\vec{a}}(\ell_{j,i})$ the case of a negated variable is not addressed. This is possible since Q_Φ is a basic first-order TFNP^2 problem and thus $\ell_{j,i}$ is never a negated variable.

If $p_{\vec{a}}(\neg\phi_j)$ is set 0, then some factor $p_{\vec{a}}(\ell_{j,i})$ of $p_{\vec{a}}(\neg\phi_j)$ is 0. There are two ways $p_{\vec{a}}(\ell_{j,i})$ 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 $\mathit{Failure}_{\Phi,n}$ is satisfied each solution to Q_Φ is ruled out.

As an example of this construction consider the case of LONELY. Recall that LONELY can be expressed by the basic first-order TFNP^2 problem $Q_{\Phi'}$, where Φ' is

$$\exists x, y, z[(f(0) = x \wedge x \neq 0) \vee (y = f(x) \wedge z = f(y) \wedge x \neq z) \vee (x \neq 0 \wedge f(x) = x)].$$

Consider an assignment to the $x_{\vec{u},v}$'s that satisfies $F_{\text{LONELY},n}$. The first disjunct of Φ' contributes $x_{0^n,u}$ to $\mathit{Failure}_{\text{LONELY},n}$ for each $u \in U_n$ not equal to 0^n. When set to 0, these polynomials express that $f(0^n) \neq u$ for all $u \neq 0^n$. Since $\sum_{v \in U_n} x_{0^n,v} - 1$ is in $\mathit{Tot}_{\text{LONELY},n}$ and is also satisfied, it must be that $x_{0^n,0^n} = 1$, and so $f(0^n) = 0^n$. This rules out solving $Q_{\Phi'}$ by satisfying the first disjunct. Similarly, the second disjunct of Φ' contributes $x_{u,v}x_{v,w}$ to $\mathit{Failure}_{\text{LONELY},n}$ for each choice of $u, v, w \in U_n$ such that $u \neq w$. When set to 0, these polynomials express that there is no u, v, w

such that $u \neq w$ and $f(f(u)) = w$; that is $f(f(u)) = u$ for all u, which rules out solving $Q_{\Phi'}$ by the second disjunct. Finally, the last disjunct contributes $x_{u,u}$ to $\mathit{Failure}_{\text{LONELY},n}$ for all $u \neq 0^n$, which expresses that $f(u) \neq u$ for all $u \neq 0^n$. Thus all possible solutions to Q_Φ are ruled out. Therefore, satisfying $F_{\text{LONELY},n}$ is equivalent to saying that LONELY is *not* total. Since LONELY *is* total, the set $F_{\text{LONELY},n}$ must be unsatisfiable, and hence, $1 \in \langle F_{\text{LONELY},n}, d\rangle$ for some d; in fact, this statement is shown in [9] to hold for $d = n^{O(1)}$.

4.3 Main Theorem and Separations

Theorem 4.3.1. *Let Q_Φ and Q_Ψ be basic first-order* TFNP^2 *problems, and suppose $Q_\Phi \leq_{\text{T}} Q_\Psi$. If $1 \in \langle F_{\Psi,m}, m^{O(1)}\rangle$, then $1 \in \langle F_{\Phi,n}, n^{O(1)}\rangle$.*

In particular, if there are $n^{O(1)}$ degree Nullstellensatz proofs that Q_Ψ is total, and any Nullstellensatz proof that Q_Φ is total requires degree $2^{n^{O(1)}}$, then Theorem 4.3.1 implies that $Q_\Phi \not\leq_{\text{T}} Q_\Psi$. This fact is used to obtain the separations in Corollary 4.3.2 below.

It is worth discussing in more detail the relation between Theorem 4.3.1 and the results of Section 3.3, namely Theorem 3.3.1 and Corollary 3.3.2. Theorem 3.3.1 states if $Q_\Phi \leq_{\text{T}} Q_\Psi$ then there are well-behaved LK proofs of the totality of Φ using (substitution instances of) the totality of Ψ as an initial sequent. Thus, if there are actual proofs of the totality of Ψ on hand, it is easy to combine all the proofs into a proof of the totality of Φ, with only logical initial sequents; this is the content of Corollary 3.3.2. Note that Corollary 3.3.2 has stronger hypotheses than Theorem 3.3.1 (another proof is assumed to exist), but also has a stronger conclusion (the constructed proof has no non-logical initial sequents). Theorem 4.3.1 follows the structure of the argument of Corollary 3.3.2 in the Nullstellensatz setting. It is worth noting that the proof that LONELY $\not\leq_{\text{T}}$ PIGEON in Corollary 3.3.3 requires the use of Theorem 3.3.1, as opposed to Corollary 3.3.2, since the result from [5] deals with proofs with non-logical initial sequents.

Is it possible to prove a stronger result in the Nullstellensatz setting that is analogous to Theorem 3.3.1? To prove such a result, it would be necessary to

have a Nullstellensatz notion corresponding to a non-logical initial sequent in LK proofs. This would require something to allow the free use of some new polynomial in the Nullstellensatz proof. However, a Nullstellensatz proof does not have enough structure to allow such a notion.

To expand on this point, consider a Nullstellensatz proof of p from $F = \{f_i\}$; the only information in the proof is the coefficient g_i of f_i for each i. Now consider adding a new polynomial q supposed to represent a Nullstellensatz non-logical initial sequent; call such a polynomial a *non-logical initial polynomial*. To mimic the initial sequent aspect, let the degree of q not count toward the degree of the Nullstellensatz proof. Then q functions like a placeholder for a Nullstellensatz proof of q, which is exactly what a non-logical initial sequent does for an LK proof. There are two ways in which q could help in the proof of p, either (1) adding q to the set F, or (2) using q to help derive each g_i.

Let F' be F with q added, so that a Nullstellensatz proof of p from F' has one more term in the sum for p, namely $p = g_q q + \sum_i g_i f_i$. Then q acts like a non-logical initial sequent in an LK proof. To make this clear, consider the following property of LK proofs. Suppose there is an LK proof $\mathcal{P}_1$ of $\Gamma \longrightarrow \Delta$ that makes use of (perhaps multiple instances of) a non-logical initial sequent $\Pi \longrightarrow \Sigma$. If there is an LK proof (one that only uses logical initial sequents) $\mathcal{P}_2$ of $\Pi \longrightarrow \Sigma$, then replacing all occurrences of $\Pi \longrightarrow \Sigma$ in $\mathcal{P}_1$ by the proof $\mathcal{P}_2$ yields an LK proof of $\Gamma \longrightarrow \Delta$ with only logical initial sequents. Now consider $p = g_q q + \sum_i g_i f_i$. Given a Nullstellensatz proof of q from F, say $q = \sum_i h_i f_i$ for some polynomials h_i, there is a Nullstellensatz proof of p from F, $p = \sum_i (g_i + h_i) f_i$. In this sense, q acts just like a non-logical initial sequent in an LK proof.

However, the analogy with LK proofs is not perfect. A LK proof could use a non-logical initial sequent multiple times, and the replacement property described in the preceding paragraph still holds. But in the sum $p = g_q q + \sum_i g_i f_i$, q is used as a non-logical initial polynomial just once. This is because we have only considered case (1). It is possible that q could help in deriving the g_i's as in case (2), which could decrease the degree of the proof. As an extreme example consider the case where g_i is q for all i and $g_q = 1$, so that $p = q + \sum_i q f_i$. When viewing q as

a non-logical initial polynomial, this derivation has degree equal to the maximum degree of the f_i's (since q does not count towards the degree of the proof), but when q is not considered as a non-logical initial polynomial, the derivation has degree equal to the maximum degree of the qf_i's and q, which is strictly larger (assuming q is not constant).

If the dependence of the g_i's on q were known, then substituting q with a Nullstellensatz proof of q from F would make the analogy with LK proofs complete. However, it seems fundamental to Nullstellensatz proofs that the exact structure of the g_i's is not explicitly known. The complexity of the expression for the g_i's is not what a Nullstellensatz proof measures, only the final degree. Seemingly, part of the power of Nullstellensatz proofs comes from this fact. Thus the possibility of case (2) rules out non-logical initial polynomials.

It might be hoped that case (2) never arises in a proof of Theorem 4.3.1 where polynomials expressing the totality of Q_Ψ are added as non-logical initial polynomials. However, because M is a *Turing* reduction there would be multiple such Nullstellensatz non-logical initial sequents corresponding to the multiple calls to Q_Ψ by M. Since the calls to Q_Ψ by M are related, different answers to a Q_Ψ query can lead to different Q_Ψ queries later, it would be expected that these initial sequents would have to interact as in case (2). In fact, this is exactly what happens when attempting to alter the proof of Theorem 4.3.1 in this manner.

A complication to this discussion is Theorem 13 of [9] which *is* a Nullstellensatz analogue to Theorem 3.3.1, assuming the model extension property and only a many-one reduction M between Q_Φ and Q_Ψ. The model extension property makes it possible to assume that M makes its first, and only, call to Q_Ψ, by enlarging the domain of all possible Q_Ψ calls to one common size. This fact imposes enough structure on M to ensure that the dependence of the g_i's is known, so that case (2) can be dealt with in an easy manner when it arises.

One might consider overcoming the difficulties presented by case (2) by expanding reworking Nullstellensatz proofs to use inference rules, obtaining a system similar to the polynomial calculus. However, trying to tailor a proof system for a particular result leads to rules which seem artificial at best. In addition, this

approach leads to a blending of Nullstellensatz into propositional LK. For instance, adding inference rules to Nullstellensatz adds height to the proofs, which more similar to LK proofs. Having an analogue to Theorem 3.3.1 is balanced against the desire to have a distinct natural proof system. We choose to keep Nullstellensatz proofs defined as above, and settle for a Nullstellensatz version of Corollary 3.3.2 but not Theorem 3.3.1.

Given this choice, it is still possible to prove separations using Theorem 4.3.1 with existing upper and lower bounds on the degree of Nullstellensatz refutations. The separations (a) and (b) were shown by direct methods in [2]. The separations (c) and (d) are new; they were previously known only with respect to many-one reducibility [9].

Corollary 4.3.2. (a) PIGEON $\not\leq_{\mathrm{T}}$ OntoPIGEON. [2]
(b) PIGEON $\not\leq_{\mathrm{T}}$ LONELY. [2]
(c) ITER $\not\leq_{\mathrm{T}}$ OntoPIGEON.
(d) ITER $\not\leq_{\mathrm{T}}$ LONELY.

Proof. It is shown in [2] that $F_{\mathrm{PIGEON},n}$ requires polynomial degree (in 2^n) degree Nullstellensatz refutations and [10, 16] shows the same for $F_{\mathrm{ITER},n}$. On the other hand, [9] shows that $F_{\mathrm{OntoPIGEON},n}$ has polylogarithmic degree Nullstellensatz refutations over any field and that $F_{\mathrm{LONELY},n}$ has polylogarithmic degree Nullstellensatz refutations over fields with characteristic 2. Thus the separations hold by Theorem 4.3.1. □

Note that since OntoPIGEON $\leq_{\mathrm{T}}$ LONELY one could also prove ITER $\not\leq_{\mathrm{T}}$ OntoPIGEON from ITER $\not\leq_{\mathrm{T}}$ 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 [9].

4.4 Proof of Theorem 4.3.1

The proof of Theorem 4.3.1 closely follows the proof of Theorem 3.3.4. Suppose $M : Q_\Phi \leq_{\mathrm{T}} Q_\Psi$ and that the type-1 input to M is α and the size

parameter is n. For each query $Q_\Psi(\beta, 0^m)$ that M makes, there is a corresponding substitution σ defined by the (α, n)-decision trees for the $\beta(\vec{w})$'s. In the context of LK proofs, σ was replaced $x_{\vec{w},y}$ with a big (size $2^{n^{O(1)}}$) disjunction of small (size $n^{O(1)}$) conjunctions of $x_{\vec{u},v}$'s. Now σ replaces the polynomial $x_{\vec{w},y}$ with $\lambda_{\vec{w},y}$, a sum of polynomials of degree $n^{O(1)}$ (note that this sum will have exponentially many terms, even though there is no notion of a "big" sum in Nullstellensatz proofs). Lemma 4.4.2 proves that the substitution is total, namely that for each $\vec{w} \in U_m$,

$$\left(\sum_{y \in U_m} \lambda_{\vec{w},y} - 1\right)\sigma \in \langle F_{\Phi,n}, n^{O(1)}\rangle.$$

Lemma 4.4.3 proves a corresponding result about the functionality of the substitution. These two lemmas are analogous to Lemma 3.5.1. The main technical result needed to prove 4.3.1 involves traces and is shown in Lemma 4.4.4 and is analogous to Lemma 3.5.2; Theorem 4.3.1 is just the special case of Lemma 4.4.4 applied to the trace to the root.

In Chapter 3, branches in (α, n)-decision trees corresponded to conjunctions; they are now associated with polynomials.

Definition 4.4.1. *If P is a branch in an (α, n)-decision tree, then we identify P with the product*

$$\prod_{i=1}^{I} x_{\vec{u}_i, v_i},$$

where $\{\alpha(\vec{u}_i) = v_i\}_{i=1}^{I}$ is the set of α values set by the edge labels in P.

A family of (α, n)-decision trees, $\mathcal{T} = \{T_{\vec{w}} | \vec{w} \in U_m\}$, induces a substitution $\sigma_\mathcal{T}$ in much the same manner as in Chapter 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 $p(\lambda_{\mathcal{T},\vec{w}_1,y_1}, \lambda_{\mathcal{T},\vec{w}_2,y_2}, \ldots)$.

Lemma 4.4.2. *Let Q_Φ and Q_Ψ be basic first-order* TFNP^2 *problems. Let $\mathcal{T} = \{T_{\vec{w}} | \vec{w} \in U_m\}$ 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 of the claim 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

$$\sum_{P \in \mathrm{br}(T)} P - 1 = \sum_{\substack{P' \in \mathrm{br}(T') \\ P' \neq P'_\mu}} P' + P'_\mu \sum_{v \in U_n} x_{\vec{u},v} - 1 =$$

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

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

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

since $|P'_\mu| + 1 \leq d$. The claim follows since d' is bounded by the height of T.

To prove the lemma let $p \in (Tot_{\Psi,m_i})\sigma_{\mathcal{T}}$ 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 $p \in \langle Tot_{\Phi,n}, d \rangle$, since the height of $T_{\vec{w}}$ is at most d. The lemma follows since $Tot_{\Phi,n} \subseteq F_{\Phi,n}$. □

Lemma 4.4.3. *Let Q_Φ and Q_Ψ be basic first-order* TFNP^2 *problems. Let $\mathcal{T} = \{T_{\vec{w}} | \vec{w} \in U_m\}$ be a family of (α, n)-decision trees and 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 $(Func_{\Psi,m})\sigma_{\mathcal{T}}$ is a sum of monomials of the form $P_1 P_2$, 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 branches in the same decision tree, 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 $P_1 P_2 \in \langle Func_{\Phi,n}, n^{O(1)} \rangle$, and the result follows. □

Recall the definition of a trace in Definition 3.4.3. If Λ is a trace, then identify Λ with the polynomial $\prod_{x_{\vec{u},v} \in \Lambda} x_{\vec{u},v}$.

Lemma 4.4.4. *Let Q_Φ and Q_Ψ be basic first-order* TFNP^2 *problems, and suppose $M : Q_\Phi \leq_{\mathrm{T}} Q_\Psi$. Let μ be a node in $T_{M,n}$, and let Λ be a trace to μ. If $1 \in \langle F_{\Psi,m}, m^{O(1)} \rangle$, then $\Lambda \in \langle F_{\Phi,n}, n^{O(1)} \rangle$.*

Proof. The proof proceeds in a manner similar to the proof of Lemma 3.5.2. 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 $\Lambda \in \langle Func_{\Phi,n}, n^{O(1)} \rangle \subseteq \langle F_{\Phi,n}, n^{O(1)} \rangle$.

For traces that are not contradictory, the result is proved by induction on the depth h_μ of μ in the decision tree $T_{M,n}$. Specifically, the induction hypothesis is that if μ is in $T_{M,n}$ and Λ is a trace to μ, then $\Lambda \in \langle F_{\Phi,n}, n^{O(1)}(1 + h_\mu) \rangle$; the lemma follows since h_μ is $n^{O(1)}$.

For the base case let μ be a leaf of $T_{M,n}$ and Λ be a trace to μ. As in the proof of Lemma 3.5.2, 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, thus p is a product of $x_{\vec{u},v}$'s. 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, it follows that $\Lambda \in \langle p, n^{O(1)} \rangle \subseteq \langle F_{\Phi,n}, n^{O(1)} \rangle$.

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

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

Rewrite Λ as

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

Thus, $\Lambda \in \langle F_{\Phi,n}, S \rangle + \langle Tot_{\Phi,n}, n^{O(1)} \rangle$, where S is the maximum of $n^{O(1)}(1 + h_{\mu'})$ taken over all children μ' of μ. Since $S + n^{O(1)}$ is

$$n^{O(1)}(\max\{1 + h_{\mu'} | \mu' \text{ is a child of } \mu\} + 1) = n^{O(1)}(1 + h_\mu),$$

the result follows.

Now let μ be labeled $Q_\Psi(\beta, 0^m)$. Let β be computed by a Turing machine M', so that M' generates a family of (α, n)-decision trees $\mathcal{T}$. Let σ be $\sigma_\mathcal{T}$.

Let μ' be a child of μ such that the edge from μ to μ' is labeled $(j, \vec{a})$. Let $p = p_{\vec{a}}(\neg\psi_j) \in \mathit{Failure}_{\Psi,m}$ and suppose p is $\prod_{i=1}^{I} x_{\vec{w}_i,y_i}$. Then $\Lambda(p\sigma)$ is

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

By induction each term of this sum is in $\langle F_{\Phi,n}, n^{O(1)}(1 + h_{\mu'})\rangle$. As j and $\vec{a}$ range over all possible values, p ranges over all member of $\mathit{Failure}_{\Psi,m}$. Thus

$$\Lambda(p\sigma) \in \langle F_{\Phi,n}, n^{O(1)}(1 + h_{\mu'})\rangle \tag{4.1}$$

for all $p \in \mathit{Failure}_{\Psi,m}$. (If p is 0 then the induction fails as then there would be no trace to μ'. However, in this case $\Lambda(p\sigma) = 0$ and is trivially in $\langle F_{\Phi,n}, n^{O(1)}(1+h_{\mu'})\rangle$.)

Note that $1 \in \langle F_{\Psi,m}, m^{O(1)}\rangle$ implies $1 \in \langle (F_{\Psi,m})\sigma, m^{O(1)} + n^{O(1)}\rangle$ by treating the $\lambda_{\mathcal{T},\vec{w},y}$'s as variables. Write

$$1 \quad = \quad \sum_{g_i \in \mathit{Tot}_{\Psi,m} \cup \mathit{Func}_{\Psi,m}} f_i(g_i\sigma) \quad + \quad \sum_{p_i \in \mathit{Failure}_{\Psi,m}} q_i(p_i\sigma). \tag{4.2}$$

By Lemmas 4.4.2 and 4.4.3, the first summation in (4.2) is in $\langle F_{\Phi,n}, n^{O(1)}\rangle$. Since $m = n^{O(1)}$, multiplying both sides of (4.2) by Λ and making use of (4.1) yields

$$\Lambda \in \langle F_{\Phi,n}, n^{O(1)}\rangle + \langle F_{\Phi,n}, S\rangle \subseteq \langle F_{\Phi,n}, n^{O(1)} + S\rangle,$$

where S is the maximum of $n^{O(1)}(1 + h_{\mu'})$ taken over all children μ' of μ. The induction is finished since

$$S + n^{O(1)} = n^{O(1)}(\max\{1 + h_{\mu'} | \mu' \text{ is a child of } \mu\} + 1) = n^{O(1)}(1 + h_\mu).$$

□

This chapter contains material from the paper "Propositional proofs and reductions between NP search problems" which is currently accepted for publication by the Annals of Pure and Applied Logic. This paper is co-authored by the dissertation author and Samuel R. Buss.

Chapter 5

Propositional LK Reversal

5.1 Discussion

This chapter proves a converse of Theorem 3.3.4, namely, that a Turing reduction between NP search problems can be extracted from proofs of the sequents (a), (b), and (c) of Theorem 3.3.4. For convenience during this discussion, these sequents are:

(a) $(\bigwedge Tot_{\Phi,n}) \longrightarrow (\bigwedge Tot_{\Psi,m_i})\,\sigma_i$, for each $i = 1, \ldots, r$,

(b) $(\bigwedge Func_{\Phi,n}) \longrightarrow (\bigwedge Func_{\Psi,m_i})\,\sigma_i$, for each $i = 1, \ldots, r$,

(c) $\bigwedge_{i=1}^{r} (\bigvee Soln_{\Psi,m_i})\,\sigma_i, F_{\Phi,n} \longrightarrow G_{\Phi,n}$.

In fact, the reversal is slightly stronger result, since Theorem 5.2.1 uses a weaker condition (a′) in place of (a), and (b) is omitted 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 Theorem 3.3.4 is reversed and not Theorem 3.3.1. The following two facts are relevant: First, the substitutions σ_i are depth 1.5 and, moreover, 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 $\bigwedge_{y' \neq y} \overline{x}_{\vec{w},y'}$. 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 $\beta_i(\vec{w})$, 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.) Theorem 3.3.1 does not have anything similar to sequents (a) and (b), and thus there are no algorithms for $\beta_i(\vec{w})$. Without being able to compute values for β_i, there is no way to input functions to a Q_Ψ call, and thus there is no way to build a reduction from Q_Φ to Q_Ψ.

We illustrate the interaction of the substitutions σ_i and sequents (a) and (b) in a more general setting by defining the notion of a "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.[1] 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

$$\Gamma \longrightarrow \Delta, x\sigma', \overline{x}\sigma''. \tag{5.1}$$

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

$$\Gamma \longrightarrow \Delta, \overline{x}\sigma', x\sigma''. \tag{5.2}$$

Conditions (ii) and (iii) ensure that either $\Gamma \longrightarrow \Delta$ is true, or $x\sigma'$ is equivalent to $\overline{x}\sigma''$. In particular, when proving $\Gamma \longrightarrow \Delta$, it can be assumed $x\sigma'$ and $\overline{x}\sigma''$ are not

[1] We never use decision tree substitutions in this general setting, but it should help motivate the formulation of Theorem 5.2.1.

equivalent. That is to say that $x\sigma'$ and $x\sigma''$ are equivalent. Since $x\sigma'$ and $\overline{x}\sigma''$ are disjunctions of small conjunctions, this means 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_i, and when $\Gamma = F_{\Phi,n}$ and $\Delta = G_{\Phi,n}$, the sequents (5.1) and (5.2), respectively, become

$$F_{\Phi,n} \longrightarrow G_{\Phi,n}, \left(\bigwedge Tot_{\Psi,m_i}\right)\sigma_i \tag{5.3}$$

and

$$F_{\Phi,n} \longrightarrow G_{\Phi,n}, \left(\bigwedge Func_{\Psi,m_i}\right)\sigma_i. \tag{5.4}$$

To understand this, let the variable x in (5.1) and (5.2) be $x_{\vec{w},y}$, let its negation $\overline{x}$ be $\bigvee_{y'\neq y} x_{\vec{w},y'}$, let $x_{\vec{w},y}\sigma'$ be $\bigvee_{P\in \mathrm{br}_y(T_{\vec{w}})} P$, and let $x_{\vec{w},y}\sigma''$

$$\bigwedge_{\substack{P\in \mathrm{br}_{y'}(T_{\vec{w}})\\ y'\neq y}} \overline{P}.$$

Then $x_{\vec{w},y}\sigma' \vee \overline{x}_{\vec{w},y}\sigma''$ is

$$\bigvee_{P\in \mathrm{br}_y(T_{\vec{w}})} P \quad \vee \quad \bigvee_{\substack{P\in \mathrm{br}_{y'}(T_{\vec{w}})\\ y'\neq y}} P \quad = \quad \bigvee_{P\in \mathrm{br}(T_{\vec{w}})} P.$$

Thus it is easy to derive (5.3) from by (5.1) applying $\vee$:right and $\wedge$:right inferences. Similarly, $\overline{x}_{\vec{w},y}\sigma' \vee x_{\vec{w}y,}\sigma''$ is

$$\bigwedge_{P\in \mathrm{br}_y(T_{\vec{w}})} \overline{P} \quad \vee \quad \bigwedge_{\substack{P\in \mathrm{br}_{y'}(T_{\vec{w}})\\ y'\neq y}} \overline{P}.$$

Similarly, it is easy to derive (5.4) from (5.2) by applying $\vee$:right and $\wedge$:right inferences. Note how (5.3) and (5.4) are weaker than the sequents (a) and (b) of Theorem 3.3.4.

This discussion here and in Section 4.3 is relevant to the lack of a reversal of the Nullstellensatz translation in Theorem 4.3.1. Lemmas 4.4.2 and 4.4.3 prove the Nullstellensatz version of the sequents (a) and (b). However, the result which states if a Boolean formula and its negation have small disjunctive normal forms then the

formula has a decision tree does not have an analogue with polynomials. If one were to try to prove such a result for polynomials in the context of a Nullstellensatz reversal, an immediate problem is the lack of structure in Nullstellensatz proofs. There is no apparent way to take the coefficient polynomials in a Nullstellensatz proof and construct a meaningful decision tree from them.

One could try to construct decision trees from Nullstellensatz proofs by adding inference rules to the Nullstellensatz calculus. This approach was discussed in Section 4.3 as a way to obtain a Nullstellensatz result analogous to Theorem 3.3.1. The addition of inference rules to the Nullstellensatz calculus would create a tree structure to Nullstellensatz proofs, and then a traversal could give rise to relevant decision trees. However as Section 4.3 explains, such an augmented Nullstellensatz calculus is unnatural, and so we do not adopt this approach.

In fact, formulating a Nullstellensatz reversal is not even desirable because of the existence of an LK reversal. To understand why, suppose there were a sensible Nullstellensatz reversal. Then Nullstellensatz proofs could be reversed to a reduction, and the LK translation would give equivalent LK proofs. Similarly, LK proofs could be reversed to give a reduction, and the Nullstellenstaz translation would give equivalent Nullstellensatz proofs. Defining an extended Nullstellensatz system in order to get a reversal would, in the end, give a system equivalent to propositional LK. This would serve no purpose, and so we do not pursue a Nullstellensatz reversal.

5.2 Main Result and Proof

This section states and proves the reversal of Theorem 3.3.4. It is somewhat stronger than just the converse of Theorem 3.3.4 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). The LK proofs are required to be uniform: this has the usual meaning that there is a polynomial time algorithm which can calculate the structure and syntactic form 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 5.2.1. *Let Q_Φ and Q_Ψ be basic first-order* TFNP^2 *problems. Suppose 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 there are polynomial time uniform* LK *proofs of size $2^{n^{O(1)}}$ and height $n^{O(1)}$ of the following sequents:*

(a′) $F_{\Phi,n} \longrightarrow G_{\Phi,n}, (\bigwedge Tot_{\Psi,m_i})\,\sigma_i$, *for each* $i = 1, \ldots, r$, *and*

(c) $\bigwedge_{i=1}^{r} (\bigvee Soln_{\Psi,m_i})\,\sigma_i, F_{\Phi,n} \longrightarrow G_{\Phi,n}$.

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.*

Note that the only negations that occur in sequents (a) and (b) are negated variables that are introduced by the substitutions σ_i.

The sequent (b)

$$\left(\bigwedge Func_{\Phi,n}\right) \longrightarrow \left(\bigwedge Func_{\Psi,m_i}\right)\sigma_i$$

of Theorem 3.3.4 can be omitted in Theorem 5.2.1 because of the lack of negated literals in the definition basic first-order TFNP^2 problems. As an example, suppose Φ is the formula $\exists x[f(x) \neq 0 \vee \phi(x)]$ such that Φ is a total $\exists$-sentence. Let $Q_{\Phi'}$ be the *basic* first-order TFNP^2 problem equivalent to Q_Φ constructed in Section 2.4, so that Φ' is

$$\exists x, y[(f(x) = y \wedge y \neq 0) \vee \phi(x)].$$

Suppose a is a solution to $Q_\Phi(f, 0^n)$, and that $f(a) = b \neq 0$, and suppose $\phi(a)$ does not hold. If f were allowed to be a multifunction, then it is possible to add 0^n to the list of values for $f(a)$, and thus a would no longer solve $Q_\Phi(f, 0^n)$. However, a still solves $Q_{\Phi'}(f, 0^n)$ even if the graph of f were augmented to be a multifunction, because there is always one value of $f(a)$ that is not 0^n. Since Theorem 5.2.1 involves basic first-order TFNP^2 problems, functionality is not necessary, and sequent (b)

can be omitted. A similar issue regarding functionality and basic first-order TFNP^2 problems appears in the proof of Lemma 8.2.3.

We fix the conventions for the proof of Theorem 5.2.1. Since Q_Φ is a basic first-order TFNP^2 problem, Φ is a total $\exists$-sentence over a language with one uninterpreted function symbol; let α be this function symbol, and let the arity of α be k. Thus an input to Q_Φ is of the form $(\alpha, 0^n)$. Assume the arity of solutions to Q_Φ is s, so that $Q_\Phi(\alpha, 0^n) \subseteq U_n^s$ and $\alpha : U_n^k \to U_n$. Similarly, Ψ is a total $\exists$-sentence over a language with uninterpreted function symbol, call it β, and let the arity of β be k'. Assume that the arity of solutions to Q_Ψ is s', so that $Q_\Psi(\beta, 0^m) \subseteq U_m^{s'}$ and $\beta : U_m^{k'} \to U_m$. 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 are conjunctions that may contain negated literals and the P's may no longer explicitly correspond to paths in a decision tree.

Theorem 5.2.1 is proved 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 5.2.2 shows how to construct β_i by traversing the proof of the sequent (a$'$). Once β_i is defined, 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 5.2.2 will define 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 $\bigvee_{\vec{w} \in U_{m_i}} \bigvee_{P \in \mathcal{D}_{i,\vec{w},y}} P$, a member of $Tot_{\Psi,m_i}\sigma_i$, then $\beta_i(\vec{w})$ is defined to be y. If A is a disjunct of $[\![\phi_j]\!]_{\vec{a}}$, a member of $Soln_{\Phi,n}$, then the computation of $\beta_i(\vec{w})$ has failed; however, this is not a problem since in this case we have already solved $Q_\Phi(\alpha, 0^n)$. Lemma 5.2.2 formalizes this intuition by constructing two functions β_i and γ_i from the proof of sequent (a$'$) such that if the computation of $\beta_i(\vec{w})$ fails

then $\gamma_i(\vec{w})$ computes a solution $Q_\Phi(\alpha, 0^n)$.

Lemma 5.2.2. *Assume there are proofs of* (a′) *as in Theorem 5.2.1. Then, for each* $i = 1, \ldots, r$*, there 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* $\vec{w} \in U_{m_i}$ *and suppose* $\gamma_i(\vec{w}) = \vec{a} \notin Q_\Phi(\alpha, 0^n)$*. Then, if* $\beta_i(\vec{w}) = y$*, the computation of* $\beta_i(\vec{w})$ *specifies enough of* α *to satisfy some* $P \in \mathcal{D}_{i,\vec{w},y}$*.*

Proof. We describe an algorithm that calculates the values $\beta_i(\vec{w})$ and $\gamma_i(\vec{w})$ simultaneously. The algorithm traverses a branch in the proof of (a′) starting at the endsequent and without backtracking. The traversal is constructed as 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 $\tau_t \vDash x_{\vec{u},v}$ (respectively $\tau_t \nvDash x_{\vec{u},v}$) if $\alpha_t(\vec{u}) = v$ (respectively $\alpha_t(\vec{u}) = v'$ for some $v' \neq v$). As the traversal proceeds, the partial assignment defined by α_t will make certain formulas (called "p.s.-settable") in the t^{th} sequent true or false:

Definition 5.2.3. *A formula appearing in the proof of (a′) is* p.s.-settable *provided 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* $(\bigwedge Tot_{\Psi,m_i})\,\sigma_i$*.)*
- *A cut-formula or 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 $x_{\vec{u},v}$ 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 the t^{th} sequent, one of the following hold:

(1) Every p.s.-settable formula in the t^{th} antecedent (respectively succedent) is set true (respectively false) by τ_t. Furthermore, if a disjunction $\bigvee_j A_j$ is a p.s.-settable formula in the t^{th} 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 $\gamma_i(\vec{w})$ is defined to be $\vec{a}$, and $\beta_i(\vec{w})$ is defined arbitrarily.

(3) The traversal has found y by step t such that $P \in \mathcal{D}_{i,\vec{w},y}$ is set true by τ_t. The algorithm then halts, and the value of $\beta_i(\vec{w})$ is defined to be y, and $\gamma_i(\vec{w})$ 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 $\beta_i(\vec{w})$ and $\gamma_i(\vec{w})$. Since the proof is polynomial height and each step of the traversal can be carried out in polynomial time, the traversal is 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.

We now describe how to execute a single step in the traversal. The argument 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 $(Tot_{\Psi,m_i})\,\sigma_i$ of the form

$$\bigwedge_{\vec{w}' \in W} \Big(\bigvee_{y' \in U_{m_i}} x_{\vec{w}',y'} \sigma_i \Big),$$

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

$$\bigvee_{y' \in U_{m_i}} x_{\vec{w},y'} \sigma_i.$$

Note that because of this step, the traversal never encounters a p.s.-settable formula of the form $P \in \mathcal{D}_{i,\vec{w}',y'}$ for $\vec{w}' \neq \vec{w}$.

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.

This finishes the proof of the $\wedge$:right case, but to better understand the intuition behind the traversal we further discuss the case where $A \wedge B$ is p.s.-settable. There are four ways in which $A \wedge B$ could be p.s.-settable in the succedent: it could be $[\![\phi_j]\!]_{\vec{a}}$ for $j \geq 0$ and $\vec{a} \in U_n$, $P \in \mathcal{D}_{i,\vec{w},y'}$ for $\vec{w}, y' \in U_{m_i}$, a cut-formula, or an ancestor of a cut-formula. However, we claim that the traversal ends before the first two cases could ever arise.

To see this, note that $[\![\phi_j]\!]_{\vec{a}}$ and P are disjuncts of $\bigvee Soln_{\Phi,n}$ and $\bigvee_{P \in \mathcal{D}_{i,\vec{w},y'}} P$, respectively. Thus to get to an $\wedge$:right inference that introduced either $[\![\phi_j]\!]_{\vec{a}}$ or P the traversal needs to proceed through a series of $\vee$:right inferences. However, the traversal is constructed so that at step t every p.s.-settable formula in the succedent is set true by τ_t. So by the time the traversal uncovers $[\![\phi_j]\!]_{\vec{a}}$, τ_t sets this formula true, which satisfies (2) and ends the traversal. Similarly, by the time traversal uncovers P, case (3) holds and the traversal halts.

$\vee$:left: Let S be $\Gamma, A \vee B \longrightarrow \Delta$, let S_0 be $\Gamma, A \longrightarrow \Delta$, 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. The condition (1) still holds.

$\wedge$:left: Let S be $\Gamma, A \wedge B \longrightarrow \Delta$, 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 $\bigvee_{v \in U_n} x_{\vec{u},v}$ or $\overline{x}_{\vec{u},v} \vee \overline{x}_{\vec{u},v'}$, for some $\vec{u}, v, v' \in U_n$. In either case, the traversal queries $\alpha(\vec{u})$, 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 $P \in \mathcal{D}_{i,\vec{w},y}$ 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 $P \in \mathcal{D}_{i,\vec{w},y}$, 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) move to S_1 (respectively S_0). Then (1) still holds. □

We now prove Theorem 5.2.1.

Proof of Theorem 5.2.1. 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 5.2.2. Let τ_t be as in the proof of Lemma 5.2.2. Expand the notion of p.s.-settable to include subformulas of $(\bigvee \mathit{Soln}_{\Psi,m_i})\sigma_i$. Note that $(\bigvee \mathit{Soln}_{\Psi,m_i})\sigma_i$ 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 $H = H(j,\vec{a})$ depends on j and $\vec{a}$. Extend the notion of setting a p.s.-settable formula true to include these new types of p.s.-settable formulas. (The notion of setting a p.s.-settable formula false does not need to be updated 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 providing the following hold: If A is a $x_{\vec{u},v}$ (resp. $\overline{x}_{\vec{u},v}$), then A is set true if $\tau \vDash A$ (resp. $\tau \nvDash A$). 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, $(\bigvee \mathit{Soln}_{\Psi,m_i})\,\sigma_i$ 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.

If τ sets a disjunction $\bigvee_j A_j$ true, it is important that the traversal know which disjunct is set true by τ. That way, 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.

The traversal does a polynomial amount of work at the t^{th} sequent (relative to α and Q_Ψ) and at each step one of the following holds:

(1) If A is in the the t^{th} 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 some $[\![\phi_j]\!]_{\vec{a}}$ is set true by τ_t. The algorithm then halts, and the reduction outputs $\vec{a}$.

It is clear that (1) cannot hold for the initial sequents of the proof, so the correctness of the proof implies 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 5.2.2.

$\wedge$: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 $(\bigvee Soln_{\Psi,m_i})\,\sigma_i$. Let β_i and γ_i be the functions constructed by Lemma 5.2.2. The traversal queries $Q_\Psi(\beta_i, 0^{m_i})$ and receives answer $\vec{a} \in U_{m_i}$. Since $\vec{a}$ is a solution, there is a conjunct ψ_j of Ψ such that $[\![\psi_j]\!]_{\vec{a}}$ is made true by the function β_i. Since β_i is polynomial time computable and there are only constantly many disjuncts of Ψ, the traversal can find j in polynomial time. Suppose $[\![\psi_j]\!]_{\vec{a}}$ is

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

so that $([\![\psi_j]\!]_{\vec{a}})\sigma_i$ is

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

Since β_i makes $[\![\psi_j]\!]_{\vec{a}}$ true, $\beta_i(\vec{w}_h) = y_h$ for each $h \in H$. The proof is finished if we can show that either (1) or (2) holds for S. Intuitively, (1) allows the traversal to continue while (2) solves Q_Φ.

If we were to show (1), we would need to show that $(\bigvee Soln_{\Psi,m_i})\,\sigma_i$ is set true by the current partial truth assignment τ_t. We could do this if we could find disjuncts $P_1, \ldots, P_H$ such that, for each $h = 1, \ldots, H$, $P_h \in \mathcal{D}_{i,\vec{w}_h,y_h}$ and P_h is set true by τ_t. A complication arises because even though $\beta_i(\vec{w}_h)$ is computed to be y_h, it could be that this computation does not reveal enough information about α to set

any $P_h \in \mathcal{D}_{i,\vec{w}_h,y_h}$ true. However, in this case (2) holds since $\gamma_i(\vec{u}_h)$ is constructed to solve $Q_\Phi(\alpha, 0^n)$.

More specifically, the traversal calculates $\gamma_i(\vec{w}_h)$ for each $1 \leq h \leq H$. If there is an h such that $\gamma_i(\vec{w}_h) = \vec{a}' \in Q_\Phi(\alpha, 0^n)$, then case (2) holds. This process only takes polynomial time since H is constant, γ_i is polynomial time computable, and solutions to Q_Φ can be verified in polynomial time. If no such h exists, Lemma 5.2.2 implies that, for each $h = 1, \ldots, H$, $\beta_i(\vec{w}_h) = y_h$ and enough of α has been specified to make a P_h in $\mathcal{D}_{i,\vec{w}_h,y_h}$ true. Furthermore, the computation of $\beta_i(\vec{w}_h)$ determines P_h, so that P_h is known to the traversal. This exactly says that $(\bigvee \mathit{Soln}_{\Psi,m_i})\,\sigma_i$ is p.s.-settable, and thus (1) holds. □

This chapter contains material from the paper "Propositional proofs and reductions between NP search problems" which is currently accepted for publication by the Annals of Pure and Applied Logic. This paper is co-authored by the dissertation author and Samuel R. Buss.

Chapter 6

Equivalences Between Many-one and Turing Reductions

6.1 Many-one and Turing Equivalences

This chapter shows that for many common TFNP classes Turing reducibility is equivalent to many-one reducibility. This includes the classes PPAD, PPADS, PPA, and PLS. Thus making polynomially many queries to OntoPIGEON, LeftPIGEON, LONELY, and ITER is no more stronger than making one query, respectively. It is an open question whether Turing reducibility implies many-one reducibility for the class PPP. In fact, it is open whether just two calls to PIGEON is stronger than one call.

Theorem 6.1.1. *Let* Q_1 *be in* TFNP *or* TFNP^2*, and let* Q_2 *be any of* OntoPIGEON*,* LeftPIGEON*,* LONELY*, or* ITER*. Then* $Q_1 \leq_{\mathrm{m}} Q_2$ *if and only if* $Q_1 \leq_{\mathrm{T}} Q_2$*.*

As a consequence of Theorem 6.1.1 we get the following corollary.

Corollary 6.1.2. *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.

Proof of Theorem 6.1.1. 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 [9]. 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 $u \in U_n$, and $y_1, \ldots, y_\ell, v, \in U_m$, and $\ell \leq p(n)$. Since F must take strings as arguments, we encode F's input as

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

where $n' = n + (m+1)p(n) + m$. A 1 in the $(n + (m+1)i + 1)^{\text{th}}$ position $(0 \leq i < p(n))$ indicates that y_{i+1} is an answer to the $(i+1)^{\text{st}}$ query to ITER. A 0 in the $(n+(m+1)i+1)^{\text{th}}$ postition indicates that no $(i+1)^{\text{st}}$ 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 $(\ell+1)^{\text{st}}$ call to 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 $1 \leq i \leq \ell$, $y_1, \ldots, y_{i-1}$ determines the i^{th} 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 $u \neq 0^n$ 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, suppose $u = 0^n$ and $y_1, \ldots, y_\ell$ is valid. Define $F(\langle u; y_1, \ldots, y_\ell; v\rangle)$ as follows:

1. Suppose 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 answering the first ℓ calls to ITER with $y_1, \ldots, y_\ell$ causes M to make an $(\ell+1)^{\text{st}}$ query to ITER. Suppose the query is $\text{ITER}(g, 0^m)$.

   a. If $v \in \text{ITER}(g, 0^m)$ 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 $v \notin \text{ITER}(g, 0^m)$ 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 $\text{ITER}(F, 0^{n'})$ and receives answer $w = \langle u; y_1, \ldots, y_\ell; v\rangle$.

Claim: u is a solution to $Q_1(\vec{x})$.

Assuming the claim, M' finishes by outputting u. The rest of the proof shows that the claim holds.

In general, if z is a solution to ITER on input g and with size parameter p, then there are three possibilities. Either (1) $z = 0^p$ and $g(0^p) = 0^p$, (2) $g(z) < z$, or (3) $g(z) > z$ and $g(g(z)) = g(z)$. We show that for F and w the first two cases cannot happen and that in the third case, u is a solution to $Q_1(\vec{x})$.

Consider computing $F(0^{n'})$; note the input $0^{n'}$ 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 $a \neq 0^n$ since M' checked if 0^n was a solution before calling $\text{ITER}(F, 0^{n'})$.) In the first case, the $(n+1)^{\text{st}}$ bit of $F(0^{n'})$ is 1, and in the second case the first n bits of $F(0^{n'})$ are not all zero. Therefore $F(0^{n'}) \neq 0^{n'}$.

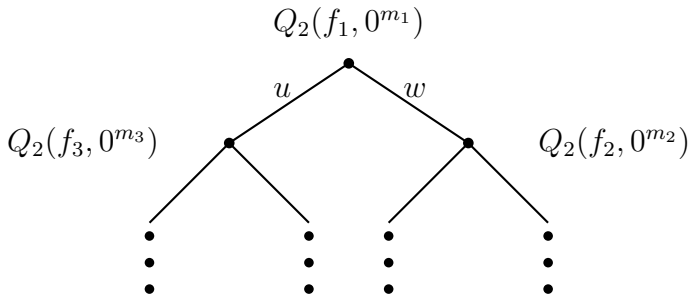

Figure 6.1: The decision tree $T_{M,n}$ for a Turing reduction M to Q_2. The only solutions to $Q_2(f_1, 0^{m_1})$ that are shown are u and w.

It is straightforward to check that the coding conventions (specifically that a 1 in the $(n + (m + 1)i + 1)^{\text{th}}$ 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.

Therefore it must be that $F(w) > w$ and $F(F(w)) = F(w)$. 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 $b_1, \ldots, b_k$ 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 $a \neq 0^n$. Then F was defined so that in this case a is a solution to $Q_1(\vec{x})$. Thus the claim is proved, and that finishes the proof of the theorem. □

The above proof relativizes easily in the case that Q_1 has a type-1 input α. The main point is that deciding if a sequence $y_1, \ldots, y_\ell$ is valid can be done in polynomial time relative to α. The reason for this is that determining the validity of a sequence essentially simulates M, and M runs in polynomial time relative to α.

There is a way to easily visualize the construction of F in Theorem 6.1.1. Let M be a Turing reduction from Q_1 to ITER. Suppose that the reduction M has decision tree $T_{M,n}$ as shown in Figure 6.1, where Q_2 is ITER. For simplicity, u and w are the only two solutions to ITER$(f_1, 0^{m_1})$ that are shown. Figure 6.2 shows the three possible instances of ITER corresponding to $(f_1, 0^{m_1})$, $(f_2, 0^{m_2})$, and $(f_3, 0^{m_3})$. The ellipses in each instance indicate that the instance is larger than what is shown.

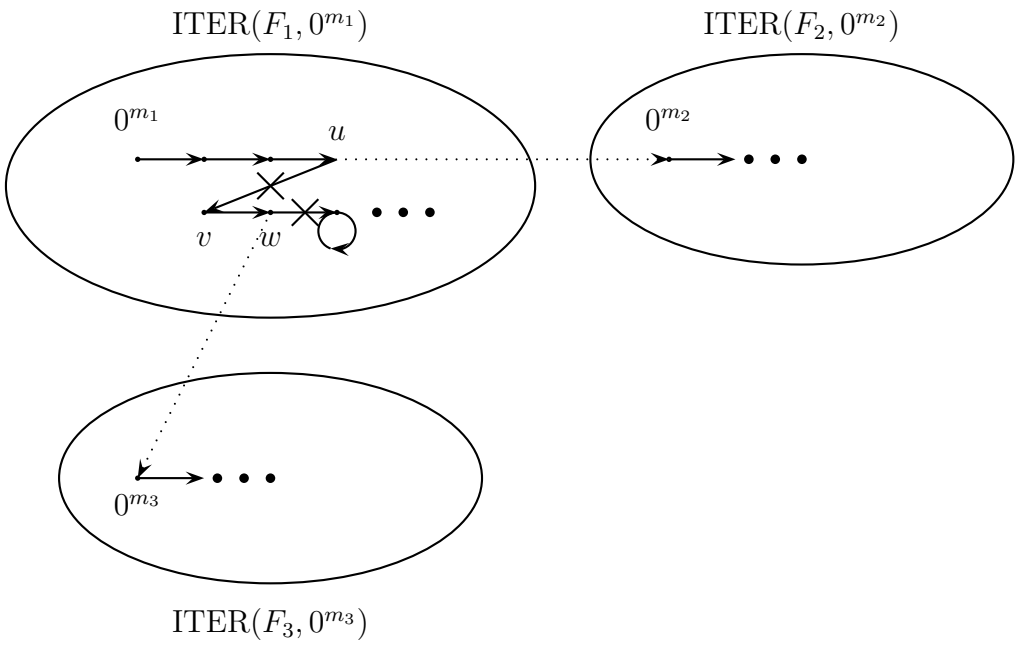

Figure 6.2: An illustration of the construction of one ITER instance from all the ITER instances in $T_{M,n}$. The outgoing edges from u and w are crossed out and replaced by the corresponding dotted edges.

The lines shown indicate the value of the respective f_i; in particular $f_1(u) = v$ and $f_1(v) = w$. Suppose that $v < u$ so that $f_1(u) < u$, which implies that u is a solution to ITER($f_1, 0^{m_1}$). Also suppose $f_1(w) > w$, and thus w is also a solution to ITER($f_1, 0^{m_1}$) since $f_1(f_1(w)) = w$. Since the solution u to ITER($f_1, 0^{m_1}$) leads M to the query ITER($f_2, 0^{m_2}$), the line from u to v is removed and a line from u to 0^{m_2} is added. Since w leads M to ITER($f_3, 0^{m_3}$), the line from w is changed to go to 0^{m_3}. Let F be the function constructed by linking the different instances of ITER in $T_{M,n}$ in this fashion. Then the only instances of ITER in $T_{M,n}$ that still have solutions correspond to the leaves of $T_{M,n}$. Therefore solving ITER on F leads to a solution of an instance ITER that appears as a leaf in $T_{M,n}$. The leaves of $T_{M,n}$ encode solutions to Q_1 since M is a correct Turing reduction, and hence this one call to ITER on F solves Q_1.

The proof of Theorem 6.1.1 for the cases OntoPIGEON, LeftPIGEON, and LONELY relies on constructions similar to the one discussed in Figures 6.1 and 6.2. Take LONELY as an example. Suppose M is a Turing reduction from Q_1 to

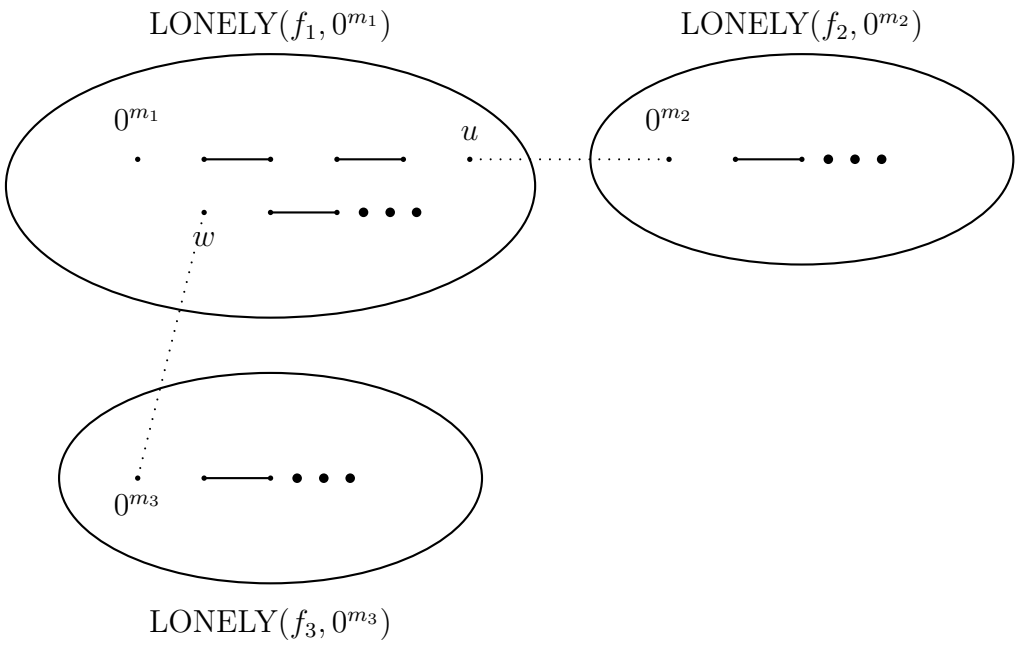

Figure 6.3: An illustration of the construction of one LONELY instance from all the LONELY instances in $T_{M,n}$. The dotted lines indicate added pairings.

LONELY, and that $T_{M,n}$ is shown as in Figure 6.1, where Q_2 is now LONELY. Let the instances of LONELY be as in Figure 6.3. For simplicity, an edge in the instance of LONELY($f_1, 0^{m_1}$) is shown only if $f_1(f_1(v)) = v$ and $f_1(v) \neq u$. Otherwise, v is a solution to LONELY($f_1, 0^{m_1}$) and no incident edges are shown. There are two solutions to LONELY($f_1, 0^{m_1}$) shown in Figure 6.3, they are u and w. Since the solution u leads M to the LONELY($f_2, 0^{m_2}$) query, pairing u and 0^{m_2} eliminates both of them as a solutions. Similarly, w is linked with 0^{m_3}. Then as before, if F is the function constructed by linking the instances of LONELY in $T_{M,n}$, the only unpaired points (other than 0^{m_1}) are in instances of LONELY that correspond to leaves in $T_{M,n}$. Again as before, since the leaves of $T_{M,n}$ encode solutions to Q_1, one call to LONELY on F solves Q_1. This process is carried out for the problems OntoPIGEON and LeftPIGEON in a similar fashion.

This chapter contains material from the paper "Propositional proofs and reductions between NP search problems" which is currently accepted for publication by the Annals of Pure and Applied Logic. This paper is co-authored by the dissertation author and Samuel R. Buss.

Chapter 7

Separations Between Many-one and Turing Reductions

7.1 Overview

While Chapter 6 showed that the classes PPA, PPAD, PPADS, and PLS are closed under Turing reductions, Theorem 7.3.1 of this chapter will prove there are TFNP^2 problems Q_1, Q_2 such that $Q_1 \leq_{\mathrm{T}} Q_2$ but $Q_1 \not\leq_{\mathrm{m}} Q_2$. The difference between a many-one and Turing reduction is that in the many-one case the reduction can only query Q_2 once, whereas a Turing reduction can make polynomially many queries to Q_2. Thus the separation between many-one and Turing reductions says that polynomially many calls to Q_2 are stronger than one. Theorem 7.3.1 actually shows the stronger result that using two calls to Q_2 is stronger than one. A even stronger separation is shown in Theorem 7.6.1, which shows that $k+1$ calls is stronger that k calls for all $k \geq 0$.

It is worth noting that the relation between many-one and Turing reductions has been previously studied by Hanika in [20], where problems A and B are constructed such that A is many-one reducible to B, but if A is Turing reducible to B then P = NP. A nice feature of this result is that A and B are unrelativized, which differs from the relativized separation of many-one and Turing reductions proved in Section 7. However, this result is not applicable to this chapter since A

and B are not NP search problems; solutions to A and B are not polynomial time verifiable (unless P = NP).

There are several tools used in the proof of Theorem 7.6.1, including the introduction of new TFNP^2 problems called MOD^d for $d > 1$. The problem MOD^d expresses the fact that a set of size $N \not\equiv 0 \bmod d$ cannot be partitioned into sets of size d. This is similar to previously studied principles such as the propositional formulas *Count* of [3]. The importance of MOD^d is the fact that it is a TFNP^2 problem, whereas the *Count* principles are not. However, [3] characterizes the provability of *Count* principles from each other. In order to apply this results to the MOD^d principles, Section 7.4 shows that the propositional translation of MOD^d is equivalent via a constant depth, polynomial size proof to an appropriate instance of *Count*. Thus the proof of Theorem 7.6.1 can use both the TFNP^2 aspect of the MOD^d problems and the provability results involving *Count*.

The proof of Theorem 7.6.1 combines different MOD^d principles through the operators $\otimes$, &, and $⅋$. These operators take TFNP^2 problems and create a new TFNP^2 problem and are defined in detail in Section 7.2. The proof of Theorem 7.6.1 also uses a new kind of reduction called an *infinitely-often* Turing reduction. Infinitely-often Turing reductions have to be correct only on infinitely many size parameters n, and are a type of non-uniform reduction. Despite this non-uniformity, Section 7.5 explains how the propositional translation of Theorem 3.3.1 extends to infinitely-often Turing reductions.

The structure of this chapter is as follows. Section 7.2 defines k-reducibility and the $\otimes$, &, and $⅋$ operators. Section 7.3 proves Theorem 7.3.1, one of the two main results of this chapter, which separates Turing and many-one reducibility. Section 7.4 defines the modular counting principles MOD^d and shows they are equivalent to the *Count* principles of [3]. Section 7.5 defines infinitely-often Turing reducibility. Lastly, Section 7.6 uses the $\otimes$, &, $⅋$ operators, the MOD^d principles, and infinitely-often Turing reductions to prove Theorem 7.6.1, the other main result of this chapter, which separates k- and $(k+1)$-reducibility.

7.2 k-reducibility and the $\otimes$, $\&$, and $\parr$ Operators

The following definition is used throughout this chapter.

Definition 7.2.1. *Let Q_1, Q_2 be* TFNP^2 *search problems. A Turing reduction M is a* k-reduction *from Q_1 to Q_2 if M is a Turing reduction from Q_1 to Q_2 such that on all inputs M makes at most k queries to Q_2. In this case, Q_1 is said to be* k-reducible *to Q_2, and is denoted $Q_1 \leq_k Q_2$.*

Clearly, 1-reducibility and many-one reducibility are equivalent, and a k-reduction is, in particular, a Turing reduction. Theorem 7.3.1 separates many-one and Turing reducibility by showing the stronger fact that 2-reducibility does not imply 1-reducibility. Theorem 7.6.1 generalizes this result by separating k- and $(k+1)$-reducibility.

The main tools used in the proofs of Theorems 7.3.1 and 7.6.1 are the operators $\otimes$, $\&$, and $\parr$ on TFNP^2 problems. The precise definitions are below, but the intuition is as follows. Fix TFNP^2 search problems Q_1 and Q_2. Then $Q_1 \otimes Q_2$ is the problem of simultaneously solving both an instance of Q_1 and an instance Q_2. A solution to the problem $Q_1 \& Q_2$ is a solution to a user specified choice of one of Q_1 or Q_2. That is, there is an extra input which determines which of Q_1 or Q_2 to solve. Finally, $Q_1 \parr Q_2$ is the problem of solving an unspecified choice of Q_1 or Q_2. In particular, there is no additional input to $Q_1 \parr Q_2$, the user cannot guarantee whether Q_1 or Q_2 is solved. The symbols $\otimes$, $\&$, and $\parr$ are motivated by linear logic. $\Gamma \longrightarrow A \otimes B$ means Γ has enough resources to solve both A and B; $\Gamma \longrightarrow A \& B$ means Γ has enough resources to solve a choice of A or B; $\Gamma \longrightarrow A \parr B$ means Γ has enough resources to solve A or B without being able to choose which one.

Definition 7.2.2. *Fix an integer $k \geq 1$. Suppose $Q_i(\vec{f_i}, \vec{x}_i) \in$* TFNP^2 *for $i = 1, \ldots, k$.*

Let $(\bigotimes_{i=1}^{k} Q_i)(\vec{f_1}, \ldots, \vec{f_k}, \vec{x}_1, \ldots, \vec{x}_k)$ be the NP *search problem "Given $\{\vec{x}_i\}_{i=1}^{k}$, where each $\vec{x}_i \in U_{n_i}$, and given $\{\vec{f_i}\}_{i=1}^{k}$, where each member of $\vec{f_i}$ has its inputs and output in U_{n_i}, find $\vec{y}_i \in Q_i(\vec{f_i}, \vec{x}_i)$ for each $i = 1, \ldots, k$."*

Let $(\&_{i=1}^{k} Q_i)(\vec{f_1}, \ldots, \vec{f_k}, \vec{x}_1, \ldots, \vec{x}_k, y)$ be the NP *search problem "Given $\{\vec{x}_i\}_{i=1}^{k}$, where each $\vec{x}_i \in U_{n_i}$, and given $\{\vec{f_i}\}_{i=1}^{k}$, where each member of $\vec{f_i}$ has*

its inputs and output in U_{n_i}, and given $y \in U_k$, find $y_i \in Q_i(\vec{f_i}, \vec{x_i})$ if $y = 0^{i-1}10^{k-i}$ and return 0 otherwise."

Let $(\bigtriangledown_{i=1}^{k} Q_i)(\vec{f_1}, \ldots, \vec{f_k}, \vec{x}_1, \ldots, \vec{x}_k)$ be the NP *search problem "Given $\{\vec{x}_i\}_{i=1}^{k}$, where each $\vec{x}_i \in U_{n_i}$, and given $\{\vec{f_i}\}_{i=1}^{k}$, where each member of $\vec{f_i}$ has its inputs and output in U_{n_i}, find $y_i \in Q_i(\vec{f_i}, \vec{x_i})$ for some i."*

Note that the "specified choice" for & mentioned above arises in the definition as an additional input string y which determines which problem to solve. Since the additional input string in absent in $\bigtriangledown$, the choice of which problem to solve for $\bigtriangledown$ is unspecified. The definitions are stated with the most general conditions, so that conventions of Chapter 3 do not apply. Namely, each Q_i is allowed to have multiple function inputs, the string inputs are not allowed restricted to be of the form 0^n, and the string inputs are allowed to be of different lengths. Even though the definitions are stated in the most general form, when used in the proofs below each Q_i has one input function and the size parameters n_i are are all equal to one common size paramter n.

7.3 A Separation Between $\leq_{\mathrm{m}}$ and $\leq_{\mathrm{T}}$

The following theorem is the first main result of this chapter.

Theorem 7.3.1. *There exist* TFNP^2 *search problems Q_1, Q_2 such that $Q_1 \leq_{\mathrm{T}} Q_2$, but $Q_1 \not\leq_{\mathrm{m}} Q_2$.*

As mentioned above, the proof will show that, in fact, $Q_1 \leq_2 Q_2$ but $Q_1 \not\leq_1 Q_2$.

Proof. Let Q_1 be LeftPIGEON $\otimes$ LONELY and let Q_2 be PIGEON&LONELY. We first show $Q_1 \leq_2 Q_2$. Note that it is shown in [2] that LeftPIGEON $\leq_{\mathrm{m}}$ PIGEON, let M' be this many-one reduction. We now describe a 2-reduction $M : Q_1 \leq_2 Q_2$. First, M queries Q_2 specifying that PIGEON be solved, where the PIGEON instance for Q_2 is the one used by M' (the LONELY instance can be arbitrarily chosen since we chose to solve PIGEON instead). When M receives an answer, it solves LeftPIGEON by simulating M'. Next, M queries Q_2 specifying that

LONELY be solved, where the LONELY instance identical to the one in Q_1 (now the PIGEON instance for Q_2 can be arbitrarily chosen). The answer to this query solves the LONELY instance. Thus M has found solutions to LeftPIGEON and LONELY, and hence has solved Q_1.

The separations in [2] are the motivation for the choice of LeftPIGEON, PIGEON, and LONELY in Q_1 and Q_2; namely, LONELY $\not\leq_{\mathrm{T}}$ PIGEON and LeftPIGEON $\not\leq_{\mathrm{T}}$ LONELY. The intuition is a many-one reduction to Q_2 chooses to solve exactly one of either LONELY or PIGEON. If LONELY is chosen, then LeftPIGEON cannot be solved, and if PIGEON is chosen, then LONELY cannot be solved. While the intuition served as the motivation for looking at these particular problems, the proof depends on results involving decision trees in [2]. The remainder of the proof makes this argument precise.

Let f, g be the function inputs to LeftPIGEON and h be the function input to LONELY. Without loss of generality, assume the size parameter for both problems is n, fixed sufficiently large. Suppose M is a many-one reduction from Q_1 to Q_2. It is enough to show how to specify f, g, h at step i of M's computation such that either a polynomial part of f, g have been specified and the known values for f, g do not contain a solution to LeftPIGEON, or a polynomial part of h has been specified and the known values for h do 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 polynomially many known values for the underlying graph. Therefore, since M returns an answer involving the unspecified part of the input, the graphs of f, g, h can be suitably altered so that M's answer is wrong. Thus M is not a correct reduction, which is a contradiction.

The rest of the proof shows how to specify f, g, h at any step i. It is clear how to do this if the i^{th} step is a query to f, g or h, since M can only make polynomially many queries and there is an exponential search space. So suppose the i^{th} step is a query to $Q_2(F, G, 0^m, 0^m, y)$, where $y \in U_2$ (the size parameters can be made equal without loss of generality). Since M is a many-one reduction, M has not made a previous Q_2 query and cannot make another Q_2 query. If y is not 01 or 10, then the Q_2 query is answered with 0, without needing to set any more of f, g, or h.

If y is the string 01, then the Q_2 query must solve LONELY on G. The function G is computed by a polynomial time Turing machine that has oracle access to f, g, and h. This gives rise to a decision tree T_u for $G(u)$ with internal nodes corresponding to f, g, or h queries; see Section 3 for the details on constructing decision trees. Since it is enough to solve the LONELY instance on G without revealing a solution to LeftPIGEON, arbitrarily fix the entire graph of h, and shorten each decision tree T_u to only include f and g queries. Theorem 6 of [2] shows that there is a way to specify a polynomial part of f and g that solves LONELY$(G, 0^m)$ without solving LeftPIGEON$(f, g, 0^n)$. This fact is proved by contradiction. First, the problem is turned into a combinatorial problem, which in turn is turned into a Nullstellensatz proof of a certain degree d. Then a lower bound on the degree is proved which is greater than d, which is the contradiction.

Note that fixing the graph of h could let M find a solution to LONELY on h. However, f, g have been set so that they do not solve LeftPIGEON. Since M can only query Q_2 once, and only queries to Q_2 can produce a solution to LeftPIGEON, if y is 01 then M never finds a solution to LeftPIGEON on f, g.

Now suppose y is the string 10. Then the query $Q_2(F, G, 0^m, 0^m, y)$ must solve PIGEON on F. It is enough to solve PIGEON on F without producing a solution to LONELY. Let T'_u be the decision tree for $F(u)$. Arbitrarily fix the graphs of f and g and shorten each decision tree T'_u to only include h queries. Lemma 4 of [2] shows how to specify a polynomial portion of h to correctly answer PIGEON$(F, 0^m)$ without solving LONELY$(h, 0^n)$; this result is generalized in Theorem 8.2.5. The proofs of Lemma 4 and Theorem 8.2.5 both rely on the the fact that if a Boolean function and its negation can be written in disjunctive normal form with terms of size at most d, then the function can be represented as a decision tree of height at most d^2. The proof finishes by showing if there is no way to set h to solve PIGEON without solving LONELY, then there is a bijection between two sets of branches with different cardinalities. The proof of Theorem 8.2.5 contains the details of this argument.

It is possible that M could find a solution to LeftPIGEON when the graphs of f, g are fixed. But now h is set so as to not solve LONELY, and M can make no

more Q_2 calls. Since Q_2 calls are the only way to solve LONELY, M will never find a solution to LONELY on h when y is 10. □

7.4 The MOD^d Principles

The MOD^d principles are used in the proof of Theorem 7.6.1. They are similar to many other modular counting principles, such as the *Count* principles of [3] (see Definition 7.4.2 below). The MOD^d principles are important for the results of this chapter because they are formulated as TFNP^2 problems, whereas the *Count* are not even NP search problems. Modular counting has already been encountered in the form of LONELY which is like counting mod 2; see Section 2.5 for details.

Definition 7.4.1. *Let d be an integer with $d > 1$. If d is not a power of 2,* MOD^d *is Q_Φ, where Φ is the formula*

$$\exists x[\bigvee_{\substack{s|d \\ s\neq d}} f^{(s)}(x) = x \vee f^{(d)}(x) \neq x].$$

If $d = 2^k$, MOD^d *is Q_Φ, where Φ is the prenex form of the formula*

$$f(0) = 0 \rightarrow \exists x[\bigvee_{\substack{s|d \\ s\neq d}} (x \neq 0 \wedge f^{(s)}(x) = x) \vee f^{(d)}(x) \neq x].$$

The notation $s|d$ indicates that s divides d.

It is clear that solutions to MOD^d are verifiable in polynomial time, and hence MOD^d is an NP search problem for all d. The problem MOD^2 appears in [21] as the $Count_2(f)$ axiom.

The intuition behind the MOD^d principle is that since $|U_n| = 2^n$ it is impossible for the orbits of f to partition U_n into sets of size d. First consider the case when d is not a power of 2. To see that MOD^d is total, suppose for all $u \in U_n$ that $f^{(s)}(u) \neq u$, for all $s|d$ such that $s \neq d$, and that $f^{(d)}(u) = u$. Fix u and let m be the order of the orbit of u under f. The first condition implies it is not the case that $m|d$ and $m \neq d$, and and the second condition implies m divides d; hence

$m = d$, so all orbits have size d. It can be checked that the orbits are disjoint, which is a contradiction since 2^n is not a multiple of d.

The above argument fails when d is a power of 2 because $|U_n| = 2^n$. The definition when d is a power of 2 makes the size of the set being partitioned odd by excluding 0. Specifically, suppose $f(0) = 0$, $f^{(d)}(u) = u$ for all $u \in U_n$, and $f^{(s)}(u) \neq u$ for all $s|d$ such that $s \neq d$ and all $u \neq 0 \in U_n$. Following the argument of the previous paragraph, the orbit of any $u \neq 0$ under f has order d, and these orbits are disjoint. Thus the non-zero elements of U_n can be partitioned in orbits of even size, which is clearly a contradiction since there are $2^n - 1$ many non-zero elements of U_n. Therefore MOD^d is also total when d is a power of 2.

Recall that the propositional LK translation of Chapter 3 associates to a total $\exists$-sentence Φ a family of sequents $F_{\Phi,n} \longrightarrow G_{\Phi,n}$. Here $F_{\Phi,n}$ encodes that the function input to the search problem Q_Φ is total and functional on U_n, and $G_{\Phi,n}$ encodes that there is a solution to Q_Φ on U_n. For convenience, the sets of formulas $F_{\mathrm{MOD}^d,n}$ and $G_{\mathrm{MOD}^d,n}$ are stated here in the case when d is not a power of 2. $Tot_{\mathrm{MOD}^d,n}$ is the set of formulas

$$\left\{ \bigvee_{v \in U_n} x_{u,v} : u \in U_n \right\},$$

$Func_{\mathrm{MOD}^d,n}$ is the set of formulas

$$\{\overline{x}_{u,v} \vee \overline{x}_{u,v'} : u, v, v' \in U_n, v \neq v'\},$$

and $F_{\mathrm{MOD}^d,n}$ is the cedent $\bigwedge Tot_{\mathrm{MOD}^d,n}, \bigwedge Func_{\mathrm{MOD}^d,n}$. The set $Soln_{\mathrm{MOD}^d,n}$ is the union of

$$\left\{ \bigwedge_{i=0}^{d-1} x_{u_i,u_{i+1}} : u_0, \ldots, u_d \in U_n, u_0 \neq u_d \right\} \tag{7.1}$$

and

$$\left\{ \bigwedge_{i=0}^{s-1} x_{u_i,u_{i+1}} : u_0, \ldots, u_s \in U_n, u_0 = u_s, s|d, s \neq d \right\}, \tag{7.2}$$

and $G_{\mathrm{MOD}^d,n}$ is the cedent $\bigvee Soln_{\mathrm{MOD}^d,n}$.

Note that the formulas in (7.1) encode the condition that $f^{(d)}(u) \neq u$ and the formulas in (7.2) encode the condition that $f^{(s)}(u) = u$.

The propositional formulas $Count_d^N$ defined in [3] express equivalent modular counting principles as the MOD^d principles (the equivalence is the subject of Lemmas 7.4.3 and 7.4.4).

Definition 7.4.2 ([3])**.** *Let V be a set of size N. Let $[V]^d$ be the subsets of V of size d. Then $Count_d^N$ is the sequent*

$$\rightarrow \bigvee_{v \in V} \bigwedge_{\substack{e \in [V]^d \\ e \ni v}} \overline{x}_e \vee \bigvee_{\substack{e,f \in [V]^d \\ e \perp f}} x_e \wedge x_f,$$

where $e \perp f$ means that $e \neq f$ and $e \cap f \neq \emptyset$. Let Φ be $\bigvee_v \bigwedge_{e \ni v} \overline{x}_e$ and let Ψ be $\bigvee_{e \perp f} x_e \wedge x_f$ so that $Count_d^N$ can be written as the sequent $\longrightarrow \Phi \vee \Psi$.

The variable x_e expresses the membership of the set e to the partition of V. Thus $Count_d^N$ expresses the statement that any sets $\{e_i\}$ that attempt to partition V either do not contain all elements of V or two different sets e_i, e_j intersect nontrivially.

An important fact is that the $Count_d^N$ principles are not NP search problems. In fact, it is not possible to recognize in polynomial time if some $v \in V$ is not contained in any set of the partition since there are exponentially many sets. Another way to see this is to note that the $\bigvee$'s in *Count* correspond to existential quantifiers and the $\bigwedge$ corresponds to a universal quantifier, so that *Count* is equivalent to a Σ_2-sentence.

The fact that the $Count_d^N$ principles are not NP search problems is the reason to introduce the MOD^d principles. However, the $Count_d^N$ principles have been studied in detail. In fact, [3] characterizes when there are constant depth, polynomial size proofs of $Count_d^N$ from $Count_{d'}^{N'}$; in particular, no such proofs exist when d and d' are relatively prime. We would like to use both MOD^d and $Count_d^N$; the former to define TFNP^2 problems, and the latter to prove separations between the TFNP^2 problems.

The crucial fact is that, even though the $Count_d^N$ principles are not NP search problems, the propositional formula $Count_d^N$ is equivalent via constant depth, polynomial size proofs to the propositional translation of MOD^d. This equivalence is the subject of the following two lemmas.

Lemma 7.4.3. *If d is not a power of 2, then for any n there are constant depth, polynomial (in 2^n) size* LK *proofs of $F_{\mathrm{MOD}^d,n} \longrightarrow G_{\mathrm{MOD}^d,n}$ from instances of $Count_d^{2^n}$.*

If r is a power of 2, then for any n there are constant depth, polynomial (in 2^n) size LK *proofs of $F_{\mathrm{MOD}^d,n} \longrightarrow G_{\mathrm{MOD}^d,n}$ from instances of $Count_d^{2^n-1}$.*

Proof. Recall that the MOD^d principles are defined by cases according to whether d is a power of 2 or not. The proof only shows the case when d is not a power of 2. The case when d is a power of 2 is similar, with only minor differences arising since the zero string is effectively eliminated from consideration. The fact that the zero string is removed accounts for the superscript $2^n - 1$ in the case when d is not a power of 2.

Let the set V in Definition 7.4.2 be U_n, so that $|V| = 2^n$. Let Φ and Ψ be as in Definition 7.4.2. The proof shows how to derive $F_{\mathrm{MOD}^d,n}, (\Phi)\sigma \longrightarrow G_{\mathrm{MOD}^d,n}$ and $F_{\mathrm{MOD}^d,n}, (\Psi)\sigma \longrightarrow G_{\mathrm{MOD}^d,n}$, for a substitution σ defined below. The lemma then follows by an $\vee$:left inference and a cut.

To define the substitution σ, begin by fixing an $e = \{e_0 < \ldots < e_{d-1}\} \in [U_n]^d$. Let S_d be the set of permutations on $\{0, \ldots, d-1\}$. For $\pi \in S_d$, let $x_{\pi,e}$ be the conjunction $\bigwedge_{i=0}^{d-2} x_{e_{\pi(i)},e_{\pi(i+1)}} \wedge x_{e_{\pi(d-1)},e_{\pi(0)}}$, and let $\overline{x}_{\pi,e}$ denote the negation of this conjunction, $\bigvee_{i=0}^{d-2} \overline{x}_{e_{\pi(i)},e_{\pi(i+1)}} \vee \overline{x}_{e_{\pi(d-1)},e_{\pi(0)}}$. Let $x_e\sigma$ be $\bigvee_{\pi \in S_d} x_{\pi,e}$.

We first prove the case $F_{\mathrm{MOD}^d,n}, (\Psi)\sigma \longrightarrow G_{\mathrm{MOD}^d,n}$. Pick any $e = \{e_0, \ldots, e_{d-1}\}$ and $f = \{f_0, \ldots, f_{d-1}\}$ such that $e \perp f$. Let $\pi, \varphi \in S_d$. Since $e \perp f$, there is a u such that $u = e_{\pi(i)} = f_{\varphi(j)}$ but $e_{\pi((i+1) \bmod d)} \neq f_{\varphi((j+1) \bmod d)}$, for some $0 \leq i, j \leq d-1$. It is clear there is a short proof of

$$\overline{x}_{u,e_{\pi((i+1) \bmod d)}} \vee \overline{x}_{u,f_{\varphi((j+1) \bmod d)}}, x_{\pi,e}, x_{\varphi,f} \longrightarrow.$$

By weakening and $\wedge$:left introduction derive

$$\bigwedge \mathit{Func}_{\mathrm{MOD}^d,n}, x_{\pi,e}, x_{\varphi,f} \longrightarrow.$$

By using $\vee$: left in a balanced fashion over all π and then over all φ derive

$$\bigwedge \mathit{Func}_{\mathrm{MOD}^d,n}, \bigvee_{\pi \in S_d} x_{\pi,e}, \bigvee_{\varphi \in S_d} x_{\varphi,f} \longrightarrow.$$

Then derive

$$\bigwedge Func_{\mathrm{MOD}^d,n}, \bigvee_{e \perp f} \left(\bigvee_{\pi \in S_d} x_{\pi,e} \wedge \bigvee_{\varphi \in S_d} x_{\varphi,f} \right) \rightarrow$$

by using an $\wedge$:left followed by $\vee$:lefts in a balanced fashion over all $e \perp f$. The desired sequent is derived by weakening and an $\wedge$:inferences.

We now prove the case $F_{\mathrm{MOD}^d,n}, (\Phi)\sigma \rightarrow G_{\mathrm{MOD}^d,n}$. To derive this sequent, it is enough to derive, for each $v \in U_n$,

$$F_{\mathrm{MOD}^d,n}, \bigwedge_{e \ni v} \bigwedge_{\pi \in S_d} \overline{x}_{\pi,e} \rightarrow G_{\mathrm{MOD}^d,n}, \tag{7.3}$$

since $(\Phi)\sigma$ is $\bigvee_{v \in U_n} \bigwedge_{e \ni v} \bigwedge_{\pi \in S_d} \overline{x}_{\pi,e}$. For the rest of the proof, let $x_{i,j}$ be short for x_{f_i,f_j}. For any $0 \leq i < d$ and any $f_0, \ldots, f_i \in U_n$ such that $v \in \{f_0, \ldots, f_i\}$, denote the sequent

$$\{x_{j,j+1}\}_{j=0}^{i-1}, \bigwedge Tot_{\mathrm{MOD}^d,n}, \bigwedge_{e \ni v} \bigwedge_{\pi \in S_d} \overline{x}_{\pi,e} \rightarrow G_{\mathrm{MOD}^d,n}$$

by $\Sigma(f_0, \ldots, f_i)$. To derive (7.3) it is enough to derive $\Sigma(f_0, \ldots, f_i)$ for any $0 \leq i < d$, since $\Sigma(f_0)$ is easily weakens to (7.3).

Derive $\Sigma(f_0, \ldots, f_i)$ by downward induction on i, with base case $i = d-1$. We first show the induction step. Let $i < d-1$ and arbitrarily choose $f_0, \ldots, f_{i-1} \in U_n$ such that $v \in \{f_0, \ldots, f_{i-1}\}$. For each choice of $f_i \in U_n$ the induction hypothesis gives a derivation of $\Sigma(f_0, \ldots, f_{i-1}, f_i)$. By $\vee$:left derive

$$\{x_{j,j+1}\}_{j=0}^{i-2}, \bigvee_{f_i \in U_n} x_{i-1,i}, \bigwedge Tot_{\mathrm{MOD}^d,n}, \bigwedge_{e \ni v} \bigwedge_{\pi \in S_d} \overline{x}_{\pi,e} \rightarrow G_{\mathrm{MOD}^d,n}.$$

Use weakening and $\wedge$:left to transform $\bigvee_{f_i \in U_n} x_{i-1,i}$ into $\bigwedge Tot_{\mathrm{MOD}^d,n}$ and contract on the left to derive $\Sigma(f_0, \ldots, f_{i-1})$.

Now consider deriving the base case $\Sigma(f_0, \ldots, f_{d-1})$. Choose $f_0, \ldots, f_{d-1} \in U_n$ such that $v \in \{f_0, \ldots, f_{d-1}\}$. First assume that $f_0, \ldots, f_{d-1}$ are all distinct, so that $f = \{f_0, \ldots, f_{d-1}\} \in [U_n]^d$. Since the definition of σ requires that the f_i's be in increasing order, pick the $\varphi \in S_d$ such that $f = \{f_{\varphi(0)} < \cdots < f_{\varphi(d-1)}\}$. In particular, for any $\pi \in S_d$, $x_{\pi,f}$ is

$$\bigwedge_{i=0}^{d-2} x_{\pi(\varphi(i)),\pi(\varphi(i+1))} \wedge x_{\pi(\varphi(d-1)),\pi(\varphi(0))},$$

and thus $x_{\varphi^{-1},f}$ is

$$\bigwedge_{i=0}^{d-2} x_{i,i+1} \wedge x_{d-1,0}.$$

Arbitrarily pick $w \in U_n$. If $w \neq f_0$, then it is clear there is a short proof of

$$x_{0,1}, \ldots, x_{d-2,d-1}, x_{d-1,w} \longrightarrow G_{\mathrm{MOD}^d,n}. \tag{7.4}$$

If $w = f_0$, then it is trivial to derive

$$x_{0,1}, \ldots, x_{d-2,d-1}, x_{d-1,w}, \overline{x}_{\varphi^{-1},f} \longrightarrow. \tag{7.5}$$

Weaken (7.4) and (7.5) and use $\vee$:introduction to derive

$$\{x_{j,j+1}\}_{j=0}^{d-2}, \bigvee_{w \in U_n} x_{d-1,w}, \overline{x}_{\varphi^{-1},f} \longrightarrow G_{\mathrm{MOD}^d,n}.$$

Apply weakening and $\wedge$:left to derive

$$\{x_{j,j+1}\}_{j=0}^{d-2}, \bigwedge Tot_{\mathrm{MOD}^d,n}, \bigwedge_{e \ni v} \bigwedge_{\pi \in S_d} \overline{x}_{\pi,e} \longrightarrow G_{\mathrm{MOD}^d,n},$$

which is the desired sequent.

Now assume that $f_0, \ldots, f_{d-1}$ are not distinct. Suppose $f_i = f_j$, for some $0 \leq i < j \leq d-1$. Without loss of generality, assume $f_i = f_j = f_0$. If $f_0 \to f_1 \to \cdots \to f_{j-1} \to f_0$ contains an s cycle, where $s|d$, then $s \neq d$ and it is easy to derive

$$\{x_{\ell,\ell+1}\}_{\ell=0}^{j-1} \longrightarrow G_{\mathrm{MOD}^d,n},$$

from which $\Sigma(f_0, \ldots, f_{d-1})$ follows by weakening.

Otherwise, assume $f_0 \to f_1 \to \cdots \to f_{j-1} \to f_0$ only contains cycles whose lengths do not divide d. Pick a cycle of shortest length from $f_0 \to f_1 \to \cdots \to f_{j-1} \to f_0$. This implies that starting on f_0 and iterating the cycle d times ends at an element other than f_0; that is, $f_{d \bmod j} \neq f_0$. Specifically, it is clear how to derive

$$\{x_{\ell,\ell+1}\}_{\ell=0}^{j-1} \longrightarrow \bigwedge_{\ell=0}^{d-2} x_{\ell \bmod j,(\ell+1) \bmod j} \wedge x_{(d-1) \bmod j,d \bmod j},$$

since each $x_{\ell,\ell+1}$ that appears on the left appears (perhaps multiple times) on the right. From this, it is easy to derive

$$\{x_{\ell,\ell+1}\}_{\ell=0}^{j-1} \longrightarrow G_{\mathrm{MOD}^d,n},$$

and $\Sigma(f_0 \ldots, f_{d-1})$ again follows by weakening. Thus $\Sigma(f_0, \ldots, f_{d-1})$ is derived for all cases. This finishes the base case, hence the induction, and hence the proof of the lemma.

It is easy to check the proofs are all polynomial size and constant depth. $\square$

Lemma 7.4.4. *If d is not a power of 2, then for any n there are constant depth, polynomial (in 2^n) size proofs of $Count_d^{2^n}$ from instances of $F_{\mathrm{MOD}^d,n} \longrightarrow G_{\mathrm{MOD}^d,n}$.*

If d is a power of 2, then for any n there are constant depth, polynomial (in 2^n) size proofs of $Count_d^{2^n-1}$ from instances of $F_{\mathrm{MOD}^d,n} \longrightarrow G_{\mathrm{MOD}^d,n}$.

Proof. As before the lemma is only proved in the case when d is not a power of 2. Define the substitution to be applied to $F_{\mathrm{MOD}^d,n} \longrightarrow G_{\mathrm{MOD}^d,n}$ as follows: Suppose $e \in [U_n]^d$, where $e = \{e_0 < \ldots < e_{d-1}\}$. If $u = e_i$ for some $0 \leq i \leq d-1$ and $v = e_{i+1 \bmod d}$, then write $e(u) = v$. Let σ replace $x_{u,v}$ with

$$\bigvee_{\substack{e \in [U_n]^d: \\ e(u)=v}} x_e.$$

Frequently $x_{u,v}\sigma$ be written $\bigvee_{e(u)=v} x_e$ to improve readability. Let Φ and Ψ be as in Definition 7.4.2. If there are constant depth, polynomial size proofs of

(i) $\longrightarrow \Phi \vee \Psi, \bigwedge Tot_{\mathrm{MOD}^d,n}\sigma$,

(ii) $\longrightarrow \Phi \vee \Psi, \bigwedge Func_{\mathrm{MOD}^d,n}\sigma$, and

(iii) $G_{\mathrm{MOD}^d,n}\sigma \longrightarrow \Phi \vee \Psi$,

then the lemma follows by cuts.

First consider (i). It is clear that there is a short proof of

$$\bigwedge_{u \in U_n} \bigvee_{e \ni u} x_e \longrightarrow \bigwedge_{u \in U_n} \bigvee_{v \in U_n} \bigvee_{e(u)=v} x_e. \tag{7.6}$$

Note that $\neg \bigwedge_{u \in U_n} \bigvee_{e \ni u} x_e$ is equivalent to $\bigvee_{u \in U_n} \bigwedge_{e \ni u} \overline{x}_e$, and this latter formula is exactly Φ. Also recall that $\bigwedge Tot_{\mathrm{MOD}^d,n}$ is $\bigwedge_{u \in U_n} \bigvee_{v \in U_n} x_{u,v}$. Thus derive

$$\longrightarrow \Phi, \bigwedge Tot_{\mathrm{MOD}^d,n}\sigma$$

from (7.6). Weaken and use $\vee$:right to derive the desired sequent (i).

Now consider (ii). Fix $u, v, v' \in U_n$ such that $v \neq v'$. It is clear there is a short proof of

$$\bigvee_{e(u)=v} x_e \wedge \bigvee_{f(u)=v'} x_f \longrightarrow \bigvee_{e \perp f} x_e \wedge x_f.$$

Since $\neg \left(\bigvee_{e(u)=v} x_e \wedge \bigvee_{f(u)=v'} x_f \right)$ is equivalent to $(\overline{x}_{u,v} \vee \overline{x}_{u,v'})\sigma$, there is a proof of

$$\longrightarrow \Phi \vee \Psi, (\overline{x}_{u,v} \vee \overline{x}_{u,v'})\sigma.$$

Derive $\longrightarrow \Phi \vee \Psi, \bigwedge Func_{\mathrm{MOD}^d,n}\sigma$ by applying $\wedge$:lefts in a balanced fashion.

Finally consider (iii). First suppose $s|d$ with $s \neq d$. Fix $u_0, \ldots u_s$ where $u_0 = u_s$, so that $\bigwedge_{i=0}^{s-1} x_{u_i,u_{i+1}}\sigma$ is a member of the set in (7.2). This formula is equivalent to

$$\bigvee_{e_0(u_0)=u_1} \cdots \bigvee_{e_{s-1}(u_{s-1})=u_0} \bigwedge_{i=0}^{s-1} x_{e_i}. \tag{7.7}$$

Claim: Each disjunct of (7.7) contains $i \neq i'$ such that $e_i \perp e_{i'}$.

If the claim failed, then by the definition of $e(u) = v$, each e_i would contain only the s elements $u_0, \ldots, u_{s-1}$, which cannot happen since e_i is a subset of d distinct elements of U_n and $s \neq d$. Thus the claim holds.

Therefore, for any choice of $e_0, \ldots, e_{s-1}$ such that $e_i(u_i) = u_{i+1 \bmod s}$ for $0 \leq i \leq s-1$, there is a proof of

$$\bigwedge_{i=0}^{s-1} x_{e_i} \longrightarrow \Psi.$$

Thus there is a proof of (7.7)$\longrightarrow \Psi$, which by the previously mentioned equivalence gives a proof of

$$\bigwedge_{i=0}^{s-1} x_{u_i,u_{i+1}}\sigma \longrightarrow \Psi. \tag{7.8}$$

Now consider $\bigwedge_{i=0}^{d-1} x_{u_i,u_{i+1}}\sigma$, where $u_d \neq u_0$, which is a member of the set in (7.1). This formula is equivalent to

$$\bigvee_{e_0(u_0)=u_1} \cdots \bigvee_{e_{d-1}(u_{d-1})=u_d} \bigwedge_{i=0}^{d-1} x_{e_i}. \tag{7.9}$$

Claim: Each disjunct of (7.9) contains $i \neq i'$ such that $e_i \perp e_{i'}$.

Suppose for a contradiction that this is not the case. Note that $u_0, \ldots, u_d$ cannot all be distinct, since then e_0 would contain $d+1$ distinct elements of U_n, which cannot happen. Thus $u_j = u_{j'}$ for some $0 \leq j < j' \leq d$. Note that it cannot be that $\{j, j'\} = \{0, d\}$ since $u_0 \neq u_d$, thus $|j - j'| < d$. But then there is a cycle $u_j \to u_{j+1} \to \cdots \to u_{j'-1} \to u_j$ of length strictly less than d. This cannot happen, otherwise e_j would contain strictly less than d elements. Thus the claim holds.

Similar to the previous argument, from the claim there is a proof of $(7.9) \longrightarrow \Psi$, and thus of

$$\bigwedge_{i=0}^{d-1} x_{u_i,u_{i+1}} \sigma \longrightarrow \Psi. \tag{7.10}$$

Applying $\vee$:lefts to the sequents in (7.8) and (7.10) yields a proof of $G_{\mathrm{MOD}^r,n}\sigma \longrightarrow \Psi$, from which $G_{\mathrm{MOD}^r,n}\sigma \longrightarrow \Phi \vee \Psi$ follows easily.

Again, the proofs are easily checked to be constant depth and polynomial size. □

The following lemma states a relation between the MOD^d principles and the $\mathfrak{Z}$ operator that will be needed in the proof of Theorem 7.6.1.

Lemma 7.4.5. *Let $p_1, \ldots, p_r$ be distinct primes, and let $d = \prod_{i=1}^r p_i$. Then $\mathrm{MOD}^d \equiv_{\mathrm{m}} \mathfrak{Z}_{i=1}^r \mathrm{MOD}^{p_i}$.*

Proof. We only show the proof in the case where $p_i \neq 2$ for $i = 1, \ldots, r$. Allowing a p_i to be 2 does not fundamentally change the proof, but the details must change to take account of the different definition of MOD^2 from other MOD^{p_i}'s.

Let S be $\mathfrak{Z}_{i=1}^r \mathrm{MOD}^{p_i}$. We describe a Turing machine M that solves S with a single call to an instance of MOD^d. Suppose the function inputs to S are $f_1, \ldots, f_r$, and without loss of generality assume each f_i is a function on the universe U_n. Then let $F(u_1, \ldots, u_r) = (f_1(u_1), \ldots, f_r(u_r))$. Since F is supposed to be an input to MOD^d, it must be a mapping on U_m for some m, so treat the vectors $(u_1, \ldots, u_r)$ and $(f_1(u_1), \ldots, f_r(u_r))$ as $m = rn$ bit strings by concatenation. The machine M simply sets up F, queries $\mathrm{MOD}^d(F, 0^m)$, and receives an answer. The following

shows that no matter what answer is received, the reduction can compute a solution to S in polynomial time relative to the f_i's.

Suppose M receives $u = (u_1, \ldots, u_r)$ as a solution to the query $\mathrm{MOD}^d(F, 0^m)$. Suppose for some $s|d$, $s \neq d$, and $F^{(s)}(\vec{u}) = \vec{u}$. Then there is a p_i such that p_i and s are relatively prime. If $f_i^{(p_i)}(u_i) \neq u_i$, then M returns u_i as a solution to S. Otherwise $f_i^{(p_i)}(u_i) = u_i$, and the order of u_i divides p_i. But $F^{(s)}(\vec{u}) = \vec{u}$ implies that $f_i^{(s)}(u_i) = u_i$, and hence the order of u_i divides s. Since p_i and s are relatively prime, the order of u_i is 1, so $f_i(u_i) = u_i$. Once again M returns u_i. In either case, M has solved MOD^{p_i}, and hence has solved S.

Now suppose instead that $F^{(d)}(\vec{u}) \neq \vec{u}$. If $f_i^{(p_i)}(u_i) = u_i$ for all $1 \leq i \leq r$, then $F^{(d)}(\vec{u}) = \vec{u}$, a contradiction. Thus M can try each of the constantly many i to find a u_i such that $f_i^{(p_i)}(u_i) \neq u_i$. Again this solves MOD^{p_i} and hence S.

We now show there is a many-one reduction M from MOD^d to S. Let f be the input function to MOD^d on the universe U_n. Let $f_i(u) = f^{(s_i)}(u)$, where $s_i = d/p_i$, for $i = 1, \ldots, r$. Then M queries S with the f_i's and gets an answer u. The following shows that u in fact solves $\mathrm{MOD}^d(f, 0^n)$.

By definition of S, u solves some $\mathrm{MOD}^{p_i}(f_i, 0^n)$ (exactly which i can be found by checking the r different possibilities). Suppose $f_i(u) = u$. Then $f^{(s_i)}(u) = u$, in which case u solves $\mathrm{MOD}^d(f, 0^n)$. Now suppose $f_i^{(p_i)}(u) \neq u$. Then u solves $\mathrm{MOD}^d(f, 0^n)$, since $u \neq f_i^{(p_i)}(u) = f^{(s_i p_i)}(u) = f^{(d)}(u)$. □

7.5 Infinitely-often Turing Reductions

A new kind of reducibility that will be needed below in the proof of Theorem 7.6.1 is infinitely-often Turing reducibility.

Definition 7.5.1. *Let $Q_1(\vec{f}, \vec{x}), Q_2(\vec{g}, \vec{y})$ be type-2* TFNP *problems. A sequence $(M_n)_n$ of type-2 oracle Turing machines is an* infinitely-often Turing reduction *from Q_1 to Q_2 if there exists d and an infinite set A such that for any $\vec{x} \in U_n$ and $n \in A$, when M_n is run on input $(\vec{f}, \vec{x})$, M_n outputs $z \in Q_1(\vec{f}, \vec{x})$ in time n^d relative to the f_i's and Q_2. In this case, Q_1 is* infinitely-often Turing reducible *to Q_2, denoted $Q_1 \leq_{\mathrm{T}}^{\mathrm{i.o.}} Q_2$ or $(M_n)_{n \in A} : Q_1 \leq_{\mathrm{T}}^{\mathrm{i.o.}} Q_2$. If B is the set of (n, m) such that $n \in A$ and*

there is an input $(\vec{f}, \vec{x})$*, where* $\vec{x} \in U_n$*, on which* M_n *makes a query to* Q_2 *with size parameter* m*, then this can be denoted* $(M_n)_{n\in A}, B : Q_1 \leq_{\mathrm{T}}^{\mathrm{i.o.}} Q_2$*.*

If M *makes only one call to* Q_2*, then* M *is* infinitely-often many-one reduction *from* Q_1 *to* Q_2*, and any of the notations* $Q_1 \leq_{\mathrm{m}}^{\mathrm{i.o.}} Q_2$*,* $(M_n)_{n\in A} : Q_1 \leq_{\mathrm{m}}^{\mathrm{i.o.}} Q_2$*, or* $(M_n)_{n\in A}, B : Q_1 \leq_{\mathrm{m}}^{\mathrm{i.o.}} Q_2$ *will be used based on context.*

Instead of using Turing machines, infinitely-often Turing reductions could have been defined equivalently using polynomial size circuits.

An important fact about infinitely-often Turing reductions is that they allow for non-uniformity; each size parameter n has its own Turing machine M_n for inputs of size n. Another important note is that infinitely-often Turing reductions relax the conditions of a Turing reduction is by only requiring the reduction to be correct for infinitely many size parameters. This fact leads immediately to the following lemma.

Lemma 7.5.2. *If* $M : Q_1 \leq_{\mathrm{m}} Q_2$*, then for any infinite set of positive integers* A *there exists an infinitely-often Turing reduction* $(M_n)_{n\in A} : Q_1 \leq_{\mathrm{m}}^{\mathrm{i.o.}} Q_2$*.*

The lemma is stated with respect to many-one reducibility as it will be applied in that situation. However, as M_n is exactly M for each n, any property of the reduction M, such as being a Turing or k-reduction, is inherited by the infinitely-often Turing reduction $(M_n)_{n\in A}$.

Note that in the proof of Lemma 7.4.5 if n is the size parameter to MOD^d, then the reduction $\mathrm{MOD}^d \leq_{\mathrm{m}} \bigtimes_{i=1}^{r} \mathrm{MOD}^{p_i}$ queries $\bigtimes_{i=1}^{r} \mathrm{MOD}^{p_i}$ with the same size parameter n. The next corollary follows from this fact and Lemma 7.5.2.

Corollary 7.5.3. *Let* $p_1, \ldots, p_r$ *be distinct primes, and let* $d = \prod_{i=1}^{r} p_i$*. Then for any infinite set of positive integers* A *there is an infinitely-often Turing reduction* $(M_n)_{n\in A}, D : \mathrm{MOD}^d \leq_{\mathrm{m}}^{\mathrm{i.o.}} \bigtimes_{i=1}^{r} \mathrm{MOD}^{p_i}$*, where* $D = \{(n, n) : n \in A\}$*.*

Another important fact is that the translation of Chapter 3 still applies to infinitely-often Turing reductions. That is, Theorem 3.3.1 applies when the condition $Q_\Phi \leq_{\mathrm{T}} Q_\Psi$ is relaxed to $Q_\Phi \leq_{\mathrm{T}}^{\mathrm{i.o.}} Q_\Psi$. It is easy to check that the details of the proof carry over exactly to the infinitely-often Turing reduction case. The

main point is that the proof of $F_{\Phi,n} \longrightarrow G_{\Psi,n}$ for size parameter n is built only from $T_{M,n}$, and since there is a uniform polynomial bound on the runtime of M, and hence the height of $T_{M,n}$ is similarly bounded, the non-uniformity of M does not affect the argument. The resulting propositional LK proofs are non-uniform, but this fact never arises in the proofs below.

Transitivity for infinitely-often Turing reductions is not the same as for ordinary Turing reductions. For instance, suppose $(M_n)_{n\in A}, B : Q_1 \leq_{\mathrm{T}}^{\mathrm{i.o.}} Q_2$ and $(M'_n)_{n\in A'}, B' : Q_2 \leq_{\mathrm{T}}^{\mathrm{i.o.}} Q_3$. It is tempting to use the following argument to show $Q_1 \leq_{\mathrm{T}}^{\mathrm{i.o.}} Q_3$. Let M''_n on input $n \in A$ simulate M_n, except that whenever M_n makes a call to Q_2 with size parameter m, M''_n instead uses M'_m to make queries to Q_3 to answer the Q_2 query. This argument fails since m may not be in A', in which case M'_m may not return a solution to Q_2. The following lemma formulates the analogue of transitivity for infinitely-often Turing reductions.

Lemma 7.5.4. *Let $(M_n)_{n\in A}, B : Q_1 \leq_{\mathrm{T}}^{\mathrm{i.o.}} Q_2$, $(M'_n)_{n\in A'} : Q_2 \leq_{\mathrm{T}}^{\mathrm{i.o.}} Q_3$, and $B_2 = \{m|\exists n(n,m) \in B\}$. If $B_2 \subseteq A'$, then $Q_1 \leq_{\mathrm{T}}^{\mathrm{i.o.}} Q_3$. In fact, there are Turing machines $(M''_n)_{n\in A}$ such that $(M''_n)_{n\in A}, B'' : Q_1 \leq_{\mathrm{T}}^{\mathrm{i.o.}} Q_3$, where B'' is the set of (n,m') such that there is an input $(\vec{f},\vec{x})$, where $\vec{x} \in U_n$, on which M_n calls Q_2 with size parameter m, and there is an input $(\vec{g},\vec{y})$, where $\vec{y} \in U_m$, on which M'_m calls Q_3 with parameter m'.*

Proof. Note that enough assumptions have been added so that the argument of the previous paragraph is now correct. □

7.6 $\mathbf{A} \leq_k$ and $\leq_{k+1}$ Separation

This is the second main result of this chapter, after Theorem 7.3.1. Note that Theorem 7.6.1 is stronger than Theorem 7.3.1.

Theorem 7.6.1. *For each $k \geq 0$ there exist* TFNP *problems Q_1, Q_2 such that $Q_1 \leq_{k+1} Q_2$ but $Q_1 \not\leq_k Q_2$. In fact, for a sequence of distinct primes $p_1, p_2, \ldots$, this statement holds for $Q_1 = \bigotimes_{i=1}^{k+1} \mathrm{MOD}^{p_i}$ and $Q_2 = \&_{i=1}^{k+1} \mathrm{MOD}^{p_i}$.*

The remainder of the section proves Theorem 7.6.1. For convenience, assume each $p_i > 2$. This avoids notational difficulties arising from the difference in the definition of MOD^d when d is a power of 2. Also, since the case $k = 0$ is trivial, assume that $k \geq 1$. As an aside, note the case $k = 1$ gives a different example than Theorem 7.3.1 of two problems for which many-one and Turing reducibility are not equivalent.

Let $k \geq 1$, fix distinct primes $p_1, \ldots, p_{k+1} > 2$, let Q_1 be $\bigotimes_{i=1}^{k+1} \text{MOD}^{p_i}$, and let Q_2 be $\&_{i=1}^{k+1}\text{MOD}^{p_i}$. It is clear that $Q_1 \leq_{k+1} Q_2$. Specifically, suppose the function input to each MOD^{p_i} in Q_1 is $f_i : U_{n_i} \to U_{n_i}$. Let M be the Turing reduction which makes $k+1$ successive queries to Q_2. For the i^{th} query, M chooses to solve MOD^{p_i} on f_i. It is clear that, after the $k+1$ queries to Q_2, M has found a solution to each $\text{MOD}^{p_i}(f_i, 0^{n_i})$, and hence has found a solution to Q_1.

Now suppose for a contradiction that $M : Q_1 \leq_k Q_2$. Let the function input to MOD^{p_i} be f_i, and, without loss of generality, assume that the string input to each MOD^{p_i} is 0^n. Also assume that M has the following properties: (1) Queries by M of the form $f_i(u)$ are replaced by the p_i many queries $f_i(u), f_i^{(2)}(u), \ldots, f_i^{(p_i)}(u)$; (2) Each Q_2 query made by M is in fact a query to an individual MOD^{p_i}, for some $i \in \{1, \ldots, k+1\}$; (3) Whenever M receives an answer to a MOD^{p_i} query, M immediately verifies the correctness of the answer. (1) can be assumed without loss of generality since the number of queries increases by a factor of at most $\max_i p_i$, which is constant in n. (2) also can be assumed because before M makes a Q_2 query q it must write a string 0^{m_i}, the descriptions of a function F_i from U_{m_i} to U_{m_i} for each $i = 1, \ldots, k$, and the string y. If $y = 0^{i-1}10^{k-i}$ then the query q can be replaced by a query to $\text{MOD}^{p_i}(F_i, 0^{m_i})$ (if y is not of this form q can be skipped entirely). (3) can be assumed because the MOD^{p_i}'s are NP search problems and thus verifying answers adds only polynomial amount of work. Note that assumption (1) still applies to any queries that arise in the verification process of (3). For a given n, let $T_{M,n}$ be the decision tree for M on size input 0^n (refer to Section 3.4 for more details) with the following adaptation. Normally, the internal nodes of $T_{M,n}$ are labeled with queries to an f_i or Q_2. However, due to assumption (2) the internal nodes are labeled as queries to the f_i's or the MOD^{p_i}'s.

The following introduces terminology to be used in the proof. If μ is a node in $T_{M,n}$, then P_μ is the path in $T_{M,n}$ from the root to μ (P_μ contains μ). If μ corresponds to a MOD^{p_i} query, then μ is at *level* j if μ is the j^{th} query to any $\text{MOD}^{p_{i'}}$ on P_μ. A path P_μ *contains a solution to* $\text{MOD}^{p_i}(f_i, 0^n)$ if the graph f_i specified by the edge labels in P_μ contains an explicit $u \in U_n$ solving $\text{MOD}^{p_i}(f_i, 0^n)$. Now consider verifying the correctness of an answer $u \in U_m$ to a $\text{MOD}^{p_i}(F, 0^m)$ query. First note that F is computed by a Turing machine with oracle access to *all* the f_j's. Thus the decision trees for F can contain internal nodes labeled "$f_j(u) =?$" for $j \neq i$, and thus it is conceivable that verifying that u solves $\text{MOD}^{p_i}(F, 0^m)$ could reveal a solution to $\text{MOD}^{p_j}(f_j, 0^n)$ for $j \neq i$. We say that *the verification that* u *solves* $\text{MOD}^{p_i}(F, 0^m)$ *contains a solution to* $\text{MOD}^{p_j}(f_j, 0^n)$ if the portion of the graph of f_j discovered only while verifying the correctness of u contains a solution to $\text{MOD}^{p_j}(f_j, 0^n)$.

Let μ be at level j and correspond to a query $\text{MOD}^{p}(F, 0^m)$. Then let V_μ be the set of i such that there is an solution to $\text{MOD}^{p}(F, 0^m)$ whose verification contains a solution to $\text{MOD}^{p_i}(f_i, 0^n)$, or P_μ contains a solution to $\text{MOD}^{p_i}(f_i, 0^n)$. Let W_μ be the set of i such that P_μ contains a query to MOD^{p_i}. A node μ at level j is *regular* if $V_\mu \subseteq W_\mu$. Thus μ is regular if, for any answer to the query corresponding to μ, after M verifies the correctness of the response, M has only solved those $\text{MOD}^{p_i}(f_i, 0^n)$'s for which it has queried some $\text{MOD}^{p_i}(F, 0^m)$. A query q to a MOD^{p_i} is regular if the node μ corresponding to q in $T_{M,n}$ is regular. The tree $T_{M,n}$ is regular if it contains no irregular nodes.

Claim: $T_{M,n}$ is irregular for all sufficiently large n.

For a contradiction, assume there exists a large n such that $T_{M,n}$ is regular. Note that at any point in its computation M has revealed only a polynomial part of each of the f_i's. Then for queries $f_i(u)$ made in the normal execution of M, that is not while verifying the correctness of the answer to a MOD^{p_i} query, it is possible to set the values $f_i(u), f_i^{(2)}(u), \ldots, f_i^{(p_i)}(u)$ to be $p_i - 1$ new points previously unset by f_i such that $f_i^{(p_i)}(u) = u$. Thus such queries to f_i do not reveal any new solutions to $\text{MOD}^{p_i}(f_i, 0^n)$ (or any $\text{MOD}^{p_j}(f_j, 0^n)$ for that matter). On the other

hand, whenever M queries MOD^{p_i}, since the query is regular, it is possible to set a polynomial amount of all the f_j's so that M can answer MOD^{p_i} and verify the solution while only creating explicit solutions to the MOD^{p_j}'s it has queried so far. Thus the only time M finds a solution to $\mathrm{MOD}^{p_i}(f_i, 0^n)$ is during a MOD^{p_j} query, and each such query finds a solution to at most one MOD^{p_i}. Since M only makes k calls to the MOD^{p_i}'s, when M terminates it has solved at most k of the $\mathrm{MOD}^{p_i}(f_i, 0^n)$'s. Therefore, M must produce an answer u to some $\mathrm{MOD}^{p_i}(f_i, 0^n)$'s without explicitly knowing $f_i(u)$. The values of $f_i(u), \ldots, f_i^{(p_i)}(u)$ can be set so that u is not a solution, contradicting the correctness of M.

So the claim holds and $T_{M,n}$ is irregular for all sufficiently large n. Since there are infinitely many such n, and only constantly many levels (k in fact), there exists a level $j' \in \{1, \ldots, k\}$ such that there are infinitely many n where $T_{M,n}$ contains an irregular node μ at level j' such that μ is the first irregular node on P_μ. Since P_μ has only regular nodes before μ, before receiving the answer to query corresponding to μ, P_μ knows the answer to at most $j' - 1$ of the $\mathrm{MOD}^{p_i}(f_i, 0^n)$'s. By the irregularity of μ , after receiving the answer to the query corresponding to μ the machine M knows solutions to a $\mathrm{MOD}^{p_i}(f_i, 0^n)$ that has not been queried. Since there are constantly many subsets of $\{p_1, \ldots, p_{k+1}\}$ of size at most $j' - 1$, there exists a constant $j \in \{1, \ldots, k\}$ and distinct primes $q, q_1, \ldots, q_{j-1}$, and an infinite set A, such that when $n \in A$, $T_{M,n}$ has an irregular node μ at level j which corresponds to a MOD^q query, P_μ contains explicit answers to only $\mathrm{MOD}^{q_1}(f_1, 0^n), \ldots, \mathrm{MOD}^{q_{j-1}}(f_j, 0^n)$, and for any answer to μ, verifying the answer solves some $\mathrm{MOD}^{p_i}(f_i, 0^n)$, where $p_i \notin \{q, q_1, \ldots, q_{j-1}\}$. Without loss of generality assume $q_i = p_i$ for $i = 1, \ldots, j-1$ and $q = p_j$. Let S be $\bigvee_{i=j+1}^{k+1} \mathrm{MOD}^{p_i}$. Note that S is not trivial since $j \leq k$. Let $d = \prod_{i=j+1}^{k+1} p_i$. Then $S \equiv_{\mathrm{m}} \mathrm{MOD}^d$ by Lemma 7.4.5.

Let $n \in A$, and consider the irregular node μ in $T_{M,n}$ described above, so that the query corresponding to μ is to MOD^{p_j}. As stated above, M finds an answer to some $\mathrm{MOD}^{p_i}(f_i, 0^n)$ for $i > j$ while verifying the answer to the query corresponding to μ. The intuition is that it should be possible to build a many-one reduction $M' : S \leq_{\mathrm{m}} \mathrm{MOD}^{p_j}$ by simulating the query corresponding to μ and verifying the response.

One problem with building M' is that the irregular nodes μ appear in $T_{M,n}$ only when $n \in A$. Another problem is that on input 0^n, M' would need the information on P_μ in order to set up the query corresponding to μ. Since the μ's (and hence the P_μ's) are not explicitly constructed, M' would have to be non-uniform. Infinitely-often Turing reductions resolve both of these problems; see the discussion after Definition 7.5.1.

Claim: There is an infinitely-often Turing reduction $(M_n)_{n\in A} : S \leq_{\mathrm{m}}^{\mathrm{i.o.}} \mathrm{MOD}^{p_j}$.

Proof of Claim: The following constructs a non-uniform family of type-2 Turing machines M_n, for $n \in A$, such that, on inputs $\{g_i\}_{i=j+1}^{k+1}$ and size parameter n, M_n runs in polynomial time $p(n)$, makes one query to MOD^{p_j}, and solves S on $\{g_i\}_{i=j+1}^{k+1}$ and size parameter n.

Fix $n \in A$, let μ be the irregular node in $T_{M,n}$ described above, and let $\mathrm{MOD}^{p_j}(F, 0^m)$ be the query corresponding to μ. The idea of the reduction is as follows. There is a polynomial time Turing machine M_F with access to the f_i's which computes F. The irregularity of μ guarantees that the verification of any answer to the query contains a solution to some $\mathrm{MOD}^{p_i}(f_i, 0^n)$ such that M has not yet made any MOD^{p_i} query; that is, the verification of the answer solves S on the f_i's. The machine M_n needs to make a query $\mathrm{MOD}^{p_j}(G, 0^m)$ for some function G in order to solve S on the g_i's. The idea is that function G should be defined by simulating M_F and replacing f_i queries with g_i queries. This approach has does not quite work for two reasons. The first is that M_F uses f_i as an oracle for $1 \leq i \leq k+1$, whereas there are g_i's only for $i > j$. The second is that P_μ contains information about the f_i's, so the g_i's need to be modified in such a way that g_i agrees with f_i on the values of f_i contained in P_μ.

The functions h_i for $1 \leq i \leq k+1$ are defined below to resolve these two problems. Before doing that, note that since M has not solved MOD^{p_i} for $i > j$ by the time M reaches μ in $T_{M,n}$, the values for each f_i in P_μ contain only disjoint orbits of length p_i. That is, for each $i > j$, the partial function f_i is one-to-one, and if $f_i(u) = v$ is in P_μ, then $v \neq u$ and $f_i^{(2)}(u), \ldots, f_i^{(p_i)}(u)$ are also in P_μ and distinct with $f_i^{(p_i)}(u) = u$. Let f_i have ℓ_i distinct p_i cycles.

The following is a description of M_n. Let M_n begin by running the following subroutines, one for each $j+1 \leq i \leq k+1$. The i^{th} subroutine is an ℓ_i stage algorithm described as follows. Let V_r be the set of unknown values of g_i at step r. At step r, M_n queries $g_i(u), \ldots, g_i^{(p_i)}(u)$, where u is the minimal element of V_r. If a solution to $\text{MOD}^{p_i}(g_i, 0^n)$ is revealed, M_n outputs that solution and halts. Otherwise, the graph of g_i at the end of stage r consists of exactly r disjoint p_i cycles. If $r < \ell_i$, then M_n continues to step $r+1$, otherwise the algorithm ends.

After running these subroutines M_n has either solved S, or knows values of g_i that form exactly ℓ_i disjoint p_i cycles, for $i > j$. If M_n has not solved S, then there are bijections φ_i, for $i > j$, on U_n such that (1) φ_i is also a bijection between the values of f_i in P_μ and the known values of g_i and (2) φ_i preserves the cycle structure of f_i and g_i, i.e. $g_i(\varphi_i(u)) = \varphi_i(f_i(u))$ when $f_i(u)$ is in P_μ. For each $i > j$, M_n defines h_i to be $\varphi_i^{-1} \circ g_i \circ \varphi_i$. For each $j \leq i$, let

$$h_i(u) = \begin{cases} v & \text{if } f_i(u) = v \text{ is contained in } P_\mu \\ 0 & \text{otherwise} \end{cases}.$$

Therefore, $h_i(u) = f_i(u)$ when $f_i(u)$ is in P_μ.

Next, M_n sets up a function G computed by a polynomial time Turing machine with oracle access to the h_i's. Recall that query corresponding to μ is $\text{MOD}^{p_j}(F, 0^m)$, and therefore M has set up a polynomial time Turing machine M_F with access to the f_i's which computes F. The idea is to compute G by simulating M_F, except that each query to $f_i(u)$ is replaced by a query to $h_i(u)$. However, h_i is defined in terms of g_i for $i > j$, so this process is described more carefully. Let $M_F^{\vec{h}}$ simulate M_F, except that if M_F queries $f_i(u)$ for $i \leq j$, then $M_F^{\vec{h}}$ instead skips the query and continues to the subtree where the $f_i(u)$ is answered with $h_i(u)$. If M_F queries $f_i(u)$ for $i > j$, then $M_F^{\vec{h}}$ queries $g(\varphi_i(u))$ and continues to the subtree where the query $f_i(u)$ is answered with $\varphi_i^{-1}(g(\varphi_i(u))$. Let G be the function computed by $M_F^{\vec{h}}$.

The final step of M_n is to query $\text{MOD}^{p_j}(G, 0^m)$, and verify the correctness of the answer. Suppose M_n receives $x \in \text{MOD}^{p_j}(G, 0^m)$. Since the verification of any answer to $\text{MOD}^{p_j}(F, 0^m)$ solves S on $\{f_i\}_{i>j}$, verifying that $x \in \text{MOD}^{p_j}(G, 0^m)$ reveals a solution to S on $\{h_i\}_{i>j}$ by construction of G. Clearly the solution to S

on the h_i's pulls back to a solution to S on $\{g_i\}_{i>j}$'s via the φ_i's. This completes the definition of M_n, and the proof of the claim. □

The following argument concludes the proof of Theorem 7.6.1. The claim, Corollary 7.5.3, and Lemma 7.5.4 imply that there is an infinitely-often many-one reduction from MOD^d to MOD^{p_j}, where $d = \prod_{i=j+1}^{k+1} p_i$. By the discussion in Section 7.5, Theorem 3.3.1 extends to infinitely-often reductions. Thus for each n in A, there exists constant depth, polynomial size (in 2^n) propositional LK proofs of $F_{\mathrm{MOD}^d,n} \longrightarrow G_{\mathrm{MOD}^d,n}$ from instances of $F_{\mathrm{MOD}^{p_j},m} \longrightarrow G_{\mathrm{MOD}^{p_j},m}$. By Lemmas 7.4.3 and 7.4.4 there are constant depth, polynomial size proofs of $Count_d^{2^n}$ from instances of $Count_{p_j}^{2^m}$. Since each $p_i > 2$, d and 2 are relatively prime. Thus there is an $0 < i < d$ such that there are infinitely many n such that there are proofs as above and $2^n \equiv i \bmod d$. Since p_j and 2 are also relatively prime, $2^m \not\equiv 0 \bmod p_j$. Therefore Theorem 3.5 of [3] implies there is a $0 < j < p_j$ with infinitely many N', $N' \equiv i \bmod d$, a constant ℓ and a number $M = N^{O(1)}$, $M \equiv j \bmod p_j$, such that there exists a (p, p_j, ℓ, M)-generic system over a set V of size N'. However, since d and p_j are relatively prime, Lemma 3.10 of [3] implies there does *not* exist such a generic system. This contradiction finishes the proof of Theorem 7.6.1.

This chapter contains material from the paper "Propositional proofs and reductions between NP search problems" which is currently accepted for publication by the Annals of Pure and Applied Logic. This paper is co-authored by the dissertation author and Samuel R. Buss.

Chapter 8

The MOD^d Counting Principles

8.1 Overview

The MOD^d counting principles were formulated in Section 7.4, but we restate them for sake of convenience. Let d be an integer with $d > 1$. If d is not a power of 2, MOD^d is Q_Φ, where Φ is the formula

$$\exists x[\bigvee_{\substack{s|d \\ s \neq d}} f^{(s)}(x) = x \vee f^{(d)}(x) \neq x].$$

If $d = 2^k$, MOD^d is Q_Φ, where Φ is the prenex form of the formula

$$f(0) = 0 \rightarrow \exists x[\bigvee_{\substack{s|d \\ s \neq d}} (x \neq 0 \wedge f^{(s)}(x) = x) \vee f^{(d)}(x) \neq x].$$

The notation $s|d$ indicates that s divides d. Note that the problem LONELY defined in Section 2.5 is exactly the problem MOD^2.

These principles intuitively state that a set V such that $|V| \not\equiv 0 \bmod d$ cannot be partitioned into sets of size d. As mentioned in Section 7.4, these principles have been studied outside the context of TFNP^2 problems, in particular the *Count* formulas of [3]. The MOD^d principles are of interest because the MOD^d principles are NP search problems and the *Count* formulas are not. The only modular counting principle previously studied as an NP search problem is the problem LONELY. The problem LONELY served as the template for generalizing to the MOD^d principles.

For each d, the many-one closure of MOD^d defines a new class of TFNP problems called PMOD^d (for "polynomial d modular argument"); thus PPA is exactly PMOD^2. This chapter begins to resolves the relative complexity of the PMOD^d's with other common TFNP classes. The picture is not complete, and some the current results require d to be prime while others do not.

The fact that MOD^2 is LONELY is a good starting place for resolving the relative complexity of the MOD^d principles. In particular, OntoPIGEON $\leq_{\text{m}}$ MOD^2, MOD^2 $\not\leq_{\text{T}}$ PIGEON, LeftPIGEON $\not\leq_{\text{T}}$ MOD^2 are shown in [2] (Corollary 3.3.2 also shows MOD^2 $\not\leq_{\text{T}}$ PIGEON), MOD^2 $\not\leq_{\text{T}}$ ITER is shown in Corollary 3.3.2, and ITER $\not\leq_{\text{T}}$ MOD^2, PIGEON $\not\leq_{\text{T}}$ MOD^2 are shown in Corollary 4.3.2.

We first state the results that generalize to any $d > 1$. Theorem 8.2.1 proves the generalization OntoPIGEON $\leq_{\text{m}}$ MOD^d and Theorem 8.2.5 proves the generalization MOD^d $\not\leq_{\text{T}}$ PIGEON; the latter separation immediately gives MOD^d $\not\leq_{\text{T}}$ LeftPIGEON. It is shown in [29] that OntoPIGEON $\not\leq_{\text{T}}$ ITER, so that MOD^d $\not\leq_{\text{T}}$ ITER.

Now let p, q be distinct primes. Theorem 8.2.4 proves the separations PIGEON $\not\leq_{\text{T}}$ MOD^p, ITER $\not\leq_{\text{T}}$ MOD^p, and MOD^p $\not\leq_{\text{T}}$ MOD^q by using the degree p Nullstellensatz refutations of MOD^p over fields of characteristic p constructed in Lemma 8.2.3. The only known proof of the first two separations is by this method, and so it is not immediately clear whether they generalize to composite p; we conjecture, however, that the first two separations do hold with composite p.

The separation MOD^p $\not\leq_{\text{T}}$ MOD^q has another proof using the LK translations. This proof uses the characterization of provability between *Count* formulas and the equivalence of the *Count* formulas with the MOD^d principles. Because the provability between *Count* formulas is known even for composite p and q it seems possible to characterize when MOD^p $\not\leq_{\text{T}}$ MOD^q for composite p and q; however, this is left as an open question.

Another open question is whether LeftPIGEON is Turing reducible to MOD^d for any d. We conjecture that there is no such reduction when d is prime. Most of the proof method of [2] for the case $d = 2$ seems to generalize easily to other d. However, since the proof uses simplifications by using fields with characteristic 2, it

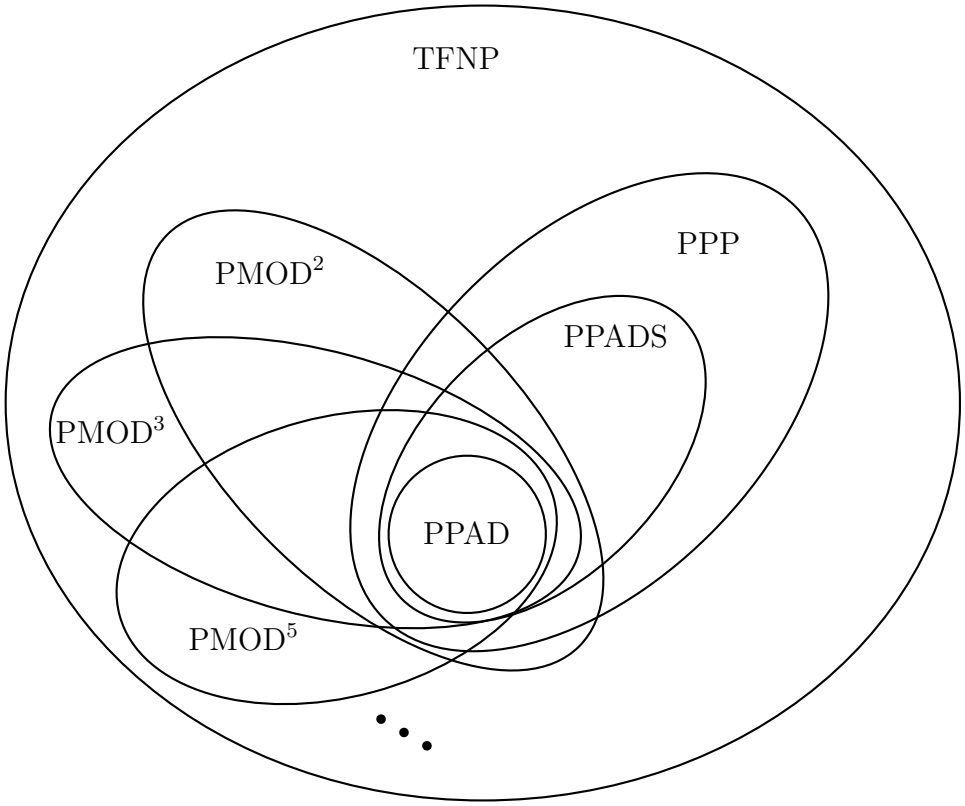

Figure 8.1: Some separations and inclusions of TFNP^2 classes in a generic relativized setting. The separation of PPADS from the PMOD^p is conjectured.

seems that a direct generalization of their proof would require similar simplifications with characteristic d. This is why we only conjecture a separation in the case when d is prime; the case for non-prime d is left as an open question.

The above mentioned relations are summarized in Figure 8.1, which is a picture of TFNP in a generic, relativized world. Figure 8.1 augments Figure 1.1 from Chapter 1 by adding in the classes PMOD^p for p a prime. The three dots in Figrue 8.1 indicate that PMOD^7, PMOD^{11}, ... continue the pattern. As mentioned above the separation of PPADS and PMOD^p for all p is only a conjecture. The classes MOD^d for composite d are left out of Figure 8.1 for clarity, even though there are results with these classes, since there are too many conjectured and open separations.

8.2 Reduction and Separation Proofs

This section proves all the reductions and separations mentioned above.

Theorem 8.2.1. *Let* $d > 1$. *Then* OntoPIGEON $\leq_{\mathrm{m}}$ MOD^d.

Proof. Note that d is allowed to be composite. The proof only considers the case when d is not a power of 2. The proof carries through when d is a power of 2 with minor changes to account for the fact that 0 is omitted from the partition.

Recall that OntoPIGEON is the first-order TFNP^2 problem defined by the $\exists$-sentence

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

This is equivalent to the problem SOURCE.or.SINK, which is "Given a directed graph on U_n of in- and out-degree at most 1 such that 0^n is source, find a different source or any sink." The reduction to SOURCE.or.SINK is given by creating a graph G where the directed edge (u, v) is in G if and only if $f(u) = v$ and $g(v) = u$. It is easy to check that u is a source if $f(g(u)) \neq u$, and u is a sink if $g(f(u)) \neq u$.

The intuition for the reduction comes from SOURCE.or.SINK, and we illustrate the construction in Figure 8.2 in the case $d = 3$. We want to make d copies of the graph input to SOURCE.or.SINK. Suppose the size parameter for SOURCE.or.SINK is n. We create an instance of MOD^d with size parameter $d+n$, which corresponds to 2^d copies of the input graph to SOURCE.or.SINK. (Note that since we want d copies it would be enough for the size parameter to be $\lceil \log d \rceil + n$; however, this makes no essential difference since d is constant.) We use the first d copies, leaving $2^d - d$ extra copies of the input graph, and these extra copies contain $2^n(2^d - d)$ individual points. In Figure 8.2 this means that the first three rows are the ones we are interested in and the last five rows are extra. We would like to partition the extra rows into disjoint d-cycles so that there is no solution to MOD^d there. However, d does not divide $2^n(2^d - d)$, since d is not a power of 2. Thus, we add $d - \ell$ of the copies of 0^n to the extra rows (leaving ℓ copies of 0^n), where ℓ is chosen so that the number of these points is divisible by d. Then these points can be partitioned into disjoint d-cycles, so that these points are effectively ignored by

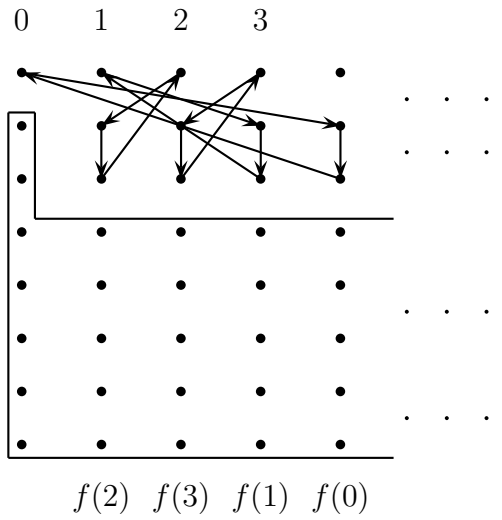

Figure 8.2: An example of the construction for Theorem 8.2.1 in the case $d = 3$, so there are $2^d - d = 5$ extra copies of U_n. The boxed points are the extra copies of U_n combined with enough copies of 0 to make the number of boxed points divisible by d. The arrows in the first three rows show the behavior of the function F input to MOD^d.

MOD^d. In Figure 8.2 $\ell = 1$ and the five extra rows and two copies of 0^n are boxed to indicate that they are inconsequential.

We describe the function F input to MOD^d on the remaining points. The function F attempts to partition the remaining points into disjoint d-cycles by (1) collecting the first ℓ copies of u with the last $d - \ell$ copies of $f(u)$, and (2) collecting the last $d - \ell$ copies of u with the first ℓ copies of $g(u)$. The case (2) does not arise for $u = 0^n$, since there are no such copies (they were added to extra rows); this corresponds to the fact that 0^n is supposed to be a source. Figure 8.2 illustrates this behavior. It is implicit in Figure 8.2 that, for example, $f(1) = 3$, $f(2) = 1$ and $g(1) = 2$, $g(3) = 1$. Note that if $f(u) = v$ and $g(v) = u$ a d-cycle is formed using points for the u^{th} and v^{th} columns, for example $u = 1$ and $v = 3$ in Figure 8.2. Thus those points where f and g are inverses of each other do not solve MOD^d on F. In fact, we show below that the only ways a d-cycle can fail to be formed is if (1) or (2) failed to make a d-cycle. Furthermore, the failure of (1) finds a sink and the failure of (2) finds a source. Note that the source found in case (2) cannot be 0^n, and thus this solves OntoPIGEON. The rest of the proof provides the details of

this argument.

Fix n and let $f, g : U_n \to U_n$ be the inputs to OntoPIGEON. We construct a many-one reduction M that creates a function $F : U_{d+n} \to U_{d+n}$, queries $\text{MOD}^d(F, 0^{d+n})$ and returns a solution to OntoPIGEON$(f, g, 0^n)$.

The domain of F is U_{d+n}, and thus can be thought of as 2^d copies of U_n. A string $w \in U_{d+n}$ will be written as $\langle u, v\rangle$, where u is the first d bits of w and v is the last n bits of w. We will only need d copies of U_n of the 2^d total copies in U_{d+n}. Let V_u be the set $\{\langle u, v\rangle : v \in U_n\}$ and let $c_i = 0^{d-i}10^{i-1}$ for $i = 1, \ldots, d$. The i^{th} copy of U_n will be associated with V_{c_i}.

The remaining copies of U_n are V_u where $u \neq c_1, \ldots, c_d$. Since these copies are not needed, we would like to define F on $W = \bigcup_{u \neq c_1, \ldots, c_d} V_u$ such that F does contain a solution in W; that is we want to define F so that its orbits partition W into sets of size d. However, since W contains $2^d - d$ copies of U_n, the size of W is $(2^d - d)2^n$. Since d is not a power of 2, $|W|$ is not a multiple of d. To resolve this issue, we add enough elements from V_{c_1} to make $|W|$ divisible by d. Let $\ell \equiv (2^d - d)2^n \bmod d$ for $0 < \ell < d$. Let X be the set $\{\langle c_i, 0^n\rangle : \ell < i \leq d\}$. Since $|X| = d - \ell$ and $W \cap X = \emptyset$, it is clear that $|W \cup X| \equiv 0 \bmod d$. Define F in some arbitrary way on $W \cup X$ so that the orbits of $F\restriction_{W \cup X}$ partition $W \cup X$ into sets of size d. Define F on $U_{d+n} \backslash (W \cup X)$ as follows:

$$F(u, v) = \begin{cases} \langle c_{i+1}, v\rangle & \text{if } u = c_i \text{ for } 1 \leq i < d, i \neq \ell \\ \langle c_{\ell+1}, f(v)\rangle & \text{if } u = c_\ell \\ \langle c_1, g(v)\rangle & \text{if } u = c_d \end{cases}.$$

Note that the only time F can map a pair $\langle u, v\rangle \notin W \cup X$ to a pair $\langle u', v'\rangle \in W \cup X$ is when $u = c_\ell, v = 0^n$ and $f(0^n) = 0^n$.

The reduction M first queries $f(0^n)$. If $f(0^n) = 0^n$, then M halts and outputs 0^n as a solution to OntoPIGEON$(f, g, 0^n)$. Otherwise $f(0^n) \neq 0^n$, and hence $F(\langle u, v\rangle) \notin W \cup X$ if $\langle u, v\rangle \notin W \cup X$. Next, M queries $\text{MOD}^d(F, 0^{d+n})$. Suppose M receives answer $\langle u, v\rangle$. There are two ways that $\langle u, v\rangle$ can solve $\text{MOD}^d(F, 0^{d+n})$: $F^{(s)}(\langle u, v\rangle) = \langle u, v\rangle$ for some $s|d$ such that $s \neq d$, or $F^{(d)}(\langle u, v\rangle) \neq \langle u, v\rangle$. We claim that the first option does not occur. First, $\langle u, v\rangle$ is not in $W \cup X$ since

the orbits of F partition $W \cup X$ into sets of size d. Also, for any $w \in U_n$, $F^{(s)}(\langle c_i, w\rangle) = \langle c_{(i+s) \bmod d}, y\rangle$ for some y, since F maps $U_{d+n}\backslash(W \cup X)$ to itself. Therefore, if $F^{(s)}(\langle u, v\rangle) = \langle u, v\rangle$ then $s = 0 \bmod d$, which is a contradiction.

Therefore, it must be that $u = c_i$ for some $1 \leq i \leq d$, $\langle u, v\rangle \notin W \cup X$, and $F^{(d)}(\langle c_i, v\rangle) \neq \langle c_i, v\rangle$. We claim that v is a solution to OntoPIGEON$(f, g, 0^n)$. If $1 \leq i \leq \ell$, then $F^{(d)}(\langle c_i, v\rangle) = \langle c_i, g(f(v))\rangle$, and thus $g(f(v)) \neq v$ so that v is a solution. If $\ell < i \leq d$, then $F^{(d)}(\langle c_i, v\rangle) = \langle c_i, f(g(v))\rangle$, and thus $f(g(v)) \neq v$. Note that since $\langle c_i, v\rangle$ is, in particular, not in X, it must be that $v \neq 0^n$, so that again v is a solution. The reduction M thus finishes by outputting v. □

The following corollary follows from Theorem 8.2.1 and the separation OntoPIGEON $\not\leq_{\mathrm{T}}$ ITER proved in [29].

Corollary 8.2.2. *Let $d > 1$. Then* MOD$^d \not\leq_{\mathrm{T}}$ ITER*.*

The rest of the section proves the remaining separations involving MOD^d. Theorem 8.2.4 proves separations when d is a prime and Theorem 8.2.5 proves a separation without restriction on d. The next lemma is used to prove Theorem 8.2.4.

Lemma 8.2.3. *Let p be a prime. Then* MOD^p *has degree p Nullstellensatz refutations over any field of characteristic p.*

Proof. The case $p = 2$ is shown in [9], so let $p > 2$. The proof of the case $p > 2$ is similar in spirit to the proof of the case $p = 2$. However, when $p > 2$ the increased cycle length adds complications not present when $p = 2$.

Fix n and let $S_k = \{(u_0, \ldots, u_{k-1}) | u_i \in U_n, i = 0, \ldots, k-1\}$. Elements $(u_0, \ldots, u_{k-1})$ of S_k will be denoted $\vec{u}$. Let $x_{\vec{u}}$ be $\prod_{i=0}^{k-2} x_{u_i,u_{i+1}}$ and let $y_{\vec{u}}$ be $x_{\vec{u}} x_{u_{k-1},u_0}$. (If $\vec{u} \in S_1$, then $x_{\vec{u}} = 1$ and $y_{\vec{u}} = x_{u_0,u_0}$.) Let S be S_p.

Recall that the Nullstellensatz translation defined in Section 4.2 creates three sets of polynomials for a TFNP^2 problem. These polynomials for MOD^p are

(i) $x_{\vec{u}} x_{u_{p-1},u_p}$, for any $\vec{u} = (u_0, \ldots, u_{p-1}) \in S$ and $u_p \in U_n$ such that $u_0 \neq u_p$.

(ii) $x_{u,u}$, for any $u \in U_n$.

(iii) $\sum_{v \in U_n} x_{u,v} - 1$, for any $u \in U_n$.

(iv) $x_{u,v}x_{u,v'}$, for any $u \in U_n$ and $v, v' \in U_n$ such that $v \neq v'$.

The polynomials in (iii) and (iv) correspond to the totality and functionality, respectively, of the function f input to MOD^p. The polynomials in (ii) enforce the condition that $f(u) \neq u$. A polynomial $x_{\vec{u}}x_{u_{p-1},u_p}$ from (i) expresses that $f^{(p)}(u_0) \neq u_p$ for all $u_p \neq u_0$. In other words, $f^{(p)}(u_0) = u_0$ for all u_0. Since p is prime, these conditions state that the orbits of f partition U_n into sets of size p.

Begin by fixing $\vec{u} = (u_0, \ldots, u_{p-1}) \in S$ and sum the polynomials in (i) to obtain

$$\sum_{\substack{u_p \in U_n \\ u_p \neq u_0}} x_{\vec{u}}x_{u_{p-1},u_p}. \tag{8.1}$$

Take the polynomial of (iii) for u_{p-1}, and multiply by $x_{\vec{u}}$ to obtain

$$x_{\vec{u}}(\sum_{u_p \in U_n} x_{u_{p-1},u_p} - 1) = \sum_{u_p \in U_n} x_{\vec{u}}x_{u_{p-1},u_p} - x_{\vec{u}}. \tag{8.2}$$

Then subtract (8.2) from (8.1) to obtain

$$x_{\vec{u}} - y_{\vec{u}}. \tag{8.3}$$

Next we claim that we can derive $\sum_{\vec{u} \in S} y_{\vec{u}}$. To do this, divide the vectors in S into two different groups, singletons and non-singletons. A vector $\vec{u} \in S$ is a *singleton* if $u_0 = \cdots = u_{p-1}$. If $\vec{u}$ is a singleton then $y_{\vec{u}}$ is derivable from (ii), so it is enough to look at the sum of $y_{\vec{u}}$ over non-singletons $\vec{u} \in S$.

Consider $\vec{u} \in S$. For $0 \leq i < p$, define $\vec{u}_i$ to be $(u_i, \ldots, u_{p-1}, u_0, \ldots, u_{i-1})$, a cyclic permutation of $\vec{u}$. Define an equivalence relation $\sim$ on vectors in S where $\vec{u} \sim \vec{v}$ if and only if $\vec{v} = \vec{u}_i$ for some $i = 0, \ldots, p-1$. Note that if $\vec{u} \sim \vec{v}$ then $y_{\vec{u}} = y_{\vec{v}}$. Let $C_{\vec{u}}$ be $\{\vec{u}_i | 0 \leq i < p\}$ so that $C_{\vec{u}}$ is the set of $\vec{v}$'s that are equivalent to $\vec{u}$. Note $|C_{\vec{u}}| = 1$ if and only if $\vec{u}$ is a singleton.

Claim: If $|C_{\vec{u}}| < p$ then $|C_{\vec{u}}| = 1$.

Proof. Since $\vec{u}_0, \ldots, \vec{u}_{p-1}$ are the p possible elements of $C_{\vec{u}}$, if $|C_{\vec{u}}| < p$, there must be a $0 < j < p$ such that $\vec{u} = \vec{u}_j$. Then for all $0 \leq \ell < p$, $u_\ell = u_{(\ell+j) \bmod p}$. Fix $0 \leq \ell < p$. Since j is relatively prime to p, the sequence $u_{\ell \bmod p}, u_{(\ell+j) \bmod p}, u_{(\ell+2j) \bmod p}, \ldots$ eventually contains u_0. Therefore $u_\ell = u_0$ for all ℓ, and hence $\vec{u}$ is a singleton. $\square$

Since $|C_{\vec{u}}| \leq p$, each equivalence class $[\vec{u}]$ has either 1 or p elements. Then

$$\sum_{\vec{u} \in S} y_{\vec{u}} = \sum_{[\vec{u}] \in S/\sim} \sum_{\vec{v} \in [\vec{u}]} y_{\vec{v}} =$$

$$\sum_{\substack{[\vec{u}] \in S/\sim \\ |[\vec{u}]|=1}} y_{\vec{u}} + \sum_{\substack{[\vec{u}] \in S/\sim \\ |[\vec{u}]|=p}} \sum_{\vec{v} \in [\vec{u}]} y_{\vec{u}} \quad = \quad \sum_{\substack{[\vec{u}] \in S/\sim \\ \vec{u} \text{ is a singleton}}} y_{\vec{u}} \quad + \sum_{\substack{[\vec{u}] \in S/\sim \\ |[\vec{u}]|=p}} p y_{\vec{u}}.$$

The first summation of the last expression is derivable since the sum is over singletons, and the second summation is 0 since we are over characteristic p (this is the only place where the characteristic plays a role). Derive

$$\sum_{\vec{u} \in S} x_{\vec{u}} \tag{8.4}$$

by summing (8.3) over all $\vec{u} \in S$ and adding $\sum_{\vec{u} \in S} y_{\vec{u}}$.

We now show by downward induction on k how to derive

$$\sum_{\vec{u} \in S_k} x_{\vec{u}} \tag{8.5}$$

for $1 \leq k \leq p$. The base case is when $k = p$ and is shown by (8.4).

For the induction step, assume (8.5) holds for $k > 1$, we show how to derive it for $k - 1$. Fix an arbitrary $\vec{u}' = (u_0, \ldots, u_{k-2}) \in S_{k-1}$. Use the polynomial in (iii) with $u = u_{k-2}$ and multiply by $x_{\vec{u}'}$, to obtain

$$x_{\vec{u}'} \left(\sum_{u_{k-1} \in U_n} x_{u_{k-2}, u_{k-1}} - 1 \right) = \left(\sum_{u_{k-1} \in U_n} x_{\vec{u}} \right) - x_{\vec{u}'}, \tag{8.6}$$

where $\vec{u} = (u_0, \ldots, u_{k-2}, u_{k-1})$. Then sum (8.6) over all $\vec{u}' \in S_{k-1}$ to derive

$$\sum_{\vec{u} \in S_k} x_{\vec{u}} - \sum_{\vec{u}' \in S_{k-1}} x_{\vec{u}'} \tag{8.7}$$

Since (8.5) is derivable by the inductive hypothesis, add it to the negative of (8.7). The resulting polynomial is exactly (8.5) with k replaced by $k - 1$, which finishes the induction.

Now apply (8.5) with $k = 1$ to obtain $\sum_{u_0 \in U_n} 1 = 2^n$. Since $p \neq 2$ is prime, 2^n has a multiplicative inverse modulo p, and hence we can derive 1. It is easy to check that the degree of the polynomials used is always bounded by p. $\square$

The Nullstellensatz refutation never needs to make use of polynomials in (iv). This differs from the proof of the case $p = 2$ from [9]. That the functionality of f is never needed corresponds to the fact that MOD^p can be expressed as a basic first-order TFNP^2 problem, meaning that in the defining total $\exists$-sentence formulas of the form $f(\vec{x}) = y$ only occur positively. Thus f not being functional only admits more solutions. In essence, the transformation to a basic first-order TFNP^2 problem assumes functionality, so it is not needed again. An example illustrating this is shown in Section 5.1. The proof of the case $p = 2$ from [9] requires the functionality polynomials since they allow formulas of the form $f(\vec{a}) \neq b$ in their $\exists$-sentences (which is disallowed in a basic first-order TFNP^2 problem).

Lemma 8.2.3 combines with Theorem 4.3.1 to give the following three separations.

Theorem 8.2.4. *Let p and q be distinct primes. Then* $\text{PIGEON} \not\leq_{\text{T}} \text{MOD}^p$, $\text{ITER} \not\leq_{\text{T}} \text{MOD}^p$, *and* $\text{MOD}^p \not\leq_{\text{T}} \text{MOD}^q$.

Proof. If $\text{PIGEON} \leq_{\text{T}} \text{MOD}^p$ or $\text{ITER} \leq_{\text{T}} \text{MOD}^p$, then by Theorem 4.3.1 and Lemma 8.2.3 there are $n^{O(1)}$ degree Nullstellensatz refutations of PIGEON and ITER over any field with characteristic p. But [2, 10, 16] show that constant degree refutations do not exist, regardless of the base field. It is shown in [11] that MOD^p requires linear (in the size of the set being partitioned) degree refutations over fields with characteristic $q \nmid p$. Since MOD^p partitions U_n, the degree lower bound on a Nullstellensatz refutation of MOD^p over a field with characteristic q is 2^n. However, if $\text{MOD}^p \leq_{\text{T}} \text{MOD}^q$ then Lemma 8.2.3 implies that there are $n^{O(1)}$ degree Nullstellenstaz refutations of MOD^p over any field of characteristic q, which is a contradiction. □

It is worth noting that the separation $\text{MOD}^p \not\leq_{\text{T}} \text{MOD}^q$ also follows from the propositional LK translation. Namely, if there were such a Turing reduction, then by Theorem 3.3.1 there would be constant depth, polynomial size proofs of $F_{\text{MOD}^p,n} \longrightarrow G_{\text{MOD}^p,n}$ from instances of $F_{\text{MOD}^q,m} \longrightarrow G_{\text{MOD}^q,m}$. By Lemmas 7.4.4 and 7.4.3, there would be constant depth, polynomial size proofs of $Count_p^N$ from instances of $Count_q^M$, for appropriately chosen N and M (the choice depends on

whether p is 2 or not). But the provability results of [3] can be applied as in the proof of Theorem 7.6.1 to show such proofs cannot exist.

The provability between the *Count* formulas is completely characterized in [3], not just in the case when p, q are prime. This suggests that deciding whether there is a reduction from MOD^p to MOD^q for arbitrary p, q can be similarly characterized. However, the conditions on p, q that are necessary and sufficient to guarantee the existence of a Turing reduction from MOD^p to MOD^q are left open.

The next separation is proved in [2] in the case $p = 2$. The following argument extends this result to any $d > 1$, and closely follows the argument of [2]

Theorem 8.2.5. *Let $d > 1$. Then $\text{MOD}^d \not\leq_{\text{T}} \text{PIGEON}$.*

Proof. Assume for a contradiction that M is a Turing reduction from MOD^d to PIGEON. Let the input to MOD^d be $f : U_n \to U_n$. We show that, for sufficiently large n, each oracle query by M can be correctly answered by setting polynomially new values to f without creating a solution to MOD^d. Therefore, M is forced to return an answer involving the unspecified part of f, which can then be set to make M's answer be incorrect. This contradicts M being a correct Turing reduction.

Suppose the i^{th} oracle query by M is to $f(u_0)$. Since only a polynomial part of f has been set, there are $d-1$ distinct, unset values $u_1, \ldots, u_{d-1}$, and f can be answered such that $f(u_i) = u_{i+1}$ for all $i = 0, \ldots, d-1$ and $f(u_{d-1}) = u_0$. Thus the query is answered without answering $\text{MOD}^d(f, 0^n)$.

Before continuing to the case where M queries PIGEON, we introduce some terminology. Let $u_1, \ldots, u_d \in U_n$ be distinct. Then $\lambda = \{u_i\}_{i=1}^d$ is a *d-cycle*. The *successor* of u_i is u_{i+1} if $1 \leq i < d$ and is u_1 if $i = d$. Two d-cycles λ and μ are *consistent* if (1) λ is a cyclic permutation of μ (in particular, λ and μ must contain the same elements), or (2) the elements of λ and the elements of μ are disjoint. The order of the d-cycle is important; if $u_1, u_2, u_3 \in U_n$ are distinct, then the 3-cycle u_1, u_2, u_3 is not consistent with the 3-cycle u_1, u_3, u_2. A set of d-cycles $\{\lambda_i\}$ is a *d-matching* if for all $i \neq j$, λ_i and λ_j are consistent. Note that at the time when M queries PIGEON, the values of f known by M form a d-matching, where each d-cycle is of the form $\{f^{(i)}(w)\}_{i=0}^{d-1}$.

Now suppose the i^{th} oracle query by M is to PIGEON on input $F : U_m \to$

U_m. Let the runtime of F be bounded by k, so that k is $n^{O(1)}$. We want paths in the decision tree for F to correspond to d-matchings of U_n. To accomplish this, alter F so that whenever its computation queries $f(w)$, it queries $f(f(w)), \ldots, f^{(d)}(w)$ before continuing. Thus when the computation of F queries w, either a solution to $\mathrm{MOD}^d(f, 0^n)$ is revealed or a new d-cycle of f is revealed. Note that the runtime of the altered F is bounded by dk. Let T_u be the decision tree for F in input u.

Prune T_u as follows. Prune all branches that make two queries to the same value of f (the answer to such a query is either redundant or contradicts the previously received value of f). Prune all branches that contain $f^{(d)}(w) \neq w$. Therefore, each remaining branch P in T_u induces a d-matching $\{\lambda_i\}$, where each d-cycle is of the form $w, f(w), f(f(w)), \ldots, f^{(d-1)}(w)$. Two branches are *mutually consistent* if they induce consistent d-matchings. The definition of consistency for d-cycle implies that two mutually consistent branchings must agree on common queries to f.

There are two ways that M can answer the PIGEON query without solving $\mathrm{MOD}^d(f, 0^n)$. The first is if there is a u so that $\mathrm{br}_{0^m}(T_u)$ is not empty, and the second is if there are mutually consistent paths $P \in \mathrm{br}_v(T_u)$ and $P' \in \mathrm{br}_v(T_{u'})$ for $u \neq u'$. The rest of the proof derives a contradiction from the failure of both cases; that is, assume for a contradiction that $\mathrm{br}_{0^m}(T_u)$ is empty for all u and that P and P' are mutually inconsistent for all $P \in \mathrm{br}_v(T_u)$ and $P' \in \mathrm{br}_v(T_{u'})$ such that $u \neq u'$.

The first step in deriving a contradiction is to define trees $\{S_v\}_{v \neq 0^m}$ which are intended to be "inverse trees" for the T_u's. That is, S_v is a tree such that a leaf label of u indicates that $F(u) = v$. (Note that there is no S_{0^m} since we are assuming for a contradiction that $\mathrm{br}_{0^m}(T_u)$ is empty for all u.) The construction of of S_v is similar to the fact that if a boolean formula ϕ and its negation can both be put in DNF form with conjunctions of size d, then ϕ can be expressed as a Boolean decision tree of height at most d^2.

To construct the S_v's, we introduce $R_v = \bigcup_u \mathrm{br}_v(T_u)$. An important property of R_v is that if $P, P' \in R_v$ and $P \neq P'$, then P and P' are mutually inconsistent. To see this, if P and P' are from the same T_u they are mutually inconsistent because they must differ on some value for f. If $P \in T_u$ and $P' \in T_{u'}$

for $u \neq u'$ then P and P' are mutually inconsistent, since this exactly one of the assumptions from which we are trying to derive a contradiction.

The tree S_v is defined to be the decision tree for the algorithm $A(v)$ defined as follows (again recall $v \neq 0^m$). The algorithm $A(v)$ attempts to find some u such that $F(u) = v$. The idea is that if $F(u) = v$ for some u, then there is a path in T_u which outputs v that is consistent with the known values of f. The set of all such paths is a subset of R_v, and contains all possible candidate paths to witness $F(u) = v$. The algorithm $A(v)$ iteratively picks a candidate path P and then checks if P is actually correct by querying f as determined by the vertex labels of P. If the answers to these queries match the edge labels on P, then P is a path that witnesses $F(u) = v$, so $A(v)$ outputs u. Otherwise, P does not witness that $F(u) = v$, so $A(v)$ picks a new candidate path that is consistent with the known values of f (including those values for f just obtained by querying the vertices of P); if no such paths remain then $A(v)$ outputs "unmapped" indicating that there is no u such that $F(u) = v$.

We describe $A(v)$ in more detail. The algorithm starts with the d-matching induced by the values of f known by M right before the PIGEON query. It then picks any path $P \in R_v$ consistent with the current d-matching and queries f on the vertices of P. We do not address the case when the answers to these queries do not form d-cycles, as this solves $\mathrm{MOD}^d(f, 0^n)$. The current d-matching is updated by adding the new d-cycles these queries to f revealed. If the newly updated d-matching is consistent with the d-matching induced by P, $A(v)$ outputs u. Otherwise a new path P' is chosen from some $\mathrm{br}_v(T_{u'})$, and the process repeats. The algorithm $A(v)$ is described by the following pseudocode:

Do

If there is no $P \in R_v$ consistent with the current d-matching:

Output "unmapped".

Else

Pick such a $P \in R_v$ and suppose $P \in \mathrm{br}_v(T_u)$. Query f on the vertices of P, and add the answers to the current d-matching. If P

is consistent with the current d-matching, output u.

Loop

We claim that $A(v)$ terminates with at most k iterations of the loop (and hence at most dk^2 queries to f). Let m_r be the matching during the r^{th} iteration of the loop. At the beginning of the r^{th} iteration of the loop, $A(v)$ either halts or picks a $P \in R_v$ consistent with m_r. Suppose $A(v)$ does not halt, so that it picks a $P \in R_v$ and consider any other $P' \in R_v$. Let P and P' induce d-matchings $\{\lambda_i\}$ and $\{\mu_i\}$, respectively. Since P and P' are mutually inconsistent, these d-matchings are inconsistent, and is a $w \in \lambda_i \cap \mu_{i'}$, such that the successor of w in λ_i is not equal to the successor of w in $\mu_{i'}$. After querying the vertices of P, each of the points in λ_i belong to some d-cycle of m_{r+1}, and in particular w is assigned to some d-cycle. If w is assigned to a d-cycle that is not a cyclic permutation of $\mu_{i'}$, then P' in inconsistent with m_{r+1}. If w is assigned to $\mu_{i'}$ by m_{r+1}, then the length of P' has been shortened by d queries. Therefore, at step $r+1$ the algorithm either halts or each remaining branch consistent with m_{r+1} has its length reduced by d. Since the branches started with length at most dk, the loop in $A(v)$ runs at most k times. Since each iteration queries at most dk values of f, $A(v)$ makes at most dk^2 queries to f. Let S_v be the decision tree for $A(v)$. We can ensure S_v has height exactly dk^2 by making dummy queries (these dummy queries are not allowed to reveal solutions to $\text{MOD}^d(f, 0^n)$).

We now create new trees T'_u from T_u. Consider the following algorithm: First compute $F(u)$, suppose this computation outputs v. Then run $A(v)$, recalling that we do not allow repeated queries to values for f, and let the output of the entire algorithm be v. Let T'_u be the decision tree for this algorithm.

Note that a path P_0 in T'_u starts as a path P in T_u followed by a path Q in some S_v. Since Q is necessarily consistent with P, it must be that Q extends a path in R_v. Since R_v contains mutually inconsistent paths, Q must extend P itself. Therefore P_0 and Q specify the same d-matching, and thus must have the same set of dk^2 edge labels. Then we can define a mapping φ from $\bigcup_{u,v} \text{br}_v(T'_u)$ to $\bigcup_{u,v} \text{br}_u(S_v)$. Furthermore, φ is one-to-one since distinct paths induce distinct d-matchings.

Now note that since the T_u's and S_v's have the same height and branching, $|\bigcup_w \mathrm{br}_w(S_v)| = |\bigcup_w \mathrm{br}_w(T_u)|$ for all u and all $v \neq 0^m$ (recall there is no tree for S_{0^m}). Then $|\bigcup_{u,v} \mathrm{br}_v(T'_u)| > |\bigcup_{u,v} \mathrm{br}_u(S_v)|$ since $v = 0^m$ does not appear in the second union. Therefore, the pigeonhole principle implies that φ is not one-to-one. But we showed above that φ is one-to-one. Thus we have obtained the desired contradiction, and the proof is finished.

□

Bibliography

[1] M. Agrawal, N. Kayal, and N. Saxena, *Primes is in P*, Annals of Mathematics, 160 (2004), pp. 781–793.

[2] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi, *The relative complexity of NP search problems*, J. Comput. System Sci., 57 (1998), pp. 3–19.

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

[4] P. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, P. Pudlák, and 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.

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

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

[7] W. D. Brownawell, *Bounds for the degrees in the Nullstellensatz*, Annals of Mathematics (second series), 126 (1987), pp. 577–591.

[8] H. Buhrman, L. Fortnow, M. Koucký, J. D. Rogers, and N. Vereshchagin, *Does the polynomial hierarchy collapse if onto functions are invertible?*, Theor. Comp. Sys., 46 (2009), pp. 143–156.

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

[10] S. R. Buss, *Lower bounds on Nullstellensatz proofs via designs*, in Proof Complexity and Feasible Arithmetics, P. Beame and S. Buss, eds., American Mathematical Society, 1998, pp. 59–71.

[11] S. R. Buss, D. Grigoriev, R. Impagliazzo, and T. Pitassi, *Linear gaps between degrees for the polynomial calculus modulo distinct primes*, Journal of Computer and System Sciences, 62 (2001), pp. 267–289.

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

[13] S. R. Buss and J. Krajíček, *An application of Boolean complexity to separation problems in bounded arithmetic*, Proc. London Math. Society, 69 (1994), pp. 1–21.

[14] L. Caniglia, A. Galligo, and J. Heintz, *Some new effectivity bounds in computational geometry*, in Applied Algebra, Algebraic Algorithms and Error Correcting Codes, Proceedings, Sixth International Conference, Rome 1988, Lecture Notes in Computer Science #357, T. Mora, ed., Berlin, 1989, Springer-Verlag, pp. 131–151.

[15] X. Chen and 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.

[16] M. Clegg, J. Edmonds, and 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.

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

[18] C. Daskalakis, P. W. Goldberg, and 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.

[19] S. A. Fenner, L. Fortnow, A. V. Naik, and J. D. Rogers, *Inverting onto functions*, in IEEE Conference on Computational Complexity, 1996, pp. 213–222.

[20] J. Hanika, *Search problems and bounded arithmetic*, in ECCCTH: Electronic Colloquium on Computational Complexity, theses, 2004.

[21] E. Jeřábek, *Abelian groups and quadratic residues in weak arithmetic*, Mathematical Logic Quarterly, 56 (2010), pp. 262–278.

[22] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, *How easy is local search?*, J. Comput. System Sci., 37 (1988), pp. 79–100.

[23] S. Kintali, *A compendium of ppad-complete problems.* `http://www.cc.gatech.edu/~kintali/ppad.html`.

[24] S. Kintali, L. J. Poplawski, R. Rajaraman, R. Sundaram, and S.-H. Teng, *Reducibility among fractional stability problems.*, in FOCS'09, 2009, pp. 283–292.

[25] J. Kollár, *Sharp effective Nullstellensatz*, Journal of the American Mathematical Society, 1 (1988), pp. 963–975.

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

[27] M. W. Krentel, *Structure in locally optimal solutions (extended abstract)*, in FOCS, 1989, pp. 216–221.

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

[29] T. Morioka, *Classification of search problems and their definability in bounded arithmetic*, master's thesis, University of Toronto, 2001.

[30] ——, *Logical Approaches to the Complexity of Search Problems: Proof Complexity, Quantified Propositional Calculus, and Bounded Arithmetic*, PhD thesis, University of Toronto, 2005.

[31] C. H. Papadimitriou, *On graph-theoretic lemmata and complexity classes (extended abstract)*, in Proceedings of the 31st IEEE Symposium on Foundations of Computer Science (Volume II), IEEE Computer Society, 1990, pp. 794–801.

[32] ——, *On the complexity of the parity argument and other inefficient proofs of existence*, J. Comput. System Sci., 48 (1994), pp. 498–532.

[33] C. H. Papadimitriou, A. A. Schffer, and M. Yannakakis, *On the complexity of local search (extended abstract)*, in STOC, 1990, pp. 438–445.

[34] A. A. Schffer and M. Yannakakis, *Simple local search problems that are hard to solve*, SIAM J. Comput., (1991), pp. 56–87.

[35] A. L. Selman, *A taxonomy of complexity classes of functions*, J. Comput. System Sci., 48 (1994), pp. 357–381.

[36] ——, *Much ado about functions*, in Eleventh Annual IEEE Conference on Computational Complexity, May 1996, pp. 198 –212.

[37] N. Thapen, *Some notes on the low complexity consequences of fragments of approximate counting*. October 2010.