Electronic Colloquium on Computational Complexity, Report No. 141 (2022)

TFNP Characterizations of Proof Systems and Monotone Circuits

Sam Buss
University of California, San Diego

Noah Fleming
Memorial University

Russell Impagliazzo
University of California, San Diego

October 19, 2022

Abstract

Connections between proof complexity and circuit complexity have become major tools for obtaining lower bounds in both areas. These connections — which take the form of interpolation theorems and query-to-communication lifting theorems — translate efficient proofs into small circuits, and vice versa, allowing tools from one area to be applied to the other. Recently, the theory of TFNP has emerged as a unifying framework underlying these connections. For many of the proof systems which admit such a connection there is a TFNP problem which *characterizes* it: the class of problems which are reducible to this TFNP problem via query-efficient reductions is *equivalent* to the tautologies that can be efficiently proven in the system. Through this, proof complexity has become a major tool for proving separations in black-box TFNP. Similarly, for certain monotone circuit models, the class of functions that it can compute efficiently is equivalent to what can be reduced to a certain TFNP problem in low communication. When a TFNP problem has both a proof and circuit characterization, one can prove an interpolation theorem. Conversely, many lifting theorems can be viewed as relating the communication and query reductions to TFNP problems. This is exciting, as it suggests that TFNP provides a roadmap for the development of further interpolation theorems and lifting theorems.

In this paper we begin to develop a more systematic understanding of when these connections to TFNP occur. We give exact conditions under which a proof system or circuit model admits a characterization by a TFNP problem. We show:

– Every well-behaved proof system which can prove its own soundness (a *reflection principle*) is characterized by a TFNP problem. Conversely, every TFNP problem gives rise to a well-behaved proof system which proves its own soundness.

– Every well-behaved monotone circuit model which admits a *universal family* of functions is characterized by a TFNP problem. Conversely, every TFNP problem gives rise to a well-behaved monotone circuit model with a universal problem.

As an example, we provide a TFNP characterization of the Polynomial Calculus, answering a question from [24], and show that it can prove its own soundness.

ISSN 1433-8092

1 Introduction

Connections between proof systems and monotone circuit models have revolutionized both areas, providing a large number of strong lower bounds that were previously not known. These connections take the form of

- *Interpolation Theorems*, which translate small proofs into efficient computations in an associated model of monotone circuit [6, 15, 16, 18, 29, 32–34, 39, 41, 43].
- *Query-to-Communication Lifting Theorems*, which translate efficient monotone computations into small proofs in an associated proof system [10, 13, 14, 20, 26–28, 31, 35, 37, 38, 42, 45].

This has allowed tools from one area to be applied to the other. Recently, the landscape of *total functional* NP (TFNP) has emerged as an organizing principle for connections between proof systems and models of monotone circuits [12, 25]. For many of the proof systems which admit an interpolation theorem or lifting theorem there is a TFNP problem which *characterizes* it in the following sense: the class of problems which are reducible to this TFNP problem via query-efficient reductions is *equivalent* to the set of tautologies that can be efficiently proven in the system. This has resulted in proof complexity becoming a major tool for proving separations in *black-box* TFNP. Conversely, the perspective offered by TFNP has provided a number interesting results for proof complexity, such as *complete* tautologies for certain proof systems, as well as striking *intersection theorems* [24].

An analogous phenomenon has emerged for monotone circuit complexity. For many monotone circuit models, the set of functions which can be computed efficiently is equivalent to the set of problems that can be reduced to a certain TFNP problem in low *communication*. When these TFNP problems collide — that is, when there is both a proof and circuit characterization of a particular TFNP problem — then we immediately obtain an interpolation theorem between this proof system and circuit model [44]! Moreover, many of the query-to-communication lifting theorems can be viewed as constructing a query-efficient reduction to a particular TFNP problems out of a communication-efficient reduction to that problem. This is exciting as it suggests understanding when TFNP problems admit such characterizations as a pathway for developing further connections between proof complexity and circuit complexity.

In this paper we give exact conditions under which a proof system or monotone circuit model admits a characterization by a TFNP problem. For proof complexity, we show that every well-behaved* proof system which can prove its own soundness (a *reflection principle*) is characterized by a TFNP problem. Conversely, every TFNP problem gives rise to a well-behaved proof system which proves its own soundness and which is closed under decision tree reductions. Furthermore, this result is constructive: for every TFNP problem we give a proof system which it characterizes. As an example, we provide a TFNP characterization of the Polynomial Calculus, answering a question from [24], and show that it can prove its own soundness. For circuit complexity, we show that every well-behaved monotone circuit model which admits a *universal family* of functions is characterized by a TFNP problem. Conversely, every TFNP problem gives rise to a well-behaved monotone circuit model with a universal problem.

1.1 Overview: Connections Proof Complexity, and Circuit Complexity, and TFNP

The connections between proof systems and monotone circuit models can be understood as relating the complexity of two families of total search problems whose complexity characterizes proof and circuit complexity respectively.

*We will say that a proof system of monotone circuit model is well-behaved if it satisfies some minor technical conditions discussed in subsection 1.2.

– *False Clause.* S_F for an unsatisfiable CNF formula $F = C_1 \wedge \cdots \wedge C_m$: given an assignment $x \in \{0,1\}^n$ output the index $i \in [m]$ of a clause such that $C_i(x) = 0$.

– *Monotone Karchmer-Wigderson.* mKW_f for a monotone boolean function f: given $x, y \in \{0,1\}^n$ such that $f(x) = 1$ and $f(y) = 0$ output $i \in [n]$ such that $x_i > y_i$.

The theory of TFNP studies the total search problems for which solutions can be efficiently verified. There is believed to be no complete problem for the class TFNP [40], and therefore much of the work on this subject has focused on identifying sub-classes which *do* admit complete problems. This has resulted in a rich landscape of classes which capture a wide variety of important problems in a range of areas including cryptography, economics, and game theory. These classes are typically defined as everything that can be efficiently reduced to a certain existence principle (of exponential size). For example, PPA is the class of search problems that can be reduced to an (exponential size) instance of the handshaking lemma. These exponential-size instances are given in a *white-box* fashion: they are represented as a polynomial-size circuit which can be queried to obtain each bit of the input.

The goal of TFNP is to understand how these classes relate. However, a separation between any pair of sub-classes would imply P $\neq$ NP. Instead, a line of work has sought to provide evidence of their relationships by proving *black-box* separations. As opposed to the white-box setting, one is only given oracle access to the circuit, which may be queried for each bit of the input; one is no longer allowed to see how the circuit is defined.

Black-Box TFNP and Proof Complexity. Beginning with [3], proof complexity has become a major tool for proving black-box TFNP separations. In fact, black-box TFNP — denoted TFNP^{dt} — can be viewed as the study of the false clause search problem. Every TFNP^{dt} problem is *equivalent* to S_F for some unsatisfiable CNF formula F. Using this connection, Göös et al. [25] observed that many prominent TFNP^{dt} problems are *characterized* by an associated proof system in the sense that the CNF formulas F that are efficiently provable in that proof system are exactly the problems S_F that are reducible to the TFNP^{dt} problem. This has led to the characterization of many of the prominent TFNP^{dt} subclasses:

– $\mathsf{FP}^{dt} = \mathsf{TreeRes}$ [36].
– $\mathsf{PLS}^{dt} = \mathsf{Res}$ [9].
– $\mathsf{PPA}^{dt} = \mathbb{F}_2\text{-}\mathsf{NS}$ [25].
– $\mathsf{PPADS}^{dt} = \mathsf{unary\text{-}NS}$ [24].
– $\mathsf{PPAD}^{dt} = \mathsf{unary\text{-}SA}$ [24].
– $\mathsf{SOPL}^{dt} = \mathsf{RevRes}$ [24].
– $\mathsf{EOPL}^{dt} = \mathsf{RevResT}$ [24].

Thus, separations between these proof systems translate into separations between their corresponding TFNP^{dt} subclasses. This has resulted in a complete picture of how the most prominent TFNP^{dt} subclasses relate [2, 7, 24, 25].

For proof complexity, the novel perspective provided by these TFNP^{dt} characterizations has given rise to a number of striking results. These include:

– *Complete Problems:* Any proof system which is characterized by a TFNP^{dt} problem S_F has F as its complete problem, in the sense that it has short proofs of exactly the formulas F' for which $\mathsf{S}_{F'}$ can be efficiently reduced to S_F. [25]

– *Intersection Theorems:* Proof systems which can efficiently prove a formula iff that formula has short proofs in several other proof systems [24].

– *Coefficient Separations:* Separations between the complexity of certain *algebraic* proof system when their coefficients are represented in unary versus binary [24].

However, there are many important TFNP^{dt} problems — such as PPP^{dt}-complete problems — which have thus far evaded characterization by a proof system. Conversely, there are many important proof systems for which no corresponding TFNP^{dt} problem is known. This raises the question of what conditions must be satisfied for such a characterization to occur.

Communication TFNP and Monotone Circuit Complexity. Karchmer and Wigderson [30] showed that the monotone formula complexity of any monotone function f is equal to the communication complexity of mKW_f. Building on this, Razborov [43] considered reductions between black-box TFNP classes where one measures the amount of *communication* needed to perform the reduction (for some suitable partition of the input), denoted TFNP^{cc}, and showed that a PLS^{cc}-complete problem characterizes monotone circuit complexity. There is good reason for this; analogous to how TFNP^{dt} is the study of the false clause search problem, TFNP^{cc} can be viewed as the study of the monotone Karchmer-Wigderson game. Indeed, every $R \in \mathsf{TFNP}^{cc}$ is equivalent to mKW_f (over the same partition of the variables) for some associated monotone function f [19, 25].

Following these results, a number of TFNP^{cc} problems have been characterized by models of monotone circuits [16,25]. However, there remain many important circuit models for which no TFNP^{cc}-characterization is known.

A Theory of Interpolation and Lifting Theorems. As we have just discussed, certain proof systems are characterized by TFNP^{dt} problems, while certain models of monotone circuits are characterized by problems in TFNP^{cc}. Göös et al. [25] observed that in all-known examples where both the black-box and communication variants of a TFNP problem R (denoted R^{dt} and R^{cc}) admit characterizations by a proof system and monotone circuit model respectively, then they admit both an interpolation theorem and a query-to-communication lifting theorem. This is to be expected, as a key component of both interpolation and query-to-communication lifting theorems proceeds by relating S_F to mKW_f for associated pairs (F, f). In fact, it is not difficult to see that whenever a TFNP class admits a characterization by both a proof system and a monotone circuit model then there is an interpolation theorem between this proof system and circuit model — this follows by the simple observation that communication protocols can simulate decision trees [44]! Thus, the landscape of TFNP, together with characterizations of TFNP problems by proofs and circuits, appears to provide a *roadmap* for potential interpolation and query-to-communication lifting theorems.

1.2 Our Results

Our first main result is a characterization of when a proof system admits a characterization by a TFNP^{dt} problem. We show that this occurs for any any proof system P which meets the following two criteria:

i) *Closure under decision-tree reductions:* whenever there is a small P-proof of a formula H, and S_F efficiently reduces to S_H, then there is also a small P-proof of F.

ii) *Proves its own soundness:* P can prove that P-proofs are sound. That is, P has small proofs of a reflection principle about itself, which is encoded in an efficiently-verifiable manner.

Conversely, we show that *every* TFNP^{dt} problem has a proof system which characterizes it. Furthermore, this proof system satisfies both conditions (i) and (ii). Out first main results can be informally stated as follows.

Theorem 1 (Informal)**.** *The following hold:*

- *For any* TFNP^{dt} *problem* R *there is a proof system* P *satisfying (i) and (ii) such that such that* R *characterizes* P *in the sense that* P *has short proofs of* F *iff* S_F *is efficiently reducible to* R*.*
- *For any proof system* P *which satisfies (i) and (ii) there is a* TFNP^{dt} *problem* R *such that* R *characterizes* P*.*

As an example of this we give a characterization of the Polynomial Calculus proof system by a TFNP^{dt} problem which we call $\mathsf{IND}_{\mathsf{PPA}}$ and show that it can prove its own soundness.

Theorem 2 (Informal)**.** $\mathsf{IND}^{dt}_{\mathsf{PPA}} = \mathbb{F}_2\text{-}\mathsf{PC}$ *and* $\mathbb{F}_2\text{-}\mathsf{PC}$ *has small proofs of an efficiently verifiable reflection principle about itself.*

Additionally, we sketch how this can be generalized in order to characterize the *unary* Polynomial Calculus over any field.

Our second main result is a characterization of the conditions under which monotone circuit models admit corresponding TFNP^{cc} problems. We formalize the concept of a monotone circuit model as a *monotone partial function complexity measure* (mpc) — a mapping of monotone functions to non-negative integers. We show that a TFNP^{cc} problem is characterized by a mpc iff the mpc meets the following criteria:

i) *Monotone under solutions:* if whenever g *solves* f — meaning that the *yes* and *no* inputs of a partial function f are a subset of the *yes* and *no* inputs of a partial function g — then $\mathsf{mpc}(g) \geq \mathsf{mpc}(f)$.

ii) *Closure under low-depth reductions:* if whenever f is a partial function and h is computable by a depth-d monotone Boolean circuit then $\mathsf{mpc}(f \circ h)$ is only polynomially larger in 2^d and $\mathsf{mpc}(f)$.

iii) *Admits a universal family:* a family of functions F_n such that whenever $\mathsf{mpc}(g) \leq n$ for a monotone partial function g, there is a string z_g so that $F(x \circ z_g)$ solves $g(x)$.

Theorem 3 (Informal)**.** *Let* mpc *be a complexity measure. There is a* $R \in \mathsf{TFNP}^{cc}$ *such that* R^{cc} *characterizes* mpc *iff* mpc *satisfies (i) – (iii).*

Furthermore, if we relax the universal function requirement (iii) to only require that mpc admits a *complete family* of functions — a generalization of universal functions to allow for low-depth reductions from the other functions with low-mpc to F_m, rather than requiring them to be a restriction of F_m — then we are able to obtain an even tighter characterization, detailed in Theorem 12.

Finally, we investigate whether this characterization can be extended from partial function complexity measures to *total function* measures. Since complexity measures on total functions induce measures on partial functions, this allows us to give a general condition under which a complexity measure on total functions has a TFNP^{cc} characterization (Theorem 16) by applying Theorem 3.

A Note on the Provability of Reflection Principles. Theorem 1 establishes that the property of P having short proofs of a reflection principle about itself is closely related to having a TFNP^{dt} characterization of P. The reflection principle for propositional proof systems has already been studied in prior work. In particular, Cook [11] showed that extended Frege (eF) has short proofs its consistency statements, and Buss [8] showed that Frege (F) has short proofs of its consistency statements. From their results, it follows readily that both proof systems, extended Frege and Frege, have short (polynomial size) proofs of their reflection principles. It is also well-known that the extended Frege and Frege proof systems can be characterized as

very strong TFNP^{dt} classes characterizable in terms of second-order theories of bounded arithmetic, see [5]. Analogous results were obtained for even stronger propositional proof systems by [22]. On the other hand, Garlik [21] showed that resolution requires exponential length for refutations of (a particular "leveled" version of) its reflection principle; Atserias-Müller [1] gave exponential lower bounds on resolution refutations of a relativized reflection principle.

Theorem 1 requires that the proof system P has short proofs of a variant of a reflection principle about itself. There are two main differences between our encodings and previous ones in the literature. The first is that the reflection principle is parameterized by a *complexity* parameter c (see Section 2) rather than the typical size parameter. The second is that the reflection principle must be *efficiently verifiable*, meaning that an error in the purported P-proof in the reflection principle can always be verified by examining in a small number of bits. Thus, for example, the bound of Garlik [21] does not contradict our results.

2 Proof Complexity and Black-Box TFNP

We begin by defining black-box TFNP. A *total search problem* is a sequence of relations $R_n \subseteq \{0,1\}^n \times \mathcal{O}_n$, one for each $n \in \mathbb{N}$ which is *total* — for each $x \in \{0,1\}^n$ there is $i \in \mathcal{O}$ such that $(x,i) \in R_n$. A total search problem is in TFNP^{dt} if solutions are also *verifiable*: for each $i \in \mathcal{O}$ there there is a decision tree T_i^o of $\mathsf{polylog}(n)$ depth such that

$$T_i^o(x) = 1 \iff (x,i) \in R_n.$$

Reductions and TFNP Subclasses. A *decision tree reduction* from $Q \in \{0,1\}^s \times \mathcal{O}'$ to $R \subseteq \{0,1\}^n \times \mathcal{O}$ is a set of decision trees $T_i : \{0,1\}^s \to \{0,1\}$ for $i \in [n]$ and $T_j^o : \{0,1\}^s \to \mathcal{O}'$ for $j \in \mathcal{O}$ such that for any $x \in \{0,1\}^s$,

$$((T_1(x), \ldots, T_n(x), j) \in R \implies (x, T_j^o(x)) \in Q.$$

That is, the T_i's map inputs to from Q to R, and the T_j^o's maps solutions to R back to solutions to Q. The *depth* d of the reduction is the maximum depth of any of the decision trees involved, and the *size* is n.The *complexity* of the reduction is $\log n + d$ and the complexity of reducing Q to R, denoted $R^{dt}(Q)$, is the minimum complexity of any decision tree reduction from Q to R. The TFNP^{dt} *subclass* associated with R, denoted R^{dt}, is all $Q \in \mathsf{TFNP}^{dt}$ such that $R^{dt}(Q) = \mathsf{polylog}(n)$.

TFNP^{dt} is intimately connected to proof complexity via the following claim from [24, 25].

Claim. Let $R \in \{0,1\}^n \times \mathcal{O}$ be any search problem in TFNP^{dt}. Then there exists an unsatisfiable CNF formula F on $|\mathcal{O}|$-many variables such that R is equivalent to S_F.

Proof. As $R \in \mathsf{TFNP}^{dt}$ there are $\mathsf{polylog}(n)$-depth decision trees $\{T_i\}_{i \in \mathcal{O}}$ which verify R. Define a *canonical CNF formula* associated with R to be

$$F := \bigwedge_{i \in \mathcal{O}} \neg T_i^o,$$

where we have abused notation and associated T_i^o with the DNF obtained by taking a disjunction over the (conjunction of the literals along) the *accepting* paths in T_i^o. This makes a $\neg T_i^o$ a CNF formula expressing that T_i^0 outputs 0. It is not difficult to check that a solution to S_F is equivalent to a solution to R. □

The upshot is that black-box TFNP is exactly the study of the false clause search problem! Thus, it suffices to study the search problems for the canonical CNF formulas S_F associated with $R \in \mathsf{TFNP}^{dt}$ instead of R itself. Furthermore, note that for any pair of decision trees $\{T_i^o\}$ and $\{T_i'^o\}$ that verify some $R \in \mathsf{TFNP}^{dt}$, the resulting false clause search problems S_F and $\mathsf{S}_{F'}$ are $\mathsf{polylog}(n)$-reducible.

Using this connection to the false clause search problem, Göös et al. [25] observed that many important proof systems are characterized by associated TFNP^{dt} problems in the sense that the CNF formulas F that are efficiently provable in that proof system are exactly the problems S_F that are efficiently reducible to that TFNP^{dt} problem.

TFNP Characterizations of Proof Systems. The known characterizations of proof systems by TFNP^{dt} problems are all in terms of a somewhat non-standard *complexity parameter*. For a proof system P and any unsatisfiable CNF formula F let

$$P(F) := \min\{\mathsf{deg}(\Pi) + \log \mathsf{size}(\Pi) : \Pi \text{ is a } P\text{-proof of } F\},$$

where deg denotes an associated *degree* measure of the proof system. For Nullstellensatz and Sherali-Adams, this degree measure is the maximum degree of any polynomial in their proofs, while for Resolution, degree is the proof width. While nonstandard, this complexity parameter is very natural. Indeed, all of the query-to-communication lifting theorems referenced in the introduction lift lower bounds on a complexity parameter for some proof system to lower bounds on some monotone circuit model.

We say that a TFNP^{dt} problem R *characterizes* a proof system P if $R^{dt} = \{\mathsf{S}_F : P(F) = \mathsf{polylog}(n)\}$; in this case, we also say that P characterizes R. In fact, all of the known characterizations hold in a stronger sense; let P be any of the proof systems listed above, and R be the canonical complete problem for its corresponding TFNP^{dt} class, then it holds that for any unsatisfiable CNF formula F,

$$P(F) = \Theta(R^{dt}(\mathsf{S}_F)).$$

While we have a number of characterizations, there remain many important proof systems for which no characterization is known. As well, there are prominent TFNP^{dt} sub-classes — such as PPP^{dt} — for which no characterizing proof system is known. In order to state our main result (Theorem 1) we must formally define a reflection principle.

What is a Reflection Principle?

The second condition of Theorem 1 is that the proof system must be able to prove its own *soundness*. A *reflection principle* Ref_P for a proof system P states that P-proofs are sound; it says that if Π is a P-proof of a CNF formula H then H must be unsatisfiable. The variables of a reflection principle specify a CNF formula H, a P-proof Π, and a truth assignment α to H. The formula states that Π is indeed a P-proof of H and α satisfies H,

$$\mathsf{Proof}_P(H, \Pi) \implies \neg\mathsf{Sat}(H, \alpha).$$

We require that the reflection principle that the proof system proves is efficiently verifiable. We say that a reflection principle is *efficiently verifiable* if it is encoded as a low-width CNF formula, and thus any solution to the false clause search problem for the reflection principle (also known as the *wrong proof problem* [4,23]) can be efficiently verified.

For a proof system P, there are many ways to encode its proofs, with the choice of the encoding potentially affecting the complexity of proving the associated reflection principle. Rather than worrying about

the particular encoding, we will instead define one reflection principle for each efficiently verifiable way of encoding P-proofs, which we will call a *verification procedure*. Recall that the complexity c of a proof is always an upper bound on the width of the CNF being proven. For this reason, and to simplify notation, we will allow the formula being proven to have width up to c.

Verification Procedure. A verification procedure V for a proof system P is a mapping of tuples (n, m, c) to CNF formulas that generically encodes complexity-c (or $O(c)$) P-proofs of n-variate CNF formulas with m clauses of width at most c. Specifically, the CNF formula $V_{n,m,c}$ has three sets of variables x, H, Π, such that:

- An assignment to the variables $H := \{C_{i,j} : i \in [m], j \in [c]\}$ specifies a CNF formula with m clauses over n variables, where $C_{i,j} \in [2n]$ is the index of the j-th literal of the i-th clause of H; if $C_{i,j} \leq n$ then it specifies a positive literal, and otherwise it specifies a negative literal.
- An assignment to the variables Π specifies a specific (purported) P-proof of H, such that any error in Π can be verified by looking at the assignment to at most poly-logarithmically many variables of $V_{n,m,c}$.
- The CNF formula $V_{n,m,c}$ has $2^{\Theta(c)}$ many variables.

As the complexity parameter c measures the logarithm of the size, and by (3), the number of variables is exponential in $\Theta(c)$, the second condition ensures that $V_{n,m,c}$ is verifiable by looking at polynomial-in-c many variables. In particular, this implies that the width of the CNF formula $V_{n,m,c}$ is $\mathsf{poly}(c)$ The third condition can be relaxed, and larger numbers of variables can be tolerated at the cost of worse bounds in Theorem 6. We give a concrete example of a verification procedure for the Polynomial Calculus proof system in Section 2.3.

For concreteness, we have fixed a particular encoding of H in order to avoid pathological codings; e.g., ones in which use a SAT oracle to decide whether the formula is satisfiable encoded by H. Since we allow arbitrary codings of proofs, this will be robust under different encodings of CNFs as long as they are polynomial-time computable from ours.

We can now define a reflection principle for any proof system and verification procedure for that proof system.

Reflection Principle. Let P be a proof system and V be a verification procedure for P-proofs. The reflection principle $\mathsf{Ref}_{P,V}$ associated with (P, V) is the unsatisfiable formula

$$\mathsf{Proof}_{n_H,m_H,c}(H, \Pi) \wedge \mathsf{Sat}_{n_H,m_H,c}(H, \alpha),$$

where H is a CNF formula over n_H variables with m_H clauses of width at most c. The j-th literal (if any) of the i-th clause of H is specified by a vector $C_{i,j}$ of $\log(2n_H + 1)$ many Boolean variables, and

- $\mathsf{Proof}_{n_H,m_H,c}(H, \Pi) := V_{n_H,m_H,c}(H, \Pi)$.
- $\mathsf{Sat}_{n_H,m_H,(d,n_F)}(H, \alpha)$ is the CNF formula stating that α is a satisfying assignment for H. This is expressed as,

$$\forall i \in [m_H], \exists j \in [c] \Big[\big([\![C_{i,j} = x_k]\!] \wedge \alpha_k \big) \vee \big([\![C_{i,j} = \neg x_k]\!] \wedge \neg \alpha_k \big) \Big],$$

  which can be encoded as a CNF formula of width $O(c \log n_H)$ and size $m_H \exp(O(c \log n_H))$.

For simplicity of notation, we will drop the subscripts P, V from Ref when the proof system and verification procedures are clear. One technicality is that TFNP^{dt} problems have one instance for each number of variables n; to ensure that this is the case for Ref we could use a pairing function on the multiple sets of variables for Ref, however we are going to ignore this detail. Each reflection principle gives rise to a TFNP^{dt} problem. Indeed, by construction Ref is verifiable by observing $\mathsf{polylog}(n)$ many bits, where n is the total number of variables.

We are now ready to prove Theorem 1 which we split Theorem 4 and Theorem 6 and prove in the following two sections.

2.1 A Proof System for any TFNP Problem

We next describe how any TFNP^{dt} problem R can be transformed into a proof system for refuting unsatisfiable CNF formulas of polylog width. A proof Π in this proof system, of the (unsatisfiability) of a CNF formula H, will consist of a low-depth decision reduction from H to the false clause search problem S_F for the unsatisfiable formula F associated with the TFNP problem R. For this, we first define a notion of reduction between CNF formulas.

Suppose C is a clause over n variables, and $T = \{T_i\}_{i\in[n]}$ is a sequence of depth-d decision trees, where $T_i : \{0,1\}^s \to \{0,1\}$. We write $C(T)$ to denote the CNF formula obtained by substituting the decision trees T_i for each of the variablse x_i in C and rewriting the result as a CNF formula. Formally, $C(T)$ is formed by creating the a stacked decision tree T^C that sequentially runs the decision trees T_i for each variable x_i used in C. A leaf of T^C is labelled with a 1 if the root-to-leaf path causes the trees T_i to output a satifying assignment for C; the other leaves are labelled with 0. Then $C(T)$ is the CNF

$$C(T) \;:=\; \bigwedge\{\neg p : p \text{ is a rejecting path of } T\},$$

where a path p is identified with the conjunction of the literals set true along the path, and $\neg p$ is its negation.

Reductions Between CNF Formulas. Next, we define what is means to reduce one false clause search problem to another. We say that a CNF formula H on n_H variables and m_H clauses *reduces* to an unsatisfiable $F = C_1 \wedge \cdots \wedge C_m$ over n variables via depth-d decision trees if there exist depth-d decision trees $T = \{T_i\}_{i\in n}$ where $T_i : \{0,1\}^{n_H} \to \{0,1\}$, and $\{T_i^o\}_{i\in[m]}$ with $T_i^o : \{0,1\}^{n_H} \to [m_H]$ so that the following conditions hold. Let F_H be the CNF formula

$$F_H := \bigwedge_{i\in[m]} \bigwedge_{p\in T_i^o} C_i(T) \vee \neg p,$$

where p ranges over all paths of T_i^o. Since $C_i(T)$ is a CNF, F_H is readily written as a CNF by distributing $\neg p$ into $C_i(T)$. Then each clause $C_i(T) \vee \neg p$ must either be tautological (contains a literal and its negation) or be a weakening of the clause of H indexed by the label at the end of the path p.

Observe that a depth-d decision tree reduction of S_H to S_F introduces a new false clause search problem S_{F_H} that is directly a refinement of H. Clearly, if F is unsatisfiable, then so is F_H and consequently also H is unsatisfiable.

Canonical Proof System. Let $\mathsf{S}_F \in \mathsf{TFNP}^{dt}$. The canonical proof system P_F for S_F proves an unsatisfiable CNF formula H on n_H variables if H is reducible to an instance of F on some n variables. A P_F-proof Π consists of the decision trees $T = \{T_i\}_{i\in[n]}$ and $T^0 = \{T_i^o\}_{i\in[m]}$. The *size* of a P_F-proof is the number

of variables n of the instance of F, and the *depth* is the maximum depth among the decision trees. The *complexity* of a proof of an unsatisfiable CNF formula is defined as

$$P_F(H) \;:=\; \min\{\mathsf{depth}(\Pi) + \log \mathsf{size}(\Pi) : \Pi \text{ is a } P_F\text{-proof of } H\}.$$

This proof system is sound as any substitution of an unsatisfiable CNF formula is also unsatisfiable. To see that it is efficiently verifiable, observe that it suffices to form the CNF F_H from F and the decision trees T_i and T_i^0, and check that each of the clauses of F_H is either tautological or is a weakening of a clause in H. This can be done in polynomial-time in the size of the proof.

The next theorem states that P_F has a short proof of H iff S_H efficiently reduces to S_F. This is almost immediate from the definitions.

Theorem 4. *Let $\mathsf{S}_F \in \mathsf{TFNP}^{dt}$ and H be an unsatisfiable CNF formula. Then,*

(a) If H has a size s and depth d proof in P_F, then S_H has a depth d and size $O(s)$ reduction to S_F.

(b) If S_H has a size s and depth d reduction to S_F, then H has a size $s2^{O(d)}$ and depth d proof in P_F.

In particular, $\mathsf{S}_F^{dt}(S_H) = \Theta(P_F(H))$.

Proof. To prove (b), suppose $T_1, \ldots, T_n$ and $T_1^o, \ldots, T_m^o$ is a size-s and depth-d decision-tree reduction from S_H to S_F. Construct F_H as above using these decision trees. Let L be a clause of $C_i(T)$ for some $i \in [m]$ and let p be a path in T_i^o. If $C_i(T) \vee \neg p$ is tautological, then we are done. Otherwise, we will argue that it is a weakening of a clause of H. Fix any assignment x which falsifies $L \vee \neg p$, then C_i is falsified by the assignment $T_1(x), \ldots, T_n(x)$ and $T_i^o(x)$ follows path p. Thus, by the correctness of the reduction, whenever $L \vee \neg p$ is false, the $T_i^o(x)$-th clause of $\neg H$ must also be false, and so $L \vee \neg p$ is a weakening of this clause. Each decision tree in the proof has depth at most d and therefore the size is at most $s2^{O(d)}$.

To prove (a), let $n, T_1, \ldots, T_n, T_1^o, \ldots, T_m^o$ be a P_F proof of H of size s and depth d. We claim that this is also a reduction from S_H and S_F. Indeed, fix any assignment x such that $T_1, \ldots, T_n(x)$ falsifies clause C_i of F and the decision tree $T_i^o(x)$ follows some path p. Then, a clause of the formula $C_i(T) \vee \neg p$ is falsified under x, and furthermore that clause is a weakening of the $T_i^o(x)$-th clause of H. Thus, $(x, T_i^o(x)) \in \mathsf{S}_H$. This reduction has depth d and size $n = O(s)$. □

The Canonical Proof System Proves its own Soundness

In this section we define a natural formulation of the reflection principle for the proof system P_F for any TFNP^{dt} problem S_F by way of defining a verification procedure for P_F. We show that the canonical proof system can prove this encoding of the reflection principle. To do so, we require the notion of a prototype of a decision tree.

A *proto-decision* tree of depth d over variables $\alpha_1, \ldots, \alpha_n$ and with output in $\mathcal{O}$ is a complete binary tree in which the label of every internal vertex v is given by a vector of $\log n$ of variables x_v whose value specifies the index of some variable α_i, and such that one child of v is labelled 0 and the other is labelled 1. Each leaf l is labelled with $\log |\mathcal{O}|$ variables x_l. For a given truth assignment to the variables x_v for all internal vertices c, the proto-decision tree induces a decision tree that queries that variables $\alpha_1, \ldots, \alpha_n$ as specified by the values of all of the x_v's. Specifically, for a given internal vertex v, the truth values assigned to the vector x_v at v in the proto-decision tree determines a value i so that α_i is queried at the corresponding vertex of the induced decision tree. Similarly, for a leaf l, the values of the variables x_l specify an $j \in \mathcal{O}$ which is the label for the corresponding leaf in the induced decision tree.

The decision tree *simulating* a proto-decision tree $\hat{T}$ is obtained from $\hat{T}$ as follows: Replace each internal vertex v of $\hat{T}$ by a complete binary tree querying the variables of x_v, and at each leaf where $x_v = i$, queries α_i. The leaves l of the proto-decision tree are replaced with complete binary trees querying x_l in which each leaf where $x_l = j$ is labelled by the output $j \in \mathcal{O}$.

Verification Procedure for P_F. Let $\mathsf{S}_F \in \mathsf{TFNP}^{dt}$. We define a verification procedure $V^P_{n_H,m_H,(d,n_F)}$ for P_F, which encodes a complexity $c = (d + \log n_F)$ P-proof Π of a CNF formula H on n_H variables and m_H clauses as follows. Π is specified by n_F depth-d proto-decision trees $\hat{T}_1, \ldots, \hat{T}_{n_F}$ with output in $\{0, 1\}$ and m_F depth-d proto-decision trees $\hat{T}^o_1, \ldots, \hat{T}^o_{m_F}$ with output in $[m_H]$. The constraints of Proof enforce that each clause of the reduced CNF formula F_H is a weakening of a clause of H. For each $i \in [n_F]$, let S_i be the decision tree simulating $\hat{T}_i$ but eliminating the queries to the variables α_i.[†] Recall that the assignment of truth values to the vector of variables x_v at a vertex v determines the index $i \in [n_H]$ of the variable being queried at v in the decision tree. Let $z_k \in [n_F]$ denote the k-th variable of F.

For each clause C_i in F, consider the following binary decision tree T_{C_i}: First, it runs the decision trees S_k for every $k \in [n_F]$ such that C_i involves z_k: this determines the literals which occur in one of clauses of F_H, namely in one of the clauses that is formed by applying the decision trees $\hat{T}_i$ to the clause C_i. We temporarily use C' to denote this clause of F_H. Note that C' involves variables of H; however, the truth values (the α_i values) of the variables in C' have not been queried and are instead treated in the next phase as being set to the values that falsify C'. Second, it runs the decision tree simulating $\hat{T}_i$. A vertex of $\hat{T}_i$ labelled with an x_v is handled by querying the variables x_v. The results of the queries to x_v specify a variable α_i. The variable α_i may appear in C' and if so is treated as having the value that falsifies C'. If, however, the variable α_i does not appear in C', then it is non-deterministically queried; that is, the tree T_{C_i} branches to try both 0 and 1 as truth values for α_i. The result of running the decision tree simulating $\hat{T}_i$ is a value ℓ specifying a clause of H. Third, it queries the vector of variables $C_{\ell,j}$ for $j \in [c]$: this determines the literals of the ℓ-th clause of H. If a path in this decision tree determines that the clause C' of F_H is not a weakening of the ℓ-th clause of H, then the path is called "bad".

The CNF formula $\mathsf{Proof}_{n_H,m_H,(d,n_F)}(H, \Pi)$ is $\bigwedge_{\text{bad } p} \neg p$, expressing that there is no bad path. It thus is satisfied only when the Π is a valid $P(F)$-proof of H.

As each proto-decision tree has depth at most d, F has width at most $\mathsf{polylog}(n_F)$, and H has width at most c, the resulting CNF formula has width $d\mathsf{polylog}(n_F) + \log m_H + c \log n_H$.

Canonical Reflection Principle. Let $\mathsf{S}_F \in \mathsf{TFNP}^{dt}$. We define its canonical reflection principle Ref_F to be the conjunction

$$\mathsf{Proof}_{n_H,m_H,(d,n_F)}(H, \Pi) \wedge \mathsf{Sat}_{n_H,m_H,(d,n_F)}(H, \alpha),$$

where Sat is defined as in the definition of the reflection principle and $\mathsf{Proof} := V^P_{n_H,m_H,(d,n_F)}$. In total, this is a CNF formula of width $d \log n_F + \log m_H + c \log n_H$ over $n = m_F 2^{d+1} + n_F 2^d \log n_H + c m_H \log 2n_H$ many variables. In particular, under any assignment to the variables, any clause of Ref_F can be evaluated by looking at the values of $\mathsf{polylog}(n)$ many variables, where n is number of variables of Ref, and so $\mathsf{S}_{\mathsf{Ref}_F} \in \mathsf{TFNP}^{dt}$.

Theorem 5. *For any $\mathsf{S}_F \in \mathsf{TFNP}^{dt}$, $P_F(\mathsf{Ref}_F) \leq \mathsf{polylog}(n)$.*

Proof. Fix an instance of $\mathsf{S}_{\mathsf{Ref}_F}$. By Theorem 4, it suffices to show that $\mathsf{S}_{\mathsf{Ref}_F}$ is reducible to an instance of S_F. Let the instance of Ref_F be specified with parameters $(n_H, m_H, (d, n_F))$, letting $c = d + \log n_F$.

[†] $\mathsf{Proof}_{n_H,m_H,(d,n_F)}(H, \Pi)$ does not involve the variables α_i.

For each proto-decision tree $\hat{T}_i$ of Ref_F, let S_i be the decision tree that simulates it. As well, let S_i^o be the decision tree that simulates $\hat{T}_i^o$.

We will define the decision trees of the reduction $T_1, \ldots, T_{n_F}, T_1^o, \ldots, T_{m_F}^o$ from $\mathsf{S}_{\mathsf{Ref}_F}$ to an instance of S_F on n_F variables. Define $T_i := S_i$, and let T_i^o be the decision tree implementing the following algorithm which takes as input $x \in \{0,1\}^n$ and outputs a falsified clause of $\mathsf{Ref}_F(x)$ provided that the assignment $T_1(x), \ldots, T_{n_F}(x)$ falsifies clause the clause C_i of F. First, the algorithm runs the decision trees T_i for each $i \in \mathsf{vars}(C_i)$, and then it runs the decision tree for S_i^o.

Let x^* be the restriction of x to the variables queried thus far in the algorithm. As $(T_1(x^*), \ldots T_{n_F}(x^*))$ falsifies C_i, there must be a clause of F_H falsified by x^*. This clause should be a weakening of $T_i^o(x^*)$-th clause of H. To check whether this is indeed the case, we ask for the indices of the variables that occur in the $T_i^o(x^*)$-th clause of H — this requires us to query at most $c \log n_H$ many variables. If our clause is indeed a weakening of the $T_i^o(x^*)$-th clause of H, then our x^* must falsify the $T_i^o(x^*)$-th clause of H, violating a constraint of SAT, and so our algorithm will output the index of this violated clause SAT. Otherwise, if this is not the case, then x^* must falsify a clause of Proof, and so we can output the index of this violated clause.

To convert this algorithm into a decision tree we must label the leaves which are the terminals of paths which are not followed in any run of this algorithm. For a path not to be followed by this algorithm, it must either correspond to a partial assignment x^* such that $(T_1(x^*), \ldots, T_{n_F}(x^*))$ satisfies C_i, and therefore the output at that leaf can be arbitrary. As H has width at most c and F has width $\mathsf{polylog}(n_F)$, the depth d^* of the resulting decision tree is $d^* = O(c(d \log n_H + \log m_H)) + \mathsf{polylog}(n_F)$ and the number of variables is n_F; thus the complexity of the reduction is $d^* + \log n_F$, which is poly-logarithmic in n, the number of variables of Ref. □

2.2 TFNP Problems for Proof systems which Prove their own Soundness

In this section we identify the necessary conditions for a proof system to be characterized by a TFNP^{dt} problem. The first necessary condition is that the proof system must be closed under *decision-tree reductions*, as TFNP^{dt} classes are closed under decision tree reductions.

Closure under Decision Tree Reductions. A proof system P is closed under decision tree reductions if whenever there is a P-proof of complexity c of an unsatisfiable formula F, and H reduces to F by depth-d decision trees, then there is a P-proof of H of complexity $O(cd)$.

Note that the canonical proof system is closed under decision tree reductions. In all proof systems which are known to admit characterization by a TFNP^{dt} problem, closure under decision tree reductions takes the form of directly substituting (an appropriate encoding of) decision trees into the proofs, resulting in a proof of complexity $O(cd)$. For example, if H reduces to F and we have a Resolution proof of F, then we can obtain a Resolution proof of H by replacing each variable in the proof of F by the (DNF formula corresponding to the accepting paths of) corresponding decision tree from the reduction.

The second condition is that the proof system must be able to prove its own *soundness*. That is, it must be admit short proofs of a reflection principle about itself. As we will show, any verification procedure for its proofs will do.

Theorem 6. *Let P be a proof system that is closed under decision tree reductions and let V be a verification procedure for P. For any unsatisfiable CNF formula H,*

i) $S^{dt}_{\mathsf{Ref}}(\mathsf{S}_H) \in O(P(H))$.

ii) $P(H) \in O(\mathsf{S}^{dt}_{\mathsf{Ref}}(\mathsf{S}_H)P(\mathsf{Ref}))$.

In particular, if P *has* $\mathsf{polylog}(n)$*-complexity proofs of* Ref *then* P *is* characterized *by* $\mathsf{S}_{\mathsf{Ref}}$.

First, we give a high-level sketch of the proof. Let P be a proof system that satisfying (i) and (ii). Observe that $\mathsf{S}_{\mathsf{Ref}} \in \mathsf{TFNP}^{dt}$ as Ref is efficiently verifiable, and let $\mathsf{S}_H \in \mathsf{TFNP}^{dt}$ for an unsatisfiable CNF formula H, such that $\mathsf{S}^{dt}_{\mathsf{Ref}}(R) = \mathsf{polylog}(n)$. Then, there is a decision-tree reduction from S_H to $\mathsf{S}_{\mathsf{Ref}}$ and, as P is closed under decision tree reductions and there is a $O(\mathsf{polylog}n)$-complexity P-proof of Ref_P, there must also be an efficient P-proof of H. Conversely, suppose that Π is a $\mathsf{polylog}(n)$-complexity P-proof of an unsatisfiable CNF formula H, then we can construct a reduction from S_H to $\mathsf{S}_{\mathsf{Ref}}$ by hard-wiring H and Π into $\mathsf{S}_{\mathsf{Ref}}$, leaving the only truth assignment free. On any input α to S_H, the hard-wired instance of $\mathsf{S}_{\mathsf{Ref}}$ must output a falsified clause of H, as Π is a valid P-proof of H.

Proof of Theorem 6. Let H be any unsatisfiable CNF formula and recall that $\mathsf{S}^{dt}_{\mathsf{Ref}}(\mathsf{S}_H)$ denotes the complexity of reducing S_H to S_{Ref}. As P is closed under decision tree reductions, this implies that there is a P-proof of H with complexity $P(H) = O(\mathsf{S}^{dt}_{\mathsf{Ref}}(\mathsf{S}_H)P(\mathsf{Ref}))$.

Conversely, suppose that Π is a complexity $c := P(H)$ proof in P of an unsatisfiable CNF formula H. Then, we can construct a reduction from S_H to an instance of $\mathsf{S}_{\mathsf{Ref}}$ as follows. Let n_H, m_H be the number of variables and number of clauses respectively. The reduction $T = (T_1, \ldots, T_n)$ hardwires the input (H, Π) into the instance of $\mathsf{S}_{\mathsf{Ref}}$ with variables n_H, m_H, c, using constant decision trees, leaving only α unspecified. Next, we argue that this reduction is correct. Let $\alpha \in \{0,1\}^{n_H}$ be any assignment to S_H then, as Π is a valid P-proof of H, the only outputs of $\mathsf{S}_{\mathsf{Ref}}(T(\alpha))$ are clauses of H which are falsified under α. As the number of variables of the instance of Ref is exponential in $\Theta(c)$, and the decision trees T are constant, $\mathsf{S}^{dt}_{\mathsf{Ref}}(\mathsf{S}_H) = O(P(H))$. □

2.3 Example: The Polynomial Calculus

As an example, we give a characterization of the Polynomial Calculus by a natural TFNP^{dt} problem and show that it can prove a reflection principle about itself, proving Theorem 2. This answers an open question from [24], asking for a characterization of the Polynomial Calculus. We note that the definition of this TFNP^{dt} problem can be adapted in a straightforward way to characterize $\mathbb{F}_q$-Polynomial Calculus, Polynomial Calculus over the integers with coefficients measured in unary, and dag-like Sherali-Adams.

The Polynomial Calculus (PC). The $\mathbb{F}_2$-Polynomial Calculus proves that an unsatisfiable system of $\mathbb{F}_2$-polynomial equations $\{p_i(x) = 0\}_{i\in[m]}$ has no solutions over $\{0,1\}$. An unsatisfiable CNF formula $F = C_1 \wedge \ldots \wedge C_m$ is encoded as such a system of equations by mapping each clause to a linear equation $\overline{C}_i$ such that $C_i(x) = 1$ iff $\overline{C}_i(x) = 0$. We will exclusively be operating with multilinear arithmetic (that is, the degree of any polynomial) — for example, x_i^2 and x_i are represented by the same function. Formally, we operate modulo the ideal $\langle x_i^2 = x_i \rangle_{i\in[n]}$.

A $\mathbb{F}_2$-PC proof of $\{a_i x = 0\}$ is a derivation of the trivially false polynomial $1 = 0$ from $\{a_i x = 0\}_{i\in[m]}$ by the following two rules:

Addition. From two previously derived polynomials p, q, deduce $p + q$.

Multiplication. From a previously derived polynomial p, deduce $x_i p$ for some $i \in [n]$.

The *size* of a proof is the number of monomials (with multiplicity) in the proof, the *length* is the number of lines (applications of rules), and the *degree* is the maximum degree of any polynomial at any step in the proof. The *complexity* of proving an unsatisfiable CNF formula F in $\mathbb{F}_2$-PC is

$$\min\{\mathsf{size}(\Pi) + \log \mathsf{degree}(\Pi) : \mathbb{F}_2\text{-PC proofs } \Pi \text{ of } F\}$$

We call the TFNP^{dt} problem which characterizes $\mathbb{F}_2$-PC $\mathsf{IND}_{\mathsf{PPA}}$ as it resembles a strong induction over a PPA-complete problem. Before stating this problem formally, we will describe it at a high level. An instance of $\mathsf{IND}_{\mathsf{PPA}}$ is given by a set of N nodes and a set of L pools which are (possibly overlapping) subsets of the N nodes. There is a distinguished node which we will call 1, and the L-th pool contains only the 1 vertex. These pools are arranged in a directed acyclic graph (dag), and for each pool we have a matching on the nodes that occur (with multiplicity) in that pool and its immediate children, such that only nodes of the same type are matched to each other; i.e., if $x \in N$ occurs in this set then it must be matched to another copy of x. Since the L-th pool contains only a single node, there must be some pool with an unmatched vertex. A solution is an unmatched or mismatched node. We remark that this dag is specified by the input to the problem. This is crucial; if the graph was fixed in advance, then this problem would be in PPA, and thus gives rise to a Nullstellensatz proof — if $P_{\ell'}^{(\ell)}$ (using the notation below) was fixed then the resulting problem would be PPA-complete.

Strong Induction PPA. The $\mathsf{IND}_{\mathsf{PPA}}$ problem is defined as follows

- *Structure.* $[L]$ *pools* and $[N]$ *nodes*. We think of each $\ell \in [L]$ as being associated with its own copy of $[N]$.
- *Variables.* For each $\ell \in L$ and $\ell' < \ell$ we have $P_{\ell'}^{(\ell)} \in \{0,1\}$ indicating whether ℓ' is an immediate *predecessor* of pool ℓ. For each pool $\ell \in [L]$ and node $m \in [N]$, we have a variable $A_m^{(\ell)} \in \{0,1\}$ indicating whether node m is *active* at pool ℓ. Finally, we have a *matching* between $\ell \in [\ell]$ and all of its predecessors. For each $\ell' < \ell$ and $m \in [N]$ we have a variable $M_{\ell',m'}^{(\ell)} \in [\ell] \times [N]$ indicating to which node the node m' of ℓ''s copy of $[N]$ is matched to. The *root* pool L has $A_1^{(L)} = 1$ and $A_m^{(L)} = 0$ for all $m \neq 1$.
- *Solutions.* Since the root has an odd number of active nodes, and each matching is even, there must be some pool $\ell \in [L]$ with an erroneous matching. A solution is any triple $(\ell, \ell', m) \in [L]^2 \times [N]$ such that ℓ' is a predecessor of ℓ and is an active node for ℓ', and m is matched to a node m' which is not matched to m. That is, $P_{\ell'}^{(\ell)} = 1$, $A_m^{(\ell)} = 1$, $M_{(\ell',m)}^{(\ell)} = (\ell'', m')$, and either $P_{\ell''}^{(\ell)} = 0$, $A_{m'}^{(\ell'')} = 0$, or $M_{\ell'',m'}^{(\ell)} \neq (\ell', m)$.

Observe that $\mathsf{IND}_{\mathsf{PPA}} \in \mathsf{TFNP}^{dt}$ as a candidate solution can be verified in by looking at only a $O(\log n)$ many variables.

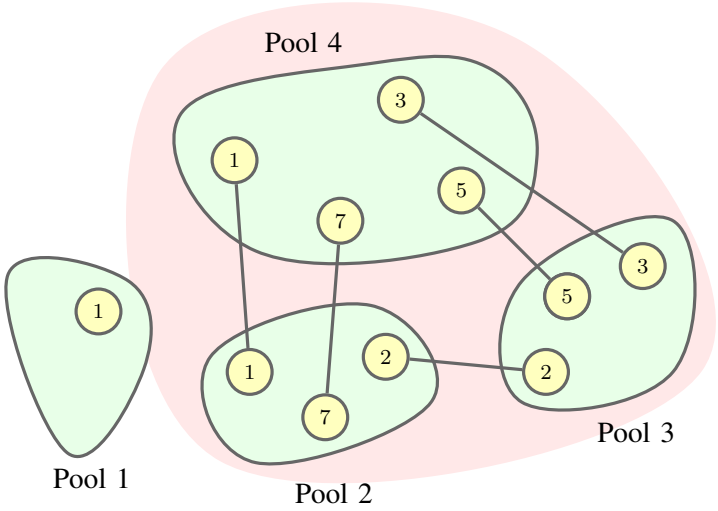

Figure 1: An example matching for Pool 4. The pink area indicates the active predecessors of Pool 4, i.e., $P_1^{(4)} = 0$ and $P_2^{(4)} = P_3^{(4)} = 1$. The yellow circles indicate the active nodes for that pool; for example Pool 1 has only node 1 active: $A_1^{(1)} = 1$, while $A_m^{(1)} = 0$ for all $m \neq 1$. Copies of the same node within the pink area are matched; e.g., $M_{2,2}^{(4)} = (3, 2)$ and $M_{3,2}^{(4)} = (2, 2)$.

Theorem 7. *For any unsatisfiable CNF formula F,*

- *If S_F has a complexity-c reduction to an instance of $\mathsf{IND}_{\mathsf{PPA}}$, then there is a degree-$O(d)$ and size $2^{O(c)}$ $\mathbb{F}_2$-PC proof of F.*
- *If F has a size-s and degree-d $\mathbb{F}_2$-$\mathsf{PC}(F)$ proof, then there is a complexity $O(d + \log s)$ reduction from S_F to $\mathsf{IND}_{\mathsf{PPA}}$.*

In particular, $\mathsf{IND}_{\mathsf{PPA}}^{dt}(\mathsf{S}_F) = \Theta(\mathbb{F}_2\text{-}\mathsf{PC}(F))$

Note that an analogous statement holds for the $\mathbb{F}_2$-PCR proof system, which builds on $\mathbb{F}_2$-PC to include additional variables $\overline{x}_i$ for each $i \in [n]$ to represent $\neg x_i$, along with the additional axioms $x_i + \overline{x}_i = 0$. Indeed, this is just a difference of the encoding of the CNF formula F as a set of linear equations and does not affect the proof or the resulting TFNP^{dt} problem!

We break the proof of this theorem into two lemmas, Lemma 8 and Lemma 9.

Lemma 8. *Let F be an unsatisfiable CNF formula. If S_F is reducible to an instance of $\mathsf{IND}_{\mathsf{PPA}}$ on n variables using decision trees of depth at most d then there is an $O(d)$-degree and size-$n^2 2^{O(d)}$ $\mathbb{F}_2$-Polynomial Calculus proof of F.*

Proof. Let F be an unsatisfiable CNF formula and suppose that it is reducible to an instance of $\mathsf{IND}_{\mathsf{PPA}}$ on n variables using decision trees of depth at most d. That is, each variable x of the $\mathsf{IND}_{\mathsf{PPA}}$ is computed by a depth-d decision tree T_x querying variables of F; for simplicity, we will abuse notation and associate each variable x with the polynomial formed by taking the sum over the (product of the literals on each of the) accepting paths of T_x. As well, let $\{T_i^o\}_i$ be the decision trees for each solution of the $\mathsf{IND}_{\mathsf{PPA}}$ instance.

By induction from $\ell = 1, \ldots, L$ we will derive the polynomial

$$q_\ell := \sum_{m \in [N]} A_m^{(\ell)} = 0$$

over $\mathbb{F}_2$, which roughly says that there is a perfect matching between the nodes in ℓ and the nodes in its predecessors. This will be sufficient to complete the proof as $A_1^{(L)} = 1$ and $A_m^{(L)} = 0$ for all $m \neq 1$, and so the decision trees for these variables are identically 1 and 0 respectively. Thus, we will have derived $0 = \sum_{m \in [N]} A_m^{(L)} = A_1^{(L)} = 1$.

Now, suppose that we have derived $q_{\ell'} = 0$ for all $\ell' < \ell$ with with a degree-$O(d)$ $\mathbb{F}_2$-PC proof; we show how to drive $q_\ell = 0$. At a high level, this follows from the fact that there is a perfect matching between the nodes of pool ℓ and all of its predecessors. For simplicity of exposition, will definitional an additional variable $P_\ell^{(\ell)} := 1$, whose decision tree is the constant 1 function.

Claim. There is a degree-$O(d)$ and size-$NL2^{O(d)}$ $\mathbb{F}_2$-Polynomial Calculus proof from F of the polynomial

$$\sum_{\ell' \leq \ell} P_{\ell'}^{(\ell)} \sum_{m \in [N]} A_m^{(\ell')} = 0.$$

This claim is sufficient to complete the proof. Indeed, we can use it in order to derive $q_\ell = 0$ from $q_{\ell'} = 0$ for $\ell' < \ell$ (which we have derived by induction) without significantly increasing the degree. To see this, multiply each $q_{\ell'}$ by $P_{\ell'}^{(\ell)}$ and sum them to obtain

$$\sum_{\ell < \ell'} P_{\ell'}^{(\ell)} q_{\ell'} = \sum_{\ell < \ell'} P_{\ell'}^{(\ell)} \sum_{m \in [N]} A_m^{(\ell')} = 0.$$

Adding this polynomial to $\sum_{\ell' \leq \ell} P_{\ell'}^{(\ell)} \sum_{m \in [N]} A_m^{\ell'} = 0$, which as a low-degree proof from F by the previous claim, gives $p_\ell = 0$. Note that as each $p_{\ell'}$ is a polynomial of degree-at-most-d, each of these polynomials has degree at most $2d$. Therefore, this inductive step requires degree $O(d)$ and size $LN2^{O(d)}$. □

Proof of Claim. For a polynomial p and outcome i let $[\![p = i]\!]$ denote the indicator polynomial which is 1 iff $p(x) = i$ and 0 otherwise. For $m \in [N]$ and $\ell' < \ell$ define the polynomial

$$\mathsf{match}_{m,\ell'}^{(\ell)} := \sum_{m^* \in [N], \ell^* \in [\ell]} \left[\!\!\left[M_{m,\ell'}^{(\ell)} = (m^*, \ell^*) \right]\!\!\right] P_{\ell^*}^{(\ell)} A_{m^*}^{(\ell^*)} \sum_{\hat{m} \in [N], \hat{\ell} \in [\ell]} \left[\!\!\left[M_{m^*,\ell^*}^{(\ell)} = (\hat{m}, \hat{\ell}) \right]\!\!\right],$$

which records whether the node m belonging to pool ℓ' is correctly matched in the matching for pool ℓ. We will break the terms of this polynomial into two sets, where C is the set of terms where the copy of m belonging to ℓ' is *correctly* matched, i.e., $P_{\ell^*}^{(\ell)} = 1$, $A_{m^*}^{(\ell^*)} = 1$ and $(\hat{m}, \hat{\ell}) = (m, \ell')$. Let E be the remaining set of terms, those where the copy of m belonging to ℓ' is erroneously matched. Using this polynomial, define

$$\mathsf{match}^{(\ell)} := \sum_{\ell' \in [\ell]} P_{\ell'}^{(\ell)} \sum_{m \in [N]} A_m^{(\ell')} \mathsf{match}_{m,\ell'}^{(\ell)}$$

which records the matching for pool ℓ. Let C, E be as above (now taken over both sums). Observe that each term in C occurs an even number of times, as (m, ℓ') is matched to (m^*, ℓ^*) iff (m^*, ℓ^*) is matched

to (m, ℓ'). Thus, $\sum_{t \in C} t = 0$. Now, consider a term $t \in E$. This term corresponds to a node m in some pool ℓ' being incorrect matched; let s be this incorrect matching and we will denote by t_s that t witnesses the incorrect matching s. Let T_s^o be the decision tree for solution s and abuse notation by identifying it with the polynomial obtained by summing over (the product of the literals on) each of its paths. Recalling that the result of summing over all paths in a decision tree is 1, we have

$$\mathsf{match}^{(\ell)} = \sum_{t^* \in C} t^* + \sum_{t_s \in E} t_s = 0 + \sum_{t_s \in E} t_s \cdot T_s.$$

An incorrect matching s is a solution to $\mathsf{IND}_{\mathsf{PPA}}$, and thus any truth assignment $x \in \{0,1\}^n$ which falsifies t_s must also falsify the $T_s(x)$-th clause C of F, as this instance of $\mathsf{IND}_{\mathsf{PPA}}$ solves S_F. It follows each term of $t_s \cdot T_s$ must contain the polynomial $\overline{C}$ for some clause C of F, and therefore $t_s \cdot T_s$ can be derived by multiplication from the axiom $\overline{C} = 0$. The degree of the proof is $7d$ and the size is $NL2^{O(d)}$. □

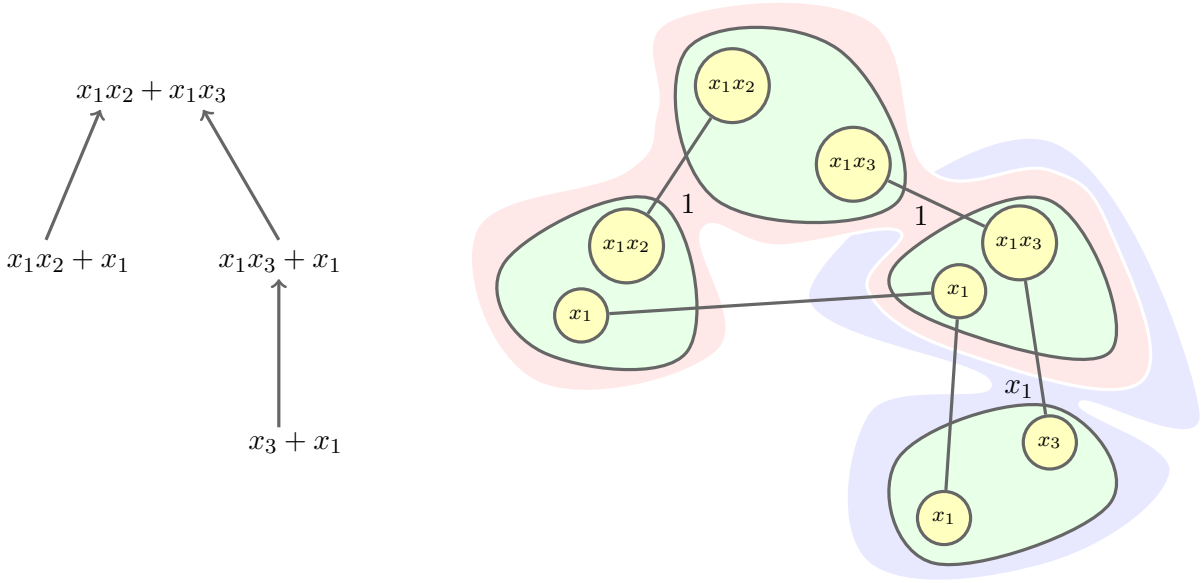

Figure 2: A $\mathsf{IND}_{\mathsf{PPA}}$ instance constructed from a Polynomial Calculus derivation. Left: a Polynomial Calculus derivation. Right: the corresponding $\mathsf{IND}_{\mathsf{PPA}}$ instance. The non-zero variable of the $\mathsf{IND}_{\mathsf{PPA}}$ is labelled with the variables that they query in their decision tree. The red area is represents the children of the pool corresponding to the line $x_1x_2 + x_1x_3$ (i.e., $P_2^{(4)} = P_3^{(4)} = 1$), while the blue area indicates the children of the line $x_1x_3 + x_1$ ($P_1^{(2)} = x_1$). The black lines indicate the matchings.

We now prove the converse of Theorem 7.

Lemma 9. *Let F be an unsatisfiable CNF formula on n variables. If there is a $\mathbb{F}_2$-Polynomial Calculus proof of F with size s, length L, and depth d then S_F is reducible by decision tress of depth $O(d)$ to an instance of $\mathsf{IND}_{\mathsf{PPA}}$ on $O(sL)$ variables.*

Note that this establishes the converse as the length of a $\mathbb{F}_2$-PC proof is always upper-bounded by the size.

Proof. Let N be the number of *distinct* monomials that appear in the $\mathbb{F}_2$-PC proof of F. We construct an instance of $\mathsf{IND}_{\mathsf{PPA}}$ over pools $[L]$ and nodes $[N]$. We abuse notation and associate each $\ell \in [L]$ with the ℓ-th line in the proof and each $m \in [N]$ with its corresponding monomial.

Fix some $\ell \in [L]$ and for each $m \in [N]$ occurring in line ℓ, define $A_m^{(\ell)}$ to be the depth-d decision tree which outputs 1 iff $m(x) = 1$. For the remaining m, set $A_m^{(\ell)} = 0$.

We set the predecessor variables as follows. If ℓ was derived by *addition* from ℓ', ℓ'', then set $P_{\ell'}^{(\ell)} = P_{\ell''}^{(\ell)} = 1$ and $P_{\ell^*}^{(\ell)}$ for all other ℓ^*. Otherwise, if ℓ was derived by *multiplication* by a variable x_i from ℓ', then set $P_{\ell'}^{(\ell)} = x_i$, and $P_{\ell^*}^{(\ell)} = 0$ for all $\ell^* \neq \ell'$. Finally, if ℓ was an initial clause of F, then we set $P_{\ell^*}^{(\ell)} = 0$ for all ℓ^*.

Next, we set the matching variables of each ℓ which does not correspond to an initial clause of F as follows. Observe that if ℓ was derived by addition from ℓ', ℓ'' then every monomial m in ℓ must occur in exactly one of ℓ', ℓ'' as otherwise it would have cancelled over $\mathbb{F}_2$. Thus, if ℓ' is the child of ℓ in which m also occurs then we set $M_{\ell',m}^{(\ell)} = (\ell, m)$ and $M_{\ell,m}^{(\ell)} = (\ell', m)$, matching those two occurrences of the m-th node. Otherwise, if m does not appear in ℓ, but is in one of the predecessors of ℓ, say ℓ', then it must also appear in ℓ''. In this case we set $M_{\ell',m}^{(\ell)} = (\ell'', m)$ and $M_{\ell'',m}^{(\ell)} = (\ell', m)$. Finally if m does not occur in any of these lines, then we set $M_{\ell^*,m}^{(\ell)}$ arbitrarily for $\ell^* \in \{\ell, \ell', \ell''\}$.

Otherwise, if ℓ was derived from ℓ' by multiplication with some variable x_i then we set the matching in a similar way as above. A monomial m occurs in ℓ if either m or $m \setminus x_i$ occurs in ℓ', but not both. For each $m \in [N]$, if m occurs in ℓ then we set $M_{\ell,m}^{(\ell)}$ match it to the m or $m \setminus x_i$ that occurs in ℓ', and set the matching variable for this node to match it back to (ℓ, m). Otherwise, if m and $m \setminus x_i$ occur in ℓ' then set $M_{\ell',m}^{(\ell)} = (\ell', m \setminus x_i)$ and $M_{\ell',m\setminus x_i}^{(\ell)} = (\ell', m)$. Finally, for match the m which do not occur in ℓ or ℓ' arbitrarily.

Finally, we set the matching variables of the $\ell \in L$ which correspond to an axiom $A \in \{\overline{C} : C \in F\}$ as follows. Each $M_{\ell,m}^{(\ell)}$ is defined by querying the variables in A (of which there are at most d by definition). If A is satisfied, then we fix an arbitrary matching between the monomials of A, and otherwise if A is falsified then we fix an arbitrary false matching (say, match each of the monomials in A in a cycle).

Observe that violations occur only in the matchings of $\ell \in [L]$ which correspond to clauses of F that are falsified. Thus, any solution to this instance of $\mathsf{IND}_{\mathsf{PPA}}$ will be a solution to S_F and we can define the output decision trees for these clauses as such. The output decision trees corresponding to other solutions can be set to output a fixed arbitrary solution as those solutions will never occur. □

The Polynomial Calculus Proves its own Soundness

Next, we state a reflection principle for the $\mathbb{F}_2$-Polynomial Calculus, using a natural verification procedure.

A Verification Procedure for $\mathbb{F}_2$-PC. We define the following verification procedure $V^{\mathsf{PC}}_{n_H,m_H,(d,s,L)}(H,\Pi)$ for $c = d + \log s + \log L$. For simplicity of description we have included a length parameter L, however as $L \leq s$, we could have used s instead and included additional variables to indicate which lines are active and which are not; this would only affect the complexity up to log-factors. For simplicity of notation, we will group the $\mathbb{F}_2$-PC rules into a single inference rule:

$$\frac{l_1 \quad l_2}{l_1 x + l_2 y}$$

for any $x, y \in \{0, 1, x_1, \ldots, x_n\}$.

Each line $\ell \in [L]$ has a list of s degree-d monomials $\mathsf{mon}_m^{(\ell)}$ for $m \in [s]$, where $\mathsf{mon}_{m,i}^{(\ell)} \in [n_H]$ for $i \in [d]$ specifies the i-th variable in m. The (n_h+1)-st value is understood to indicate the 1 polynomial. However, we do not want every line to have m monomials, and so we include a variable $a_m^{(\ell)} \in \{0,1\}$ which indicates whether the i-th monomial is *active* — actually occurs in ℓ. We reserve the first m_H lines $\ell \in [L]$ for the input clauses of H. Each line $\ell > m_H$ has two predecessor pointers $p_1^{(\ell)}, p_2^{(\ell)} \in [\ell - 1]$ indicating the lines from which ℓ was derived (if any), and $v_1^{(\ell)}, v_2^{(\ell)} \in [n_H + 2]$ indicating the the variables x, y that the lines indicated by $p_1^{(\ell)}, p_2^{(\ell)}$ were multiplied by in order to obtain ℓ; the final two values $n_H + 1, n_H + 2$ indicate the constants 0 and 1 respectively. Finally, to ensure that each inference is sound, for every line ℓ there is a matching between the monomials of ℓ and those of $\ell' < \ell$. We will require that each active monomial for ℓ is matched to an appropriate active monomial of its predecessors. The matching is given by variables $\mathsf{match}_{\ell',m'}^{(\ell)} \in \{0,1,2\} \times [s]$, where 0 indicates that m' is matched to a monomial in ℓ, 1 to a monomial in $p_1^{(\ell)}$ and 2 means that it is matched to a monomial in $p_2^{(\ell)}$. For the leaves $\ell \in [m_H]$ we enforce that there is a matching between its active monomials $\mathsf{match}_{\ell,m'}^{(\ell)} \in [s]$.

The constraints are as follows:

- *Initial Clauses.* We enforce that the first m_H lines are active, that the monomials of $\ell \in [m_H]$ are exactly the monomials of the ℓ-th clause of H, and that each active monomial is matched with another active monomial in ℓ.
- *Root.* To require that this is indeed a proof of H, we enforce that the root L of the proof has $a_1^{(L)} = 1$, $\mathsf{mon}_{1,i}^{(L)} = n_H + 1$ (i.e., is the constant 1 polynomial) for all $i \in [d]$, and $a_m^{(\ell)} = 0$ for all $m \neq 1$.
- *Inference.* To express the inference rule, we would like to state that if line $\ell > m_H$ was derived from lines $p_1^{(\ell)}, p_2^{(\ell)}$ with variables $v_1^{(\ell)}, v_2^{(\ell)}$, then the monomials of ℓ are exactly those in $v_1^{(\ell)} p_1^{(\ell)} + v_2^{(\ell)} p_2^{(\ell)}$ after cancelling $\mathrm{mod}2$. More concretely, that each active monomial in ℓ is matched to with appropriate active monomial in $p_1^{(\ell)}$ or $p_2^{(\ell)}$.

Define $\mathsf{Ref}^{\mathsf{PC}} := \mathsf{Sat} \wedge \mathsf{Proof}^{\mathsf{PC}}$ where $\mathsf{Proof}^{\mathsf{PC}} := V^{\mathsf{PC}}$. We show that $\mathbb{F}_2$-PC has short proofs of $\mathsf{Ref}^{\mathsf{PC}}$.

Theorem 10. $\mathsf{PC}(\mathsf{Ref}^{\mathsf{PC}}) \leq \mathsf{polylog}(n)$.

Proof. By Theorem 7 it suffices to construct a reduction from $\mathsf{Ref}^{\mathsf{PC}}$ to $\mathsf{IND}_{\mathsf{PPA}}$ Fix an instance of $\mathsf{Ref}^{\mathsf{PC}}$ with parameters $n_H, m_H, (d, s, L)$. We construct an instance of $\mathsf{IND}_{\mathsf{PPA}}$ with L pools and s nodes. The high-level idea of the proof is simple: we view $\mathsf{Ref}^{\mathsf{PC}}$ as $\mathsf{IND}_{\mathsf{PPA}}$, where each node for each line corresponds to a monomial which is encoded by $d \log n_H$ bits. We then arrange the matching in the $\mathsf{IND}_{\mathsf{PPA}}$ instance so that two nodes m, m' that are matched in $\mathsf{Ref}^{\mathsf{PC}}$ are matched in $\mathsf{IND}_{\mathsf{PPA}}$ if they were correctly derived — if m is derived from m' by multiplication by a variable x then we check that indeed $m = m'x$.

First, we define the decision trees for each of the variables of the $\mathsf{IND}_{\mathsf{PPA}}$ instance. For each $\ell \in [L]$ and $\ell' < \ell$, we define its predecessor variables $P_{\ell'}^{(\ell)}$ by querying $p_1^{(\ell)}$ and $p_2^{(\ell)}$ and outputting 1 iff either of these is ℓ'.

The m-th node of ℓ has its activity variable $A_m^{(\ell)}$ defined by checking that $a^{(\ell)} = 1$, then querying the $d \log n_H$ bits of $\mathsf{mon}_m^{(\ell)}$, and then checking that $\alpha_i = 1$ for all $i \in \mathsf{Vars}(\mathsf{mon}_m^{(\ell)})$ (the variables in monomial m), and outputting 1 if all of these checks pass, and 0 otherwise.

Finally, the matching variables $M_{\ell',m'}^{(\ell)}$ are defined as follows. If $\ell' \neq \ell$ we query $p_1^{(\ell)}$ and $p_2^{(\ell)}$ to determine if either ℓ' is one of the children of ℓ. If it is not then the output of $M_{\ell',m'}^{(\ell)}$ can be arbitrary.

Otherwise, if $\ell' = \ell$ then we can continue. We query $v_1^{(\ell)}$ to determine the variable y that was used to derive monomial m', and we query $\mathsf{match}_{\ell',m}^{(\ell)}$ to obtain a pair $j \in \{0,1,2\} \times [s]$ and $m^* \in [s]$ indicating to which child of ℓ and which monomial m^* the monomial m is matched. As well, we query $\mathsf{match}_{p_j^{(\ell)},m^*}^{(\ell)}$ to ensure that this matching is consistent. Finally, query $\mathsf{mon}_m^{(\ell)}$ and $\mathsf{mon}_{m^*}^{(p_j^{(\ell)})}$, where $p_0^{(\ell)} := \ell$. If the variables occurring m are not the the same as those in $v_1^{(\ell)} m^*$, then let $M_{\ell',m}^{(\ell)}$ be some arbitrary $(\hat{\ell}, \hat{m})$ such that such that $\hat{\ell} \neq p_1^{(\ell)}, p_2^{(\ell)}$. In particular, this means that $a^{(\hat{\ell})} = 0$ and this will cause a violation (solution). Otherwise, let the output of $M_{\ell',m}^{(\ell)} = (p_j^{(\ell)}, m^*)$, where $p_0^{(\ell)} = \ell$.

Next, we define the output decision trees for each solution. Let (ℓ, ℓ', m) be any solution to this instance of $\mathsf{IND}_{\mathsf{PPA}}$. We create a decision tree mapping this solution back to a falsified clause of $\mathsf{Ref}^{\mathsf{PC}}$. If ℓ is one of the initial clauses C_ℓ of H, i.e., $\ell \leq m_H$, then we query whether $C_\ell(\alpha) = 0$, and if so we output the index of the ℓ-th clause of H.

Otherwise, this decision tree first queries the decision tree for $M_{\ell',m}^{(\ell)}$. If this decision tree discovered that m was matched to a monomial m^* in the instance of $\mathsf{Ref}^{\mathsf{PC}}$ for which $m \neq v_1^{(\ell)} m^*$ then we output the index of the clause of $\mathsf{Ref}^{\mathsf{PC}}$ which states that this should not happen. Similarly, if we discovered that m was matched to a monomial m^* but that monomial was not matched back to m then we output the clause of $\mathsf{Ref}^{\mathsf{PC}}$ stating that this cannot happen.

□

Polynomial Calculus over Different Domains

We end this section by sketching how $\mathsf{IND}_{\mathsf{PPA}}$ can be modified in order to characterize proof systems that are similar to the $\mathbb{F}_2$-Polynomial Calculus. First, it is straightforward to generalize $\mathsf{IND}_{\mathsf{PPA}}$ to $\mathsf{IND}_{\mathsf{PPA}_q}$ in order to characterize $\mathbb{F}_q$-Polynomial Calculus for other q. Rather than matching pairs of nodes, in this principle, one matches q-tuples of nodes.

The *unary Polynomial Calculus* uPC proof system is the Polynomial Calculus operating over the integers, rather than a finite field. The addition rule is generalized to allow multiplication by integer-valued constants: from previously derived p, q, one may deduce $\alpha p + \beta q$ for $\alpha, \beta \in \mathbb{Z}$. *Unary* refers to the fact that the size of a uPC proof is the number of monomials counted with multiplicity; i.e., if the monomial αm occurs in some line in the proof then it contributes $|\alpha|$ towards the size. To characterize uPC, one can define $\mathsf{IND}_{\mathsf{PPAD}}$ which is a variant of $\mathsf{IND}_{\mathsf{PPA}}$ with *directed* matchings; this is a generalization of the result from [24] which showed that the unary Nullstellensatz proof system is characterized by PPAD^{dt}. More formally, for each pool ℓ, the predecessor variables $P_{\ell'}^{(\ell)}$ for $\ell' < \ell$ now take value in $\{0, \ldots, N\}$ indicating that line pool ℓ has $P_{\ell'}^{(\ell)}$ copies of ℓ' as its children (and each copy is then involved in the matching for ℓ). Each matching is directed: each node involved in the matching for a pool ℓ has either a successor or a predecessor. Let $\ell' < \ell$, then one should think of a directed edge from a node belonging to (a copy of) ℓ' to a node of ℓ as a monomial with a *positive* coefficient in the ℓ-th line in the uPC proof, while a directed edge from a node of ℓ to a node belong to (a copy of) ℓ' as being a *negative* monomial in line ℓ. If two nodes which belong to children $\ell', \ell'' < \ell$ are matched, then this correspond to a positive instance of a monomial cancelling with a negative instance during addition. The solutions of $\mathsf{IND}_{\mathsf{PPAD}}$ are the solutions of $\mathsf{IND}_{\mathsf{PPA}}$ together with the following any violation to the following two conditions. The first, which prevents nodes from being used multiple times, is that each copy of a node in the matching for ℓ should have a predecessor or successor, but not both. The second, which essentially says that the sign of a monomial should always remain the same, is that if a copy of a node m from line $\ell' < \ell$ is involved in the matching for ℓ then, if it

has a successor (predecessor) in the matching for ℓ then it should have a predecessor (successor) in the in the matching for ℓ'.

By a similar approach we can define an induction PPADS which captures the unary *dag-like Sherali-Adams* proof system (see e.g., [17] for a definition). In fact, it seems that this is a fairly general phenomenon: one can take a TFNP problem which characterizes a *static* proof system and construct an *induction* variant of that TFNP class, in the manner to what we have done here, to characterize the dag-like variant of that proof system.

3 Communication TFNP and Monotone Circuit Complexity

In addition to proof systems, the communication complexity versions of TFNP sub-classes have been used to provide characterizations of monotone circuit models [25]. When combined with lifting techniques translating decision tree lower bounds to communication complexity lower bounds, this has resulted in numerous new lower bounds for a variety of monotone circuit models. For example, Nullstellensatz lower bounds were used to prove PPA lower bounds in the black-box setting, which were lifted to PPA lower bounds in communication, which implied monotone span program bounds. In this section, we give generic conditions under which a class of monotone functions has a communication-TFNP characterization, proving Theorem 3. As has been pointed out in the past, there is a direct mapping from TFNP problems to partial monotone functions, and we utilize this mapping. This gives an exact characterization of when a complexity measure on partial functions has a TFNP characterization. Since complexity measures on total functions induce complexity measures on partial functions, this also gives a general condition under which a complexity measure on total monotone functions has a TFNP characterization. Unfortunately, we don't have a converse statement for total functions and it is conceivable that measures that don't meet our criteria also have TFNP characterizations.

It would be plausible to propose that some of the results here might have analogs for non-monotone models of computation. However, the techniques we use seem not to hold for these models, which might indicate why TFNP or other communication complexity characterizations of non-monotone circuits are much more difficult to use to prove lower bounds.

For n bit strings x and x', we say that x' dominates x, written $x \leq x'$, if for every $1 \leq i \leq n$, $x_i \leq x'_i$. A partial Boolean function f on n bit strings is described by two disjoint sets of strings, No_f the strings which should be rejected, and Yes_f, the strings that should be accepted. f is total if $No_f \cup Yes_f = \{0,1\}^n$. A partial Boolean function f is monotone if whenever $x \in No_f$ and $x' \leq x$, then $x' \in No_f$ and whenever $y \in Yes_f$ and $y \leq y'$ then $y' \in Yes_f$. For partial functions f and g, we say f is *solved* by g if $No_f \subseteq No_g$ and $Yes_f \subseteq Yes_g$.

Let h be a mapping from strings of length n to strings of length n', and f a partial function on n' bit inputs. Then $f \circ h$ is the partial function where $Yes_{f \circ h} = \{x | h(x) \in Yes_f\}$ and $No_{f \circ h} = \{x | h(x) \in No_f\}$. If h is monotone in its input, and f is monotone, then $f \circ h$ is monotone.

Monotone Partial Function Complexity Measures. A monotone partial function complexity measure mpc is a map from partial monotone functions to non-negative integers. Typical such measures are the minimum circuit size in a monotone model of a total function that solves f, but we won't include a circuit model explicitly. For convenience, we assume for any partial function f on n bit inputs, $\mathsf{mpc}(f) \geq n$.

We are now ready to define what a *communication*-TFNP *characterization* of a measure means. For a partial Boolean function f on n inputs, the *Karchmer-Wigderson game* for f, denoted KW_f, is the commu-

nication problem where one player has $x \in No_f$ the other has $y \in Yes_f$ and the output is a position i so that $x_i \neq y_i$. Similarly, for monotone Boolean function f on n inputs, the *monotone Karchmer-Wigderson game* for f, denoted mKW_f, is a restriction of the Karchmer-Wigderson game to require that the output is a position i such that $x_i < y_i$. Karchmer and Wigderson [30] showed that communication complexity of KW_f (mKW_f) is an exact characterization of the (monotone) circuit depth needed to compute f.

Communication TFNP. Consider relational communication problems defined by a predicate $R(x, y, i)$, where $x \in X$, $y \in Y$, and $i \in [M]$. The corresponding communication problem has one player given $x \in X$, the other $y \in Y$, and the goal being to output an i so that $R(x, y, i)$ holds. We say this problem is in t-bit *communication*-TFNP if for every $x \in X$, $y \in Y$, for some i, $R(x, y, i)$; and given i, there is a t-bit communication protocol $V(x, y, i)$ to determine whether $R(x, y, i)$ holds. We say that $R \in \mathsf{TFNP}^{cc}$ if R is in $\mathsf{polylog}(n)$-bit communication TFNP.

We say that one communication problem $R(x, y, i)$ *mapping reduces* to another $R'(x', y', i')$ with communication t if there are functions $M_X : X \to X'$, $M_Y : Y \to Y'$ and a t bit communication protocol $S(x, y, i')$ which outputs i, so that whenever $R'(M_X(x), M_Y(y), i')$, then $R(x, y, S(x, y, i'))$. In particular this means that R requires at most t more bits of communication than R' to solve.

The following lemma from [19] says that TFNP^{cc} is exactly the study of the monotone Karchmer-Wigderson search problem.

Lemma 11. *For any search problem R_N there is a partial function F_N such that R_N is equivalent to mKW_{F_N}.*

Proof. Let $S(x, y, j)$ be a t-bit protocol that verifies that j is a valid solution on input (x, y). We define a partial function F_N on $N = 2^t M$. We think of each co-ordinate as representing a solution j and a communication pattern for $S(x, y, j)$. We then construct the sets No_{F_N} as, for each x in the first players domain of R, putting a 0 in a coordinate corresponding to a communication in the verification for j if the protocol accepts and x is consistent with the communication, and a 1 otherwise. To construct Yes_{F_N}, we do the same for each y in the domain for the second player, except reverse 0 and 1. We claim mKW_{F_N} is equivalent to R_N, using this construction as the map. If we are given a solution j to the constructed instances, we can simulate $S(x, y, j)$, and output j together with the communication pattern for the simulation. This gives a bit where the construction is 0 for the x player and 1 for the y player, solving mKW_{F_N}. In the reverse direction, if we are given such a bit, we know $S(x, y, j)$ accepts, and we can return j. □

Thus, we can restrict attention to instances of the monotone Karchmer-Wigderson search problem. Let R_N be a sequence of communication problems where $X_N, Y_N \subseteq \{0, 1\}^{\mathsf{poly}(N)}$ and $M_N = \mathsf{poly}(N)$. Define the complexity measure R^{cc} on monotone partial Boolean functions f as

$$R^{cc}(f) := \min \log N + t,$$

over the set of N, t so that mKW_f mapping reduces to R_N with t-bits of communication. We say that the sequence R_N is *nestable* if there is a polynomial p and a function $t(N) = O(\log N)$ so that R_N is $t(N')$-communication reducible to $R_{N'}$ for all $N' \geq p(N)$.

The TFNP^{cc} *subclass* associated with R, denoted R^{cc}, is all $Q \in \mathsf{TFNP}^{cc}$ such that $R^{dt}(Q) = \mathsf{polylog}(n)$. Finally, we say that a TFNP^{cc} problem R *characterizes* a mpc if $R^{cc} = \{f : \log \mathsf{mpc}(f) = \mathsf{polylog}(n)\}$.

In the remainder of this section we will prove Theorem 3. We will first prove a tighter characterization, showing when, for an mpc, there exists $R \in \mathsf{TFNP}^{cc}$ such that $R^{cc}(f) = \Theta(\log \mathsf{mpc})$ for all monotone

partial functions f. This tighter characterization will involve a weaker notion of a universal family of functions, which we will call *complete families*. Following this, we give a weaker characterization involving universal function families (Theorem 3).

3.1 Complete Problems give TFNP Characterizations

The characterization of mpc measures with TFNP^{cc} connections involves three properties:

i) *Monotonicity Under Solutions.* A monotone partial function complexity measure mpc is *monotone under solutions* if whenever g solves f, $\mathsf{mpc}(g) \geq \mathsf{mpc}(f)$.

ii) *Closed Under Low-Depth Reductions.* We say that an mpc is closed under low-depth reductions if for any h from n bit to n' bit inputs that is computable by monotone Boolean circuits of depth d, and any partial monotone function f on n' bit inputs, $\mathsf{mpc}(f \circ h) \leq \mathsf{poly}(n, n', \mathsf{mpc}(f), 2^d)$.

iii) *Admits a Complete Family.* A complete family for a monotone partial function complexity measure mpc is a family F_m of functions such that for every partial monotone function f on n bit inputs with $\mathsf{mpc}(f) \leq m$, there is a $\log m$-depth monotone circuit computing h so that $F_m \circ h$ solves f, and $\mathsf{mpc}(F_m) \leq \mathsf{poly}(m)$.

We say that two measures $\mathsf{mpc}_1, \mathsf{mpc}_2$ are *polynomially equivalent* if $\mathsf{mpc}_1(f) \leq \mathsf{poly}(\mathsf{mpc}_2(f))$ and $\mathsf{mpc}_2(f) \leq \mathsf{poly}(\mathsf{mpc}_1(f))$.

We are now ready to prove the main theorem of our section which describes when mpc measures have TFNP^{cc} characterizations.

Theorem 12. *Let* mpc *be a complexity measure. Then there is a nestable sequence of* TFNP *communication problems R_N so that $R^{cc}(f) = \Theta(\log \mathsf{mpc}(f))$ for every monotone partial function f iff (i)–(iii) hold. Moreover, the sequence R_N can be made explicit if and only if the sequence of complete functions for f can be made explicit.*

To prove this, we will use the following lemma, saying that reductions between the Karchmer Wigderson games and reductions between the functions are identical.

Lemma 13. *Let f and g be monotone partial Boolean functions. Then mKW_f has a communication t mapping reduction to mKW_g if and only if there is a function h computable by a depth t monotone circuit so that g solves $f \circ h$.*

Proof. As before, let Yes_f, No_f and Yes_g, No_g be the set of accepting and rejecting inputs of f and g respectively.

For the if direction, we can define the reduction from mKW_f to mKW_g by using h as both M_X and M_Y. Since g solves $f \circ h$, for each $x \in No_f$, it follows that $h(x) \in No_g$, and similarly $y \in Yes_f$ implies $h(y) \in Yes_g$. Thus $(h(x), h(y))$ is a valid input pair to mKW_g. A solution to mKW_g for this input is a bit position i so that $h(x)_i < h(y)_i$. Since h_i is computable in depth t by a monontone circuit, there is a t bit protocol for $\mathsf{mKW}_{h,i}$. Following this protocol on (x, y), will output a position j such that $x_j < y_j$, which is a solution to mKW_f.

Conversely, assume that we have a t-communication reduction $M_X, M_Y, S(x, y, i)$ from mKW_f to mKW_g. For each i, look at the monotone partial function H_j whose *No* inputs are the x for which there is an $x \leq x'$ with $x' \in No_f$ and $M_X(x')_i = 0$ and whose *Yes* instance are those y where there is $y \leq y'$ with $y' \in Yes_f$ and $M_X(y')_i = 1$.

By the definition of reduction, whenever $x' \in No_f, M_X(x')_i = 0, y' \in Yes_f$ and $M_Y(y')_i = 1$, the communication protocol $S(x', y', i)$ returns a position j with $x'_j < y'_j$. Given x and y where there is a dominating and dominated pair (x', y') as above, the parties can without communication, find x' and y' and run $S(x', y')$. Then $x_j \leq x'_j < y'_j \leq y_j$, so this modified protocol solves the $\mathsf{mKW}_{H,i}$ game. Therefore, there is a depth t monontone circuit computing an h_i that rejects all $x \in No_f$ with $M_X(x)_i = 0$ and accepts all $y \in Y_f$ with $M_Y(y)_i = 1$. For each i, for each $x \in No_f$, if $M_X(x)_i = 0$, then $x \in No_{H,i}$, and we have $h_i(x) = 0$, so $h_i(x) \leq M_X(x)_J$ for all $x \in No_f$. Thus, for each $x \in No_f$, $h(x) \leq M_X(x) \in No_g$, so by monotonicity of g, $h(x) \in No_g$. Similarly, if $y \in Yes_f$, we have $h_i(y) = 1$ if $M_X(y)_i = 1$, or $M_X(y) \leq h(y)$, so $h(y) \in Yes_g$. Thus, h reduces f to g and has depth t. □

We will now use the lemma to prove the theorem.

Proof of Theorem 12. Let mpc be a complexity measure with the three listed properties and a complete family of partial monotone functions F_m. Let $R_m := \mathsf{mKW}_{F_m}$ be the monotone Karchmer-Wigderson game for F_m. Observe that F_m reduces to $F_{m'}$ for all $m' \geq \mathsf{mpc}(F_m) = \mathsf{poly}(m)$ via depth $\log m'$ reductions. Thus by Lemma 13, $R_m = \mathsf{mKW}_{F_m}$ reduces to $R_{m'} = \mathsf{mKW}_{F_{m'}}$ with communication $\log m'$ for all such m', and so R is nestable.

We claim $R^{cc}(f) = \Theta(\mathsf{mpc}(f))$ for every monotone partial function f. Letting $m = \mathsf{mpc}(f)$, then f reduces to F_m in depth $\log m$, as F_m is complete. Then by Lemma 13, mKW_f reduces to mKW_{F_m} with communication $\log m$. It follows by definition that $R^{cc}(f) \leq \log(m2^{\log m}) = 2\log m = O(\log \mathsf{mpc}(f))$.

In the other direction, let $R^{cc}(f) = M$. Then there are m, t with $t + \log m = M$ so that mKW_f is t-communication reducible to mKW_{F_m}. By Lemma 13, it follows that $F_m \circ h$ solves f for some depth t circuit h. Then by monotonicity under solutions, and closure under low-depth reductions,

$$\mathsf{mpc}(f) \leq \mathsf{mpc}(F_m \circ h) \leq \mathsf{poly}(\mathsf{mpc}(F_m), 2^t) = \mathsf{poly}(m, 2^t) = \exp(O(M)).$$

Next we prove the converse direction of the theorem. Let R_m be any nestable sequence of communication TFNP problems. Let $\mathsf{mpc}(f) := 2^{R^{cc}(f)}$ for every monotone partial function. We will show that mpc has the properties (i)–(iii). First, by construction, mpc is monotone under solutions. Second, assume $g \circ h$ solves f and h is computable in depth t, and let $M = R^{cc}(g)$. Then mKW_g has a t' bit reduction to R_m where $\log(m2^{t'}) = M$ and by Lemma 13, mKW_f has a t bit reduction to mKW_g. Thus, mKW_f has a $t + t'$ bit reduction to R_N, so $\mathsf{mpc}(f) \leq N2^{t+t'} = M2^{t'}$. Thus, mpc is closed under low-depth reductions.

Finally, we give a complete partial monotone function family for mpc.

Let f be a partial monotone function and let $M = R^{cc}(f)$. Then mKW_f reduces to R_m in communication t where $\log m + t = M$. In particular, t is at most M and $\log m \leq M$. Then by nesting, we can reduce this to some $M' = \mathsf{poly}(M)$ in further communication $O(\log M') = O(\log M)$. So mKW_f reduces to $R_{M'}$ in $O(\log M)$ communication, which is equivalent to reducing to $F_{M'}$ in $O(\log M')$ communication. It follows by Lemma 13 that f reduces to $F_{M'}$ in depth $O(\log M')$. We can then rescale to hide the constants in the order to get a subsequence which is complete under low-depth reductions. □

A partial characterization for complexity measures on total functions

Analogous to measures on partial functions, let a *monotone complexity measure* mc map total monotone functions to non-negative integers. We can define a monotone complexity measure mpc on partial monotone functions from mc by

$$\mathsf{mpc}(F) := \min\{\mathsf{mc}(f) : \text{ total } f \text{ solving } F\}.$$

Observe that mpc will always satisfy monotonicity under solutions because if g solves f, the set of total functions that solve g is a subset of those that solve f, so the min for g will be at least that for f.

Generalizing the definition for partial functions, say that a monotone complexity measure mc has a *complete family* if there is a family of *total* monotone functions F_m such that for every total monotone function f on n bit inputs with $\mathsf{mc}(f) \leq m$, there is a $\log m$-depth monotone circuit computing a function h so that $F_m \circ h$ solves f, and $\mathsf{mc}(F_m) \leq \mathsf{poly}(m)$.

We will prove the following lemma, whose corollary gives sufficient conditions for a monotone complexity measure to give rise to a corresponding TFNP^{cc} problem.

Lemma 14. mpc *is closed under low-depth reductions and has a complete (partial function) family if and only if* mc *is closed under low-depth reductions and has a complete total function family.*

Corollary 15. *If* mc *is closed under low-depth reductions and has a complete family, then it has a* TFNP^{cc} *characterization by a sequence of nestable relations. If not,* mpc *has no such characterization.*

This still leaves open the possibility that there is a characterization of the complexity measure that does not extend to partial functions for some complexity measures without complete problems.

Proof of Lemma 14. To prove the lemma, we will first assume mc is closed under low-depth reductions, e.g., $\mathsf{mc}(f \circ h) \leq \mathsf{poly}(\mathsf{mc}(f), 2^d)$ when h is computable in depth d. Let F be a partial function, and let f be a total function of minimal complexity solving F. Then $f \circ h$ solves $F \circ h$, so $\mathsf{mpc}(F \circ h) \leq \mathsf{mc}(f \circ h) \leq \mathsf{poly}(\mathsf{mc}(f), 2^d) = \mathsf{poly}(\mathsf{mpc}(F), 2^d)$. Conversely, since $\mathsf{mpc}(f) = \mathsf{mc}(f)$ for total functions, it follows immediately that if mpc is closed under low-depth reductions, then so is mc.

If F_m is a class of complete partial functions for mpc, let f_m be corresponding minimal complexity total functions solving F_m. Note that $\mathsf{mc}(f_m) = \mathsf{mpc}(F_m) = \mathsf{poly}(m)$. Let g be a total function and let $m = \mathsf{mpc}(g) = \mathsf{mc}(g)$. Then $F_m \circ h$ solves g for some h of depth $\log m$, and $f_m \circ h$ solves $F_m \circ h$, so $f_m \circ h$ solves g. But the only way for one total function to solve another is to be equal, so $f_m \circ h = g$. So f_m is also complete and, by assumption, is total.

Conversely, if f_m is complete for mc, then let G be a partial function and g a minimal complexity total function solving G, and let $m = \mathsf{mpc}(G) = \mathsf{mc}(g)$. Then $g = f_m \circ h$ for some h of depth $\log m$, and so solves G. Thus, f_m is also complete for mpc. □

3.2 Universal Functions vs. Complete Functions

If we are willing to characterize complexity up to quasi-polynomial amounts rather than polynomial, we can simplify the condition that there be complete functions in the class to having *universal families* of functions, replacing (iii) in Theorem 16 by the following:

(iii') *Admits a Universal Family.* Let F_m be a sequence of partial monotone functions, and let mpc be a complexity measure on such functions. We say F_m is *universal* for mpc if whenever $\mathsf{mpc}(g) \leq m$, there is a fixed string z_g so that $F(x \circ z_g)$ solves $g(x)$. Observe that such an F_m can be viewed as complete under depth 0 reductions.

Theorem 3 will follow by combining the following theorem with Theorem 12.

Theorem 16. *Let* mpc *be a complexity measure with closure under low-depth reductions. Then* mpc *has a universal sequence F_m with* $\mathsf{mpc}(F_m) \leq \mathsf{quasipoly}(m)$ *if and only if it has a sequence G_m of problems that are complete under polylogarithmic depth reductions and* $\mathsf{mpc}(G_m) \leq \mathsf{quasipoly}(m)$.

Proof. If there is a universal family for mpc, F_m, we can let $G_m = F_m$ since as mentioned above, F_m is complete under depth 0 reductions.

Conversely, suppose that $G_m(x)$ is complete under depth $d(m)$ reductions, where $|x| = M \leq \mathsf{poly}(m)$. We want to construct a partial function F_m which can code any composition $g(x) = G_m(h(x))$ for any g with $\mathsf{mpc}(g) \leq m$ and for any h a function with monotone circuit depth at most $d(m)$. We will actually end up coding a more powerful set of reductions, because we cannot code exactly this family and be monotone. Observe that h has at most m input bits, M output bits, and at most $2^{d(m)}$ gates total. Thus, we can embed h into a depth $2d(m)$ alternating unbounded fan-in $\wedge$-$\vee$ circuit with m inputs, M outputs, and $2^{d(m)}M$ gates at each intermediate level. We can represent the connectivity of the embedding by having one bit for each pair of gates, including inputs and outputs, saying whether the earlier gate is an input to the later one.

So we let F_m be a partial monotone function with $m + (m + (2d(m) - 2)M2^{d(m)} + M)^2$ inputs. The first m inputs to F_m code the input x to g, and the other bits, denoted $B_{i,j}$, code the connectivity relation for the circuit computing h. The gates at even levels will be $\vee$-gates, and those at odd levels $\wedge$-gates. Because we need the circuit evaluation problem to be monotone, we cannot enforce that each gate has exactly two incoming wires, so we allow the gates to be arbitrary fan-in instead. If j is a gate on an even levels, for each earlier gate i including input positions, we let $B_{i,j}$ be 1 if i is an input to j and 0 otherwise. For odd levels, we reverse the roles of 0 and 1.

To compute F_m, we work our way up the circuit computing a bit H_i for each gate i. For i in the first level, H_i is the i-th input bit (the i-th bit of x. For other levels, we use the rule $H_j = \bigvee(H_i \wedge B_{i,j})$ at even levels, and $H_j = \bigwedge(H_i \vee B_{i,j})$ at odd levels, where the scope of i is all gates at earlier levels. After computing the values H_j for the gates at the top level, we apply G_m to the result.

By construction, F_m reduces to G_m via a depth $4d(m)$ monotone circuit with fan-in $M2^{d(m)}$ $\wedge$'s and $\vee$'s, which can also be computed by a depth $4d(m)(d(m) + \log M)$ depth fan-in two monotone circuit. Thus, by composition with low-depth reductions, $\mathsf{mpc}(F_m)$ is quasi-polynomial in m. Also, for any g with $\mathsf{mpc}(g) \leq m$, g can be solved by $F \circ h$ where h can be computed by monotone depth d circuits. The input z_g includes the values $B_{i,j}$ according to the connectivity for h; unused bits in z_g can be set to 0. By construction, $F_m(x \circ z_g) = G_m(h(x))$ which solves g. □

4 Future Directions

The TFNP connection, mapping proof systems to circuit lower bounds via lifting, has been extremely successful. Our results show that this TFNP connection is generic , and characterize the conditions under which it can be made. However, there are many gaps left in making these lower bounds systematic rather than ad hoc, and extending them to new models of computation and proof systems.

In particular,

1. We have a generic relationship between proof systems and decision tree TFNP problems, and a generic relationship between monotone circuit complexity problems and circuit lower bounds. Can we complete the chain by proving a generic lifting theorem, and show that for each TFNP problem, lower bounds for the corresponding proof systems and complexity measures are equivalent?

2. Our characterization of proof systems that correspond to TFNP problems involves proving their own soundness. Can we use this to show a version of Gödel's second incompleteness theorem, that some proof systems cannot prove their own soundness because they do not have a tight TFNP connection?

3. Can we use these connections in the other way, showing relationships between TFNP classes by showing relationships between the corresponding proof systems?

4. TFNP has a direct connection to monotone complexity via the monotone KW games. Can we similarly characterize the class of communication problems corresponding to non-monotone KW games?

5. We showed that *reductions* between the monotone KW games were equivalent to small depth monotone reductions between the corresponding functions. Does this extend to non-monotone games and non-monotone reductions? If not, can we give an example of functions with reductions between the KW games and no reductions between the corresponding functions? (Since this is interesting even for sub-logarithmic bit reductions, this could possibly be shown unconditionally without proving new formula lower bounds.)

Acknowledgements

N.F. thanks Robert Robere for extensive discussions about TFNP and its connections to proof and circuit complexity.

References

[1] Albert Atserias and Moritz Müller. Automating rsolution is NP-hard. *Journal of the Association for Computing Machinery*, 67(5):31:1–31:17, 2020.

[2] Paul Beame, Chris Beck, and Russell Impagliazzo. Time-space trade-offs in resolution: Superpolynomial lower bounds for superlinear space. *SIAM J. Comput.*, 45(4):1612–1645, 2016.

[3] Paul Beame, Stephen A. Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. The relative complexity of NP search problems. *J. Comput. Syst. Sci.*, 57(1):3–19, 1998.

[4] Arnold Beckmann and Sam Buss. The NP search problems of frege and extended frege proofs. *ACM Trans. Comput. Log.*, 18(2):11:1–11:19, 2017.

[5] Arnold Beckmann and Samuel R. Buss. The NP search problems of Frege and extended Frege proofs. *ACM Transactions on Computational Logic*, 18(2):Article 11, 2017.

[6] Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. Lower bounds for cutting planes proofs with small coefficients. *J. Symb. Log.*, 62(3):708–728, 1997.

[7] Josh Buresh-Oppenheim and Tsuyoshi Morioka. Relativized NP search problems and propositional proof systems. In *19th Annual IEEE Conference on Computational Complexity (CCC 2004), 21-24 June 2004, Amherst, MA, USA*, pages 54–67. IEEE Computer Society, 2004.

[8] Samuel R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proceedings of the 19-th Annual ACM Symposium on Theory of Computing*, pages 123–131, May 1987.

[9] Samuel R. Buss, Leszek Aleksander Kolodziejczyk, and Neil Thapen. Fragments of approximate counting. *J. Symb. Log.*, 79(2):496–525, 2014.

[10] Siu On Chan, James R. Lee, Prasad Raghavendra, and David Steurer. Approximate constraint satisfaction requires large LP relaxations. *J. ACM*, 63(4):34:1–34:22, 2016.

[11] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, pages 83–97. Association for Computing Machinery, 1975.

[12] Susanna F. de Rezende, Mika Göös, and Robert Robere. Guest column: Proofs, circuits, and communication. *SIGACT News*, 53(1):59–82, 2022.

[13] Susanna F. de Rezende, Or Meir, Jakob Nordström, Toniann Pitassi, Robert Robere, and Marc Vinyals. Lifting with simple gadgets and applications to circuit and proof complexity. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 24–30. IEEE, 2020.

[14] Susanna F. de Rezende, Jakob Nordström, and Marc Vinyals. How limited interaction hinders real communication (and what it means for proof and circuit complexity). *Electron. Colloquium Comput. Complex.*, page 6, 2021.

[15] Noah Fleming. *The Proof Complexity of Integer Programming*. PhD thesis, University of Toronto, Canada, 2021.

[16] Noah Fleming, Mika Göös, Stefan Grosser, and Robert Robere. On semi-algebraic proofs and algorithms. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 69:1–69:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[17] Noah Fleming, Pravesh Kothari, and Toniann Pitassi. Semialgebraic proofs and efficient algorithm design. *Found. Trends Theor. Comput. Sci.*, 14(1-2):1–221, 2019.

[18] Noah Fleming, Denis Pankratov, Toniann Pitassi, and Robert Robere. Random $\Theta(\log n)$-CNFs are hard for cutting planes. *J. ACM*, 69(3):19:1–19:32, 2022.

[19] Anna Gál. A characterization of span program size and improved lower bounds for monotone span programs. *Comput. Complex.*, 10(4):277–296, 2001.

[20] Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. Monotone circuit lower bounds from resolution. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 902–911. ACM, 2018.

[21] Michal Garlik. Resolution lower bounds for refutation statements. In *Proc. 4 Intl. Symp. on Mathematical Foundations of Computer Science (MFCS)*, pages 37:1–37:13, 2019.

[22] Paul Goldberg and Christos Papadimitriou. Towards a unified complexity theory of total functions. *Journal of Computer and System Sciences*, 94:167–192, 2018.

[23] Paul W. Goldberg and Christos H. Papadimitriou. Towards a unified complexity theory of total functions. *Electron. Colloquium Comput. Complex.*, page 56, 2017.

[24] Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Separations in proof complexity and TFNP. *CoRR*, abs/2205.02168, 2022.

[25] Mika Göös, Pritish Kamath, Robert Robere, and Dmitry Sokolov. Adventures in monotone complexity and TFNP. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 38:1–38:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[26] Mika Göös, Sajin Koroth, Ian Mertz, and Toniann Pitassi. Automating cutting planes is NP-hard. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 68–77. ACM, 2020.

[27] Mika Göös, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles are nonnegative juntas. *SIAM J. Comput.*, 45(5):1835–1869, 2016.

[28] Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM J. Comput.*, 47(6):2435–2450, 2018.

[29] Pavel Hrubeš and Pavel Pudlák. Random formulas, monotone circuits, and interpolation. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 121–131, 2017.

[30] Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM J. Discret. Math.*, 3(2):255–265, 1990.

[31] Pravesh K. Kothari, Raghu Meka, and Prasad Raghavendra. Approximating rectangles by juntas and weakly-exponential lower bounds for LP relaxations of CSPs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 590–603. ACM, 2017.

[32] Jan Krajícek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. Symb. Log.*, 62(2):457–486, 1997.

[33] Jan Krajícek. Interpolation by a game. *Math. Log. Q.*, 44:450–458, 1998.

[34] Jan Krajícek. Randomized feasible interpolation and monotone circuits with a local oracle. *J. Math. Log.*, 18(2):1850012:1–1850012:27, 2018.

[35] James R. Lee, Prasad Raghavendra, and David Steurer. Lower bounds on the size of semidefinite programming relaxations. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 567–576. ACM, 2015.

[36] László Lovász, Moni Naor, Ilan Newman, and Avi Wigderson. Search problems in the decision tree model. *SIAM J. Discret. Math.*, 8(1):119–132, 1995.

[37] Shachar Lovett, Raghu Meka, Ian Mertz, Toniann Pitassi, and Jiapeng Zhang. Lifting with sunflowers. In *Electron. Colloquium Comput. Complex*, page 111, 2020.

[38] Toniann Pitassi and Robert Robere. Lifting nullstellensatz to monotone span programs over any field. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1207–1219. ACM, 2018.

[39] Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.

[40] Pavel Pudlák. On the complexity of finding falsifying assignments for herbrand disjunctions. *Arch. Math. Log.*, 54(7-8):769–783, 2015.

[41] Pavel Pudlák and Jirí Sgall. Algebraic models of computation and interpolation for algebraic proof systems. In Paul Beam and Samuel R. Buss, editors, *Proof Complexity and Feasible Arithmetics, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, April 21-24, 1996*, volume 39 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 279–295. DIMACS/AMS, 1996.

[42] Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Comb.*, 19(3):403–435, 1999.

[43] Alexander Razborov. Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. *Izvestiya Mathematics*, 59(1):205–227, 1995.

[44] Robert Robere. Separations in proof complexity and TFNP. Talk, Satisfiability: Theory, Practice, and Beyond Reunion, 2022.

[45] Robert Robere, Toniann Pitassi, Benjamin Rossman, and Stephen A. Cook. Exponential lower bounds for monotone span programs. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 406–415. IEEE Computer Society, 2016.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il