TFNP Characterizations of Proof Systems and Monotone Circuits

Sam Buss
University of California, San Diego, USA

Noah Fleming
Memorial University, Canada

Russell Impagliazzo
University of California, San Diego, USA

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 a communication-efficient manner. 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 [25], and show that it can prove its own soundness.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Proof Complexity, Circuit Complexity, TFNP

Digital Object Identifier 10.4230/LIPIcs.ITCS.2023.65

Funding *Noah Fleming*: NSERC
Russell Impagliazzo: NSF CCF 2212135 and the Simons Foundation

1 Introduction

In recent years, connections between proof systems and monotone circuit models have revolutionized the areas of proof and circuit complexity, allowing for the tools from one area to be applied to problems from the other. These connections take the form of

14th Innovations in Theoretical Computer Science Conference (ITCS 2023).
Editor: Yael Tauman Kalai; Article No. 65; pp. 65:1–65:39
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

– *Interpolation Theorems*, which translate small proofs into efficient computations in an associated model of monotone circuit [6, 16, 17, 19, 30, 34–36, 41, 43, 45].
– *Query-to-Communication Lifting Theorems*, which translate efficient monotone computations into small proofs in an associated proof system [10, 14, 15, 21, 27–29, 33, 37, 39, 40, 44, 47].

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, 26]. 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 set of TFNP problems which are reducible to this 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 novel perspective offered by TFNP has provided a number unique results for proof complexity, such as *complete* tautologies for certain proof systems, as well as striking *intersection theorems* [25].

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 using *communication*-efficient reductions. 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 [46]! Moreover, many of the query-to-communication lifting theorems can be viewed as constructing a query-efficient reduction to a particular TFNP problem 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[1] proof system which can prove its own soundness (a *reflection principle*) is characterized by a TFNP problem — simply the search problem associated with its reflection principle. This gives a recipe for constructing a TFNP problem which characterizes a given proof system, simply write down the search problem for a reflection principle corresponding to that proof system! 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 [25], and show that it can prove its own soundness. For circuit complexity, we show that every well-behaved model of monotone circuit which admits a *universal family* of functions is characterized by a natural 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.

[1] 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 total function NP considers the total search problems for which solutions can be efficiently verified, grouping them into the class TFNP. There is believed to be no complete problem for TFNP [42], 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 principal goal in the study of TFNP is to understand how these sub-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 may no longer observe 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. [26] observed that many prominent TFNP^{dt} problems are *characterized* by associated proof systems 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 well-studied TFNP^{dt} subclasses:

- $\mathsf{FP}^{dt} = \mathsf{TreeRes}$ [38].
- $\mathsf{PLS}^{dt} = \mathsf{Res}$ [9].
- $\mathsf{PPA}^{dt} = \mathbb{F}_2\text{-}\mathsf{NS}$ [26].
- $\mathsf{PPA}_q^{dt} = \mathbb{F}_q\text{-}\mathsf{NS}$ for any prime q [31]
- $\mathsf{PPADS}^{dt} = \text{unary-}\mathsf{NS}$ [25].
- $\mathsf{PPAD}^{dt} = \text{unary-}\mathsf{SA}$ [25].
- $\mathsf{SOPL}^{dt} = \mathsf{RevRes}$ [25].
- $\mathsf{EOPL}^{dt} = \mathsf{RevResT}$ [25].

That is, these proof systems are characterized by complete problems for these classes, and therefore an unsatisfiable formula F can be efficiently proven in one of these proof systems iff S_F lies in the corresponding class. 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, 25, 26].

This relationship has led to a number of striking results for proof complexity as well. 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. [26]

- *Intersection Theorems:* Proof systems which can efficiently prove a formula iff that formula has short proofs in several other proof systems [25].
- *Coefficient Separations:* Separations between the complexity of certain *algebraic* proof system when their coefficients are represented in unary versus binary [25].

Despite all of this there are still many important TFNP^{dt} problems — such as PPP^{dt}-complete problems — which have thus far evaded characterization by a proof system, as well as many important proof systems for which no corresponding TFNP^{dt} problem is known.

Communication TFNP and Monotone Circuit Complexity.

Karchmer and Wigderson [32] showed that the monotone formula complexity of any monotone function f is equal to the communication complexity of mKW_f. Building on this, Razborov [45] 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 PLS^{cc}-complete problems characterize 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 [20, 26].

Following these results, a number of TFNP^{cc} problems have been characterized by models of monotone circuits [17, 26]. 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. [26] observed that in all-known examples of TFNP problems which admit both a characterization by a proof system and a monotone circuit, there exists both an interpolation theorems and query-to-communication lifting theorem between that proof system and monotone circuit. 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 [46]! 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 its proofs are sound. That is, P has small proofs of a reflection principle about itself, 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* 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*.*

By writing down an efficiently verifiable reflection principle for a proof system, this provides a somewhat systematic way of generating a TFNP^{dt} problem which characterizes that proof system. As an example, we define a new TFNP subclass called IND-PPA, which contains problems which can be solved by inductive *inductive* parity arguments. We show that the IND-PPA-complete problem characterizes the $\mathbb{F}_2$-Polynomial Calculus proof system, and furthermore that the $\mathbb{F}_2$-Polynomial Calculus can prove its own soundness.

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

As a bonus, we show that the technique that we use to generate the TFNP^{dt} problem which characterizes the $\mathbb{F}_2$-Polynomial Calculus can readily be applied in order to generate TFNP^{dt} problems which characterize all of the dynamic variants of static proof systems for which TFNP^{dt} are known. In Subsection 2.4, we provide TFNP^{dt} problems for $\mathbb{F}_q$-Polynomial Calculus, unary Polynomial Calculus, and unary dag-like Sherali-Adams.

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 partial 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) *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 $\mathsf{mpc}(f)$.
ii) *Admits a universal family:* a family of functions F_m such that whenever $\mathsf{mpc}(g) \leq m$ 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) and (ii).*

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 17) 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 [23]. On the other hand, Garlik [22] showed that resolution requires exponential length for refutations of (a particular "leveled" version of) its reflection principle, and 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 [22] 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} its solutions are *verifiable*: for each $i \in \mathcal{O}$ there there is a decision tree T_i^o of $\mathsf{polylog}(n)$ depth such that

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

Decision Tree Reductions. 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 $x \in \{0,1\}^s$,

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

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* of the reduction is d, 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 the set of all $Q \in \mathsf{TFNP}^{dt}$ such that $R^{dt}(Q) = \mathsf{polylog}(n)$.

Black-box TFNP is intimately connected with proof complexity. This connection can be summarized by the following claim from [25, 26].

▷ Claim 1. 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 this is robust as for any pair of decision trees $\{T_i^o\}$ and $\{T_i'^o\}$ that verify the same $R \in \mathsf{TFNP}^{dt}$, the resulting false clause search problems S_F and $\mathsf{S}_{F'}$ are $\mathsf{polylog}(n)$-reducible.

Using this connection, Göös et al. [26] 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.

Complexity Measure. The known characterizations of proof systems by TFNP^{dt} problems are in terms of a somewhat non-standard, but very natural, *complexity parameter*. For a proof system P and unsatisfiable CNF formula F let the complexity required by P to prove F be

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

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 $R^{dt} = \{\mathsf{S}_F : P(F) = \mathsf{polylog}(n)\}$; this is reflexive and so we also say that P characterizes R. In fact, many of these characterizations hold in the following 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 for any unsatisfiable CNF formula F,

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

In this section we give necessary and sufficient conditions for such a characterization to occur. The first condition is that the proof system proves an efficiently verifiable variant of 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. This is formalized with variables encoding a CNF H, a proof Π, and a truth assignment α to H. The formula (falsely) asserts that Π is a P-proof of H and α satisfies H,

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

We say that a reflection principle is *efficiently verifiable* if it is encoded as a low-width CNF formula. In this case, solutions to the false clause search problem for the reflection principle (also known as the *wrong proof problem* [4, 24]) can be efficiently verified, which is essential for the reflection principle search problem to belong to TFNP.

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 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 bound the width of the CNF H by 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 (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 bounds the logarithm of the size of the proof, and by the third point, the number of variables is exponential in $\Theta(c)$, the second condition ensures that $V_{n,m,c}$ has width $\mathsf{poly}(c)$ and can be verified by looking at polynomial-in-c many variables. 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 a SAT oracle is used to decide whether the formula is satisfiable. 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 based on a verification procedure.

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,

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

where $[\![p = \ell]\!]$ is the indicator function of p being equal to ℓ. This 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 procedure is 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.

Conditions for a TFNP Characterization

The first necessary condition for a proof system to admit a characterization by a TFNP^{dt} problem will be that the proof system must efficiently prove a reflection principle about itself. The second necessary condition is that the proof system must be closed under *decision-tree reductions*, as TFNP^{dt} is closed under these 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 $O(cd)$.

In all of the 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.

We are now ready to prove Theorem 1, which we state formally as follows.

▶ **Theorem 1.** The following hold:

i) For any TFNP^{dt} problem R there is a proof system P such that R characterizes P. Furthermore, P is closed under decision tree reductions and there is a reflection principle Ref_P for P such that $P(\mathsf{Ref}_P) \leq \mathsf{polylog}(n)$.
ii) For any proof system P which is closed under decision tree reductions and for which there is a reflection principle Ref_P of which P has $\mathsf{polylog}(n)$-complexity proofs, there is a TFNP^{dt} problem R which characterizes P.

In fact, we prove a tighter characterization over the following two subsections, from which Theorem 1 will follow. Part (i) follows from Theorem 6, with the "furthermore" part proven in Theorem 5, while part (ii) is proven in Theorem 4.

2.1 A Proof System for any TFNP Problem

We begin by describing how any TFNP^{dt} problem R can be transformed into a proof system for refuting unsatisfiable CNF formulas of polylog width. The key observation is that because each TFNP^{dt} problem is equivalent to the search problem for some unsatisfiable CNF formula, we can view decision tree reductions between TFNP^{dt} problems as proofs in a proof system — indeed, these reductions are sound and efficiently verifiable! More formally, a proof Π in this proof system, of the (unsatisfiability) of a CNF formula H, will consist of a low-depth decision reduction from S_H to an instance of 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 variables 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 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 satisfying 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 $T^0 = \{T_i^o\}_{i\in[m]}$ of the reduction. The *size* of Π 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 proving an unsatisfiable CNF formula H is then the minimum over all P-proofs of H,

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

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. Finally, note that the Note that the canonical proof system is closed under decision tree reductions.

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

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 $n = O(s)$. ◀

Canonical Proof Systems Prove their 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 encode proofs Π in the canonical proof system — which are decision tree reductions — we require the notion of a generic of a decision tree, which is a template for decision trees of depth at most d — any decision tree of depth at most d (over a set of variables $\alpha_1, \ldots, \alpha_n$ and output set $\mathcal{O}$) can be recovered from an assignment to the variables of a generic decision tree.

A *generic 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, the generic decision tree induces a decision tree that queries the 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 generic 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.

Recall that in the reflection principle $\mathsf{Proof}(H, \Pi)$ states that the proof Π (which we will encode using generic decision trees) is indeed a proof of H. To state $\mathsf{Proof}(H, \Pi)$, it will be helpful to have the following definition. The decision tree *simulating* a generic 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 generic 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 $V^{P_F}_{n_H, m_H, (d, n_F)}$ 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 generic decision trees $\hat{T}_1, \ldots, \hat{T}_{n_F}$ with output in $\{0,1\}$ and m_F depth-d generic decision trees $\hat{T}_1^o, \ldots, \hat{T}_{m_F}^o$ 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.[2] 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.

We will construct the constraints of Proof from the following decision trees T_{C_i}, for each clause C_i in F: 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 the 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

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

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 generic 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 $n = m_F 2^{d+1} + n_F 2^d \log n_H + cm_H \log 2n_H$ 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. Thus, $\mathsf{S}_{\mathsf{Ref}_F} \in \mathsf{TFNP}^{dt}$.

▶ **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 $c = d + \log n_F$. For each generic 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 $T_1, \ldots, T_{n_F}, T_1^o, \ldots, T_{m_F}^o$ of the reduction 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 truth assignment $(T_1(x), \ldots, T_{n_F}(x))$ falsifies 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. Thus, 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_F. ◀

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} is closed under these reductions. That is, it must 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, let V be a verification procedure for P, and denote $\mathsf{Ref}_{P,V}$ by Ref. For any unsatisfiable CNF formula H, the following hold.*

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* $\mathsf{S}_{\mathsf{Ref}}$.

The first statement says that any P-proof of H induces a reduction from S_H to $\mathsf{S}_{\mathsf{Ref}}$ of the same complexity. The second statement is a converse, saying that if there is a reduction from S_H to $\mathsf{S}_{\mathsf{Ref}}$ *and* P can efficiently prove Ref then there is a P-proof of H whose complexity is not much larger than the complexity of the reduction — in particular, it is factor of the complexity needed for P to prove Ref larger than the complexity of the reduction.

Before proving this theorem we will give a high-level sketch of the proof for the case of $\mathsf{polylog}(n)$-complexity reductions. Let P be any proof system that is closed under decision tree reductions. Observe that $\mathsf{S}_{\mathsf{Ref}} \in \mathsf{TFNP}^{dt}$ as Ref is efficiently verifiable. Consider any $\mathsf{S}_H \in \mathsf{TFNP}^{dt}$ such that $\mathsf{S}^{dt}_{\mathsf{Ref}}(\mathsf{S}_H) = \mathsf{polylog}(n)$ (S_H reduces to $\mathsf{S}_{\mathsf{Ref}}$ with polylog-depth decision trees). Then, 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. 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 variables free. On any input α to the variables of 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. We will begin by proving (ii). 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 $\mathsf{S}_{\mathsf{Ref}}$. As P is closed under decision tree reductions, there is a P-proof of H with complexity $P(H) = O(\mathsf{S}^{dt}_{\mathsf{Ref}}(\mathsf{S}_H)P(\mathsf{Ref}))$.

To prove (i), suppose that Π is a complexity $c := P(H)$ proof in P of an unsatisfiable CNF formula H. We will 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 of H respectively. The reduction $T = (T_1, \ldots, T_n)$ hardwires the input (H, Π) into the instance of $\mathsf{S}_{\mathsf{Ref}}$ with parameters 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, establishing Theorem 2. This answers an open question from [25], asking for a characterization of the Polynomial Calculus. To define our characterization of the $\mathbb{F}_2$-Polynomial Calculus, we will leverage the characterization of its *static* variant, $\mathbb{F}_2$ Nullstellensatz, by PPA-complete problems [26]. PPA is the class of TFNP problems which can be solved by parity arguments, and the standard PPA-complete problem is $LEAF$ — given a fan-in ≤ 2 graph and a designated leaf v^*, find another leaf. To characterize the $\mathbb{F}_2$-Polynomial Calculus, we define the TFNP class IND-PPA which corresponds to *inductive* parity arguments, and whose complete problem is the $LEAF$ problem defined over a directed acyclic graph. At the end of this section we discuss how this appears to be a general phenomenon — for any TFNP problem which characterizes a *static* proof system, we can define an *induction* variant of that problem to characterize the *dynamic* variant of that proof system. Using this, we give TFNP problems which characterize the $\mathbb{F}_q$-Polynomial Calculus, unary Polynomial Calculus, and unary 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 the equation $\overline{C}_i$ such that $C_i(x) = 1$ iff $\overline{C}_i(x) = 0$ (for example, $(x_1 \vee \neg x_2 \vee x_3)$ represented as $(1 + x_1)x_2(1 + x_3) = 0$). Note that the degree of $\overline{C}_i$ is equal to the width of C_i. We will operate exclusively with multilinear arithmetic; that is, 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 is a derivation of the trivially false polynomial $1 = 0$ from $\{p_i(x) = 0\}_{i\in[m]}$ by the following two rules:

Addition. From two previously derived polynomials p, q, deduce $p + q$.
Multiplication by a Variable. From a previously derived polynomial p, deduce x_ip 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

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

Next, we define IND-PPA, the subclass of TFNP^{dt} problems which are reducible to the IND-PPA-complete problem *IND-LEAF*, which will characterize $\mathbb{F}_2$-PC. At a high level this is the *LEAF* problem defined over a directed acyclic graph (dag). An instance of this problem is given by a set set of N nodes (corresponding to monomials) and a set of L pools (corresponding to lines in the proof). The pools are arranged in a dag; each pool $\ell \in [L]$ has a set of immediate predecessors described by variables $P^{(\ell)}_{\ell'} \in \{0, 1\}$ for $\ell' < \ell$. Each pool ℓ is associated with a set of nodes $A^{(\ell)} \subseteq [N]$ and we hard-code that the root pool L has $A^{(\ell)} = \{1\}$ for some distinguished 1-node. We have an instance of *LEAF* defined over the nodes of this dag as follows: for each pool ℓ we have a matching $M^{(\ell)}$ between the nodes of ℓ and the nodes of its predecessors; see Figure 1. Since the L-th pool contains only a single node, there must be some pool with an unmatched node. A solution is an unmatched or mismatched node.

We remark that the dag of pools is specified by input variables $P^{(\ell)}_{\ell'}$ to the problem. This is crucial; if the dag was fixed in advance, then this problem would be in PPA — there is a simple reduction to *LEAF* — and thus gives rise to a Nullstellensatz proof.

Induction PPA. The IND-PPA-complete problem *IND-LEAF* 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 $A^{(\ell)}_m \in \{0, 1\}$ indicating whether node m is *active* at pool ℓ. Finally, we have a *matching* between the nodes of

$\ell \in [\ell]$ and the nodes of all of its predecessors: For each $\ell' < \ell$ and $m \in [N]$ there is a variable $M^{(\ell)}_{\ell',m'} \in [\ell] \times [N]$ indicating where ℓ''s copy of node m' is matched in the matching for pool ℓ. The *root* pool L has $A^{(L)}_1 = 1$ and $A^{(L)}_m = 0$ 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 m is an active node for ℓ', and m is matched to some node m' of some pool ℓ'' which is not matched to m. That is, $P^{(\ell)}_{\ell'} = 1$, $A^{(\ell)}_m = 1$, $M^{(\ell)}_{(\ell',m)} = (\ell'', m')$, and either $P^{(\ell)}_{\ell''} = 0$, $A^{(\ell'')}_{m'} = 0$, or $M^{(\ell)}_{\ell'',m'} \neq (\ell', m)$.

Observe that this problem is in TFNP^{dt}, as any candidate solution can be verified by observing the values of $O(\log n)$ many variables.

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

– *If there is a depth-d reduction fron S_F to an instance of IND-LEAF on s variables, then there is a degree-$O(d)$, size-$s^2 2^{O(d)}$ $\mathbb{F}_2$-PC proof of F.*

– *If F has a size-s and degree-d $\mathbb{F}_2$-PC proof of F, then there is a depth-$O(d)$ reduction from S_F to an instance of IND-LEAF on $O(s^2)$-variables.*

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

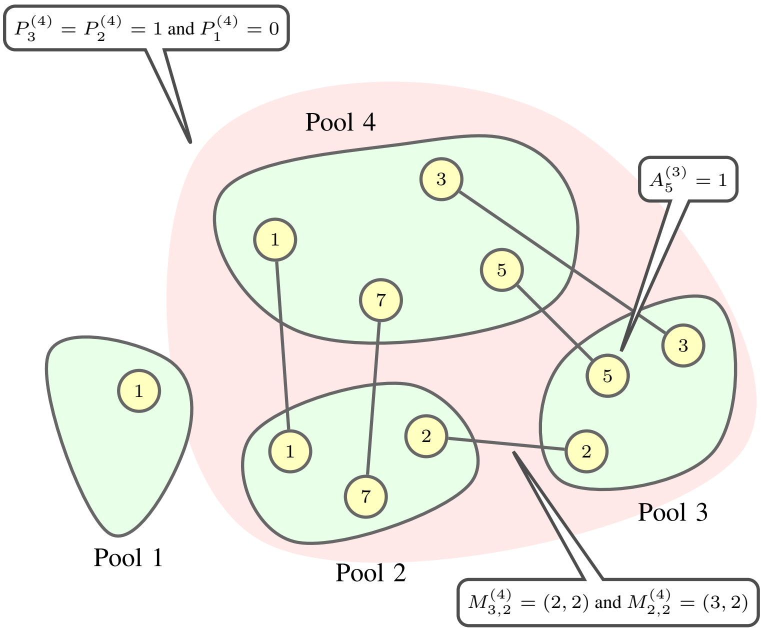

Figure 1 An example matching for Pool 4. The pink area indicates the active predecessors of Pool 4. The yellow circles indicate the active nodes for that pool; for example Pool 1 has only node 1 active: $A^{(1)}_1 = 1$, while $A^{(1)}_m = 0$ for all $m \neq 1$. The edges correspond to the matching for pool 4. For example, $M^{(4)}_{2,2} = (3,2)$ and $M^{(4)}_{3,2} = (2,2)$ meaning that in the matching for pool 4, the copy of node 2 in pools 3 and 2 are matched.

We remark that an analogous statement holds for the $\mathbb{F}_2$-PCR proof system, which builds on $\mathbb{F}_2$-PC to include additional "dual" 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 only a change to the encoding of the CNF formula F as a set of polynomials and does not affect the resulting TFNP^{dt} problem. Note that this does not contradict the separation between PC and PCR due to de Rezende et al. [13], as their separation is in terms of size, while this characterization is in terms of the complexity measure.[3]

We break the proof of this theorem into two lemmas, Lemma 8 and Lemma 9. In the proofs of both lemmas we will crucially use the fact that any depth-d decision tree (as well as any path in that decision tree) can be encoded as a degree-d polynomial.

▶ **Lemma 8.** *Let F be an unsatisfiable CNF formula. If S_F is reducible to an instance of IND-LEAF 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 S_F is reducible to an instance of *IND-LEAF* on n variables using decision trees of depth at most d. That is, each variable x of the *IND-LEAF* instance 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 (those labelled 1). As well, let $\{T_i^o\}_i$ be the decision trees for each solution i of the *IND-LEAF* instance.

For $\ell \in L$ let

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

over $\mathbb{F}_2$. We will derive by induction on $\ell = 1, \ldots, L$ that $q_\ell = 0$. Roughly, this polynomial states 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, we will define an additional variable $P_\ell^{(\ell)} := 1$, whose decision tree is the constant 1 function.

▷ Claim 2. There is a degree-$O(d)$, size-$NL2^{O(d)}$ $\mathbb{F}_2$-PC proof of the polynomial

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

from the axioms.

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 has a low-degree proof from F by the previous claim, gives $p_\ell = 0$. Note that since every $p_{\ell'}$ is a degree-d polynomial, each of these

[3] Indeed, for any CNF formula F of width w, there are $2w$-depth decision tree reductions between S_F and S_D where D is the encoding of F as a system of polynomial equations using dual variables. That S_F reduces to S_D is immediate. To reduce S_D to S_F define decision trees $T_i = x_i$ for each $i \in [n]$ (querying the positive dual variable for x_i). For each clause C_j of F define decision trees T_j^o as follows: for each variable $x_i \in C_j$, query x_i and its dual variable $\overline{x}_i$; if these variables are not consistent, output the index of the constraint $x_i + \bar{x}_i = 0$ which is violated. Otherwise, if all x_i and $\bar{x}_i$ are consistent, output the index of the (polynomial encoding the) clause C_j.

polynomials has degree at most $2d$. Therefore, this inductive step requires degree $O(d)$ and size $LN2^{O(d)}$. ◀

Proof of Claim 2. To prove this claim we will show that this polynomial can be written as a sum of indicator functions of whether each active monomial in a predecessor of ℓ is correctly matched. Then, we break this polynomial up into indicators corresponding to correct and erroneous matchings. We show that the correct matchings sum to 0 by a parity argument, and that the erroneous matchings can be derived from the axioms (using the fact that they produce a solution to the *IND-LEAF* instance).

For any function f element o in the range of f, denote by $[\![f = o]\!]$ the indicator polynomial which is 1 on input x if $f(x) = o$ and 0 otherwise. For $m \in [N]$ and $\ell' < \ell$ consider the polynomial

$$\mathsf{match}^{(\ell)}_{m,\ell'} := \sum_{\substack{m^*\in[N],\\ \ell^*\in[\ell]}} \left[\!\!\left[M^{(\ell)}_{m,\ell'} = (m^*,\ell^*)\right]\!\!\right] \sum_{\alpha,\beta\in\{0,1\}} \left[\!\!\left[\mathsf{P}^{(\ell)}_{\ell^*} = \alpha\right]\!\!\right] \left[\!\!\left[A^{(\ell^*)}_{m^*} = \beta\right]\!\!\right] \sum_{\substack{\hat{m}\in[N],\\ \hat{\ell}\in[\ell]}} \left[\!\!\left[M^{(\ell)}_{m^*,\ell^*} = (\hat{m},\hat{\ell})\right]\!\!\right],$$

which records all possible matchings for m and matchings of the node that it is matched to. That is, $\mathsf{match}^{(\ell)}_{m,\ell'}$ is the sum over all of the paths in the decision tree that results from sequentially running the decision trees for $M^{(\ell)}_{m,\ell'}, P^{(\ell)}_{\ell^*}, A^{(\ell^*)}_{m^*}$, and finally $M^{(\ell)}_{m^*,\ell^*}$. As $\mathsf{match}^{(\ell)}_{m,\ell'}$ is the sum over all of the paths in a decision tree, it follows that $\mathsf{match}^{(\ell)}_{m,\ell'} = 1$. Using this polynomial, define

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

which records the matching for pool ℓ. Note that $\mathsf{match}^{(\ell)} = \sum_{\ell'\in[\ell]} \sum_{m\in[N]} P^{(\ell)}_{\ell'} A^{(\ell')}_m$ as $\mathsf{match}^{(\ell)}_{m,\ell'}$ is equal to 1.

We will partition the terms of $\mathsf{match}^{(\ell)}$ into two sets, where C is the set of terms where the copy of m belonging to ℓ' is *correctly* matched — that is, for all $\ell' \le \ell$ and $m \in [N]$ with $P^{(\ell)}_{\ell'} = 1$ and $A^{(\ell')}_m = 1$, m is matched to a node $m^* \in [N]$ belonging to a pool $\ell^* \le \ell$ ($M^{(\ell)}_{\ell',m} = (\ell^*, m^*)$) with $P^{(\ell)}_{\ell^*} = 1$ and $A^{(\ell^*)}_{m^*} = 1$ which is matched back to m ($M^{(\ell)}_{\ell^*,m^*} = (\ell', m)$) — and E the remaining terms corresponding to *erroneous* matchings. 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, summing over the terms in C gives $\sum_{t\in C} t = 0$.

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^o_s 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

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

An incorrect matching s is a solution to *IND-LEAF* instance. Thus, as this instance of *IND-LEAF* solves S_F, any truth assignment x which satisfies t_s must also falsify the $T^o_s(x)$-th clause of F. It follows each term of $t_s \cdot T^o_s$ which is not identically 0 must contain the polynomial $\overline{C}$ for some clause C of F, and therefore $t_s \cdot T^o_s = 0$ can be derived by multiplication from the axiom $\overline{C} = 0$. Thus, as each of the $P^{(\ell)}$, $M^{(\ell)}$, and $A^{(\ell)}$ variables are computed by depth-d decision trees,

$$\sum_{\ell'\le\ell} P^{(\ell)}_{\ell'} \sum_{m\in[N]} A^{(\ell')}_m = \sum_{\ell'\in[\ell]} P^{(\ell)}_{\ell'} \sum_{m\in[N]} A^{(\ell')}_m \mathsf{match}^{(\ell)}_{m,\ell'} = \mathsf{match}^{(\ell)} = \sum_{t_s\in E} t_s \cdot T^o_s = 0$$

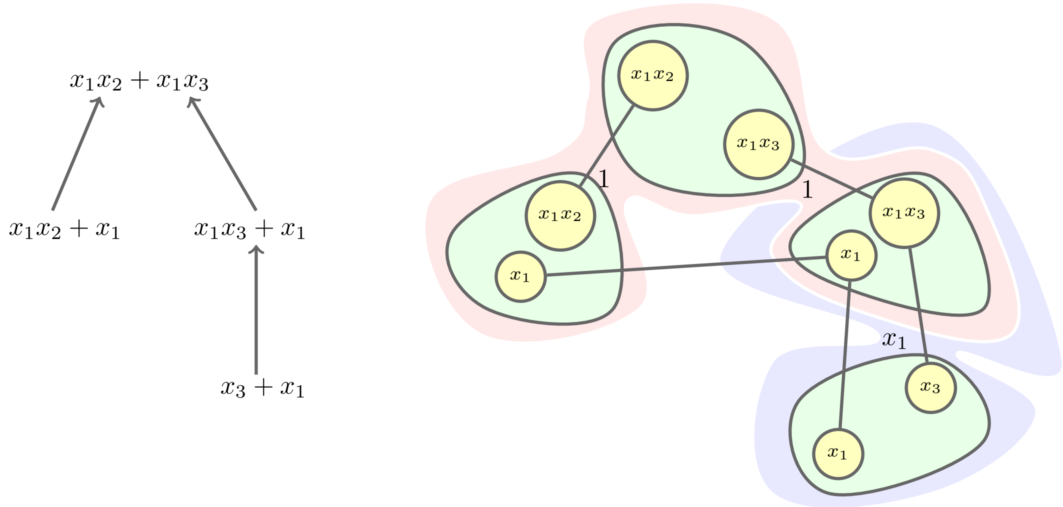

Figure 2 A *IND-LEAF* instance constructed from a Polynomial Calculus derivation. Left: a Polynomial Calculus derivation. Right: the corresponding *IND-LEAF* instance. The non-zero variable of the *IND-LEAF* is labelled with the variables that they query in their decision tree. The red area is represents 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.

has a degree-$6d$ and size-$NL2^{O(d)}$ $\mathbb{F}_2$-PC derivation. ◀

We now prove the converse of Theorem 7, which follows from the next lemma noting that the length of a $\mathbb{F}_2$-PC proof is always upper-bounded by the size.

▶ **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 degree-d then S_F is reducible by decision trees of depth $O(d)$ to an instance of* IND-LEAF *on $O(sL)$ variables.*

A representation of this construction is given in Figure 2.

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 *IND-LEAF* over pools $[L]$ and nodes $[N]$. We will 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 monomial $m \in [N]$ 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 monomials m, set $A_m^{(\ell)} = 0$. Next, 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 $P_{\ell^*}^{(\ell)} = 0$ for all other $\ell^* \in [L]$. Otherwise, if ℓ was derived by *multiplication* by a variable x_i from ℓ', then we 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 $M^{(\ell)}_{\ell,m}$ 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 $M^{(\ell)}_{\ell',m} = (\ell', m \setminus x_i)$ and $M^{(\ell)}_{\ell',m\setminus x_i} = (\ell', m)$. Finally, for match the m which do not occur in ℓ or ℓ' arbitrarily.

Lastly, we set the matching variables of the $\ell \in L$ which correspond to an axiom $A \in \{\overline{C} : C \in F\}$. Each $M^{(\ell)}_{\ell,m}$ 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 *IND-LEAF* 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 since $L \leq s$, we could have used s instead and included additional variables to indicate which lines are actually part of the proof and which are not; this would only affect the complexity up to log-factors. As well, for simplicity, 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 $x, y \in \{0, 1, x_1, \ldots, x_n\}$.

Every line $\ell \in [L]$ is described by a list of s degree-d monomials $\mathsf{mon}^{(\ell)}_m$ for $m \in [s]$, where $\mathsf{mon}^{(\ell)}_{m,i} \in [n_H]$ 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, not every line contains all s monomials, and so we include a variable $a^{(\ell)}_m \in \{0, 1\}$ which indicates whether the i-th monomial is *active* — that is, whether it occurs in line 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 $p^{(\ell)}_1, p^{(\ell)}_2 \in [\ell - 1]$ indicating the lines from which ℓ was derived (if any), and a pair of indices $v^{(\ell)}_1, v^{(\ell)}_2 \in [n_H + 2]$ indicating the variables x, y that the lines indicated by $p^{(\ell)}_1, p^{(\ell)}_2$ 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 $\mathsf{match}^{(\ell)}_{\ell',m'} \in \{0, 1, 2\} \times [s]$, where 0 indicates that m' is matched to a monomial in ℓ, 1 to a monomial in $p^{(\ell)}_1$ and 2 means that it is matched to a monomial in $p^{(\ell)}_2$. For the leaves $\ell \in [m_H]$ we enforce that there is a matching between its active monomials $\mathsf{match}^{(\ell)}_{\ell,m'} \in [s]$.

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 will require 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 $\bmod 2$. More concretely, that each active monomial in ℓ is matched to the active monomial in $p_1^{(\ell)}$ or $p_2^{(\ell)}$ from which it was derived.

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 efficient 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 $\mathsf{S}_{\mathsf{Ref}^{\mathsf{PC}}}$ to the IND-PPA-complete problem *IND-LEAF* Fix an instance of $\mathsf{Ref}^{\mathsf{PC}}$ with parameters $n_H, m_H, (d, s, L)$. We construct an instance of *IND-LEAF* with L pools and s nodes. The high-level idea of the proof is simple: we view $\mathsf{Ref}^{\mathsf{PC}}$ as *IND-LEAF*, 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 *IND-LEAF* instance so that two nodes m, m' that are matched in $\mathsf{Ref}^{\mathsf{PC}}$ are matched in *IND-LEAF* 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 the variables of *IND-LEAF*. 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 if either of these is ℓ', and 0 otherwise.

We define the activity $A_m^{(\ell)}$ of the m-th node of ℓ by querying whether $a^{(\ell)} = 1$, 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). $A_m^{(\ell)} = 1$ 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 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 in 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 $\hat{\ell} \neq p_1^{(\ell)}, p_2^{(\ell)}$. In particular, this means that $a^{(\hat{\ell})} = 0$ and this will cause a violation (solution). Otherwise, set $M_{\ell',m}^{(\ell)} = (p_j^{(\ell)}, m^*)$.

Next, we define the output decision trees for the solutions of this instance of *IND-LEAF*. Let (ℓ, ℓ', m) be a solution, we create a decision tree mapping this solution back to a falsified clause of $\mathsf{Ref}^{\mathsf{PC}}$ as follows. 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 falsified constraint of SAT which states that the ℓ-th clause of H is satisfied under α. Otherwise, this decision tree queries the decision tree for $M_{\ell',m}^{(\ell)}$. If we discover that m is matched to a monomial m^* with $m \neq v_1^{(\ell)} m^*$, or if m is matched to a monomial m^* but that monomial is not matched to m, then we output the index of the clause of $\mathsf{Ref}^{\mathsf{PC}}$ which states that this should not happen.

This completes the description of the reduction. Each of the decision trees involved queries at most $\mathsf{polylog}(n)$ many variables and thus by Theorem 7 it follows that there is a $\mathsf{polylog}(n)$-complexity $\mathbb{F}_2$-PC proof of $\mathsf{Ref}^{\mathsf{PC}}$. ◀

2.4 Characterizing Dynamic Variants of Static Systems

We end this section by discussing how *induction* variants of TFNP problems can be used to generalize TFNP^{dt} characterizations of static proof systems (such as Nullstellensatz and Sherali-Adams) to characterizations of their dynamic variants (such as the Polynomial Calculus and dag-like Sherali-Adams). At a high-level, this is done as follows: if a static proof system is characterized by a TFNP problem R then we can define an *IND-*R problem to characterize the dynamic version of the proof system as follows: there are pools $1, \ldots, L$ which correspond to the lines of the proof, and each $\ell \in [L]$ has children defined by variables $P^{(\ell)}_{\ell'}$ which indicates whether ℓ' is an immediate predecessor of ℓ. Thus, the pools together with their predecessors define the dag-structure of the proof. We then have an instance of the TFNP problem R defined over this dag. The crucial part is that the predecessors $P^{(\ell)}$ of ℓ are not fixed. As examples of this, we show how to leverage the known TFNP^{dt} characterizations of the static proof systems $\mathbb{F}_q$-Nullstellensatz [31], unary Nullstellensatz [25], and unary Sherali-Adams [25] to define TFNP^{dt} problems which characterize their dynamic variants, $\mathbb{F}_q$-PC, unary PC, and unary dag-like Sherali-Adams.

$\mathbb{F}_q$-Polynomial Calculus.

First, it is straightforward to generalize the IND-PPA-complete problem *IND-LEAF* to characterize $\mathbb{F}_q$-Polynomial Calculus for other $q \neq 2$. The IND-PPA$_q$-complete problem *IND-LEAF*$_q$ will be defined as *IND-LEAF* except that one matches q-tuples rather than pairs. It is not difficult to see that this characterizes $\mathbb{F}_q$-Polynomial Calculus. Using *IND-LEAF*$_q$, one can obtain a variant of Theorem 7 for $\mathbb{F}_q$-PC by an almost identical proof.

Unary Polynomial Calculus.

The *unary Polynomial Calculus* (uPC) proof system is the Polynomial Calculus operating over the integers, rather than a finite field. *Unary* refers to the fact that the size of a uPC proof is measured with coefficients written in unary — if a monomial αm, for $\alpha \in \mathbb{Z}$, occurs in some line in the proof then it contributes $|\alpha|$ towards the size. We will use the following non-standard definition of the Polynomial Calculus over the integers. An unsatisfiable CNF formula $F = C_1 \wedge \ldots \wedge C_m$ is encoded as a system of equations by mapping each C_i clause to the polynomial equation $\overline{C}_i$ such that $C_i(x) = 1$ iff $\overline{C}_i(x) = 0$. The *unary Polynomial Calculus* will prove that F is unsatisfiable by deriving the constant -1 from the equations $\{\overline{C}_i(x) = 0, -\overline{C}_i(x) = 0 : C_i \in F\}$ using the the *addition* and *multiplication by a variable* rules as stated for $\mathbb{F}_2$-PC[4]. As before, we operate over the ideal $\langle x_i^2 = x_i \rangle_{i \in [n]}$, thus multi-linearization is done implicitly.

Using the characterization of the unary Nullstellensatz proof system (the static version of uPC) by the PPAD-complete problem *END-OF-LINE* [25], one can define an *IND-END-OF-LINE* problem which will be complete the complete problem for a corresponding IND-PPAD class, in order to characterize uPC. The main difference between *IND-END-OF-LINE* and *IND-LEAF* is that the edges in the matchings of *IND-END-OF-LINE* are *directed*. The direction of the edges in the matching will be used to indicate the signs of monomials in the uPC proof as follows: If a node $m \in [N]$ belonging to pool ℓ occurs are the head of an arrow (directed edge) in the matching $M^{(\ell)}$ then it is considered *positive*, while if it occurs are the tail of an arrow in $M^{(\ell)}$ then it is *negative*. However, if

[4] Typically, the Polynomial Calculus is defined with a *multiplication* rule rather than addition, where one may derive $\alpha p + \beta q$ from previously derived polynomials p, q and $\alpha, \beta \in \mathbb{Z}$. However, as we are measuring coefficients in unary, multiplication by positive coefficients may be simulated by repeated addition. To simulate the use of negative coefficients, we push the negations to the leaves of the proof and include both $\overline{C}_i = 0$ and $-\overline{C}_i = 0$ as axioms.

m belongs to a pool ℓ then if it occurs at the head of an arrow in $M^{(\ell^*)}$ for $\ell^* > \ell$ then it is considered negative and if it as the tail then it is positive. This change in meaning depending on whether this is the matching for the pool ℓ to which it belongs versus a parent pool should be thought of as the sign of monomials propagating from the children ℓ to the parent ℓ^* in the matching $M^{(\ell^*)}$.

2.4.0.1 Induction PPAD.

The IND-PPAD complete problem *IND-END-OF-LINE* is defined as follows:

- *Structure.* $[L]$ pools and $[N]$ nodes. Each $\ell \in [L]$ will correspond to a line in the polynomial calculus proof and be associated with its own copy of $[N]$.
- *Variables.* For each $\ell \in [L]$ and $\ell' < \ell$ we will have a *predecessor* variable $P^{(\ell)}_{\ell'} \in \{0,1\}$ indicating whether ℓ' is a predecessor of ℓ. For each pool $\ell \in [L]$ and each node $m \in [N]$ we have a variable $A^{(\ell)}_m \in \{0,1\}$ indicating whether node m is *active* in pool ℓ. There is a distinguished node $1 \in [N]$ and we hardcode that $A^{(\ell)}_1 = 1$ and $A^{(\ell)}_m = 0$ for all $m \neq 1$. Finally, we have a *directed matching* between the nodes in pools $\ell' \leq \ell$, defined by variables $M^{(\ell)}_{\ell',m} \in \{-,+\} \times [L] \times [M]$ indicating to which node and pool ℓ''s copy of m is matched in a directed fashion, and whether it appears at the head (+) or tail (-) of the arrow.
- *Solutions.* IND-PPAD will state the following: (i) For each pool ℓ with no predecessors, $M^{(\ell)}$ enforces that there is a matching between the nodes of pool ℓ. (ii) if $\ell' < \ell$ is a predecessor of pool ℓ then either every active node of m of ℓ occurs at the *opposite* end of an arrow in the matching $M^{(\ell)}$ for ℓ than in matching for $M^{(\ell')}$ (e.g., m occurs at the tail of an edge in $M^{(\ell)}$ and the head of an edge in $M^{(\ell')}$), or every active node m of ℓ occurs at the *same* end of an arrow in $M^{(\ell)}$ as in $M^{(\ell')}$. (iii) The root pool L contains only a distinguished 1-node. Observe that (i) – (iii) cannot hold simultaneously, and thus IND-PPAD is total. Formally, the solution of IND-PPAD are as follows:
  - *Matching Solutions.* A triple $(\ell, \ell', m) \in [L]^2 \times [N]$ such that ℓ' is either a predecessor of ℓ or ℓ itself, m is an active node of ℓ' and m is matched to a node m'' of some pool ℓ'' but m'' is not matched back to m. That is, $P^{(\ell)}_{\ell'} = 1$ or $\ell = \ell'$, $A^{(\ell')}_m = 1$, $M^{(\ell)}_{\ell',m} = (\alpha, \ell'', m'')$ for some $\ell'' \in [L], m'' \in [N], \alpha \in \{-,+\}$, but either $A^{(\ell'')}_{m''} = 0$ or $M^{(\ell)}_{\ell'',m''} \neq (\beta, \ell', m)$, where β is the opposite sign of α (i.e., m is the head of an arrow to m'', but m'' is not the tail).
  - *Polarity Solutions.* A tuple $(\ell, \ell', m) \in [L]^2 \times [N]^2$ which violates (ii). That is, $A^{(\ell')}_m = 1$, $P^{(\ell)}_{\ell'} = 1$, $M^{(\ell')}_{\ell',m} = (\alpha, *, *)$ and $M^{(\ell)}_{\ell',m} = (\alpha, *, *)$.

A portion of an instance of *IND-END-OF-LINE* is depicted in Figure 3.

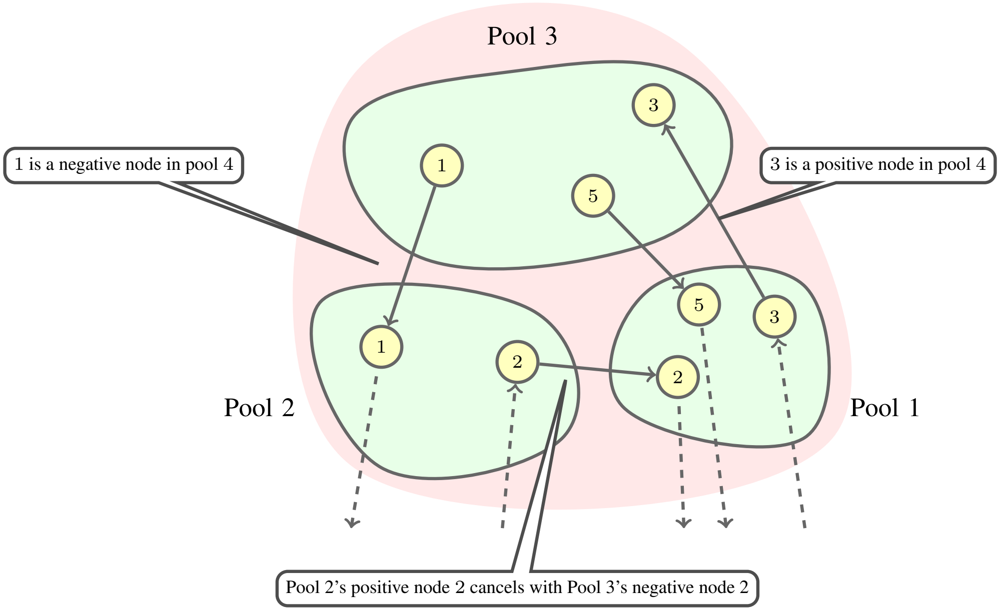

Figure 3 Part of an *IND-END-OF-LINE* instance. The yellow circles indicate the active nodes of each pool; for example $A_1^{(4)} = A_3^{(4)} = A_5^{(4)} = 1$ and $A_m^{(4)} = 0$ for all other m. The pink area indicates the predecessors of pool 4; $P_1^{(4)} = P_2^{(4)} = 1$. The solid arrows indicate the matching $M^{(4)}$ for pool 4, while the dashed arrows indicate that matchings for pools 1 and 2. For example $M_{4,1}^{(4)} = (+, 2, 1)$ and $M_{2,1}^{(4)} = (-, 4, 1)$. Positive nodes are nodes which correspond to positive monomials in the uPC proof, while negative nodes correspond to negative monomials.

▶ **Theorem 11.** *For any unsatisfiable CNF formula F,*

- *If there is a depth-d reduction from S_F to an instance of IND-END-OF-LINE on s variables then there is a degree-$O(d)$ and size-$s^3 2^{O(d)}$ uPC proof of F.*
- *If F has a size-s and degree-d uPC proof of F then there is a depth-$O(d)$ reduction from S_F to an instance of IND-END-OF-LINE on $O(s^2)$-many variables.*

In particular, IND-END -OF-LINE$^{dt}(\mathsf{S}_F) = \Theta(\mathsf{uPC}(F))$.

A proof of this theorem is given in the Appendix.

Unary DAG-Like Sherali-Adams.

The *unary dag-like Sherali-Adams* proof system is a generalization of the uPC proof system and the Sherali-Adams proof system (see e.g., [18] for a definition), which allows one to introduce additional conical juntas at each step in the proof. A *conical junta* is a polynomial of the form $\mathcal{J} = \sum \lambda_i D_i$ where $\lambda_i \geq 0$ and D_i is of the form $\prod_{i \in S} x_i \prod_{j \in T}(1 - x_j)$ for some $S, T \subseteq [n]$. Formally, unary dag-like Sherali-Adams (uDSA) proves that an unsatisfiable CNF formula F is unsatisfiable by deriving the contradiction $-1 \geq 0$ from the equations $\{\overline{C}_i(x) = 0, -\overline{C}_i(x) = 0 : C_i \in F\}$ using the *addition* and *multiplication by a variable* rules from uPC along with the following addition rule:

- *Junta Rule.* From a previously derived polynomial $p \geq 0$, derive $p + \mathcal{J} \geq 0$ for ay conical junta $\mathcal{J}$.

As before, we work over the ideal $\langle x_i^2 = x_i \rangle_{i\in[n]}$, multi-linearizing implicitly. We measure the degree of a uDSA proof by the maximum degree of any polynomial derived, and the size as the sum of the sizes of the polynomials derived, where coefficients are written in unary.

Using the characterization of unary Sherali-Adams by the PPADS complete problem *SINK-OF-LINE*, we can define a TFNP subclass IND-PPADS whose complete problem *IND-SINK-OF-LINE* will characterize uDSA. *IND-SINK-OF-LINE* restricts the solutions of *IND-END-OF-LINE* to permit nodes occurring at the head of arrows to be incorrectly matched. This corresponds to allowing one to introduce positive monomials (and thus conical juntas) free-of-charge in the uDSA proof. Formally, we replace the matching solutions with the following:

- *Matching Solutions*.* A triple $(\ell, \ell', m) \in [L]^2 \times [N]$ such that m is an active node of ℓ' and either (a) ℓ' is a predecessor of ℓ and m is matched to some node m'' of some pool ℓ'' but m'' is not matched back to m, or (b) $\ell' = \ell$ and m occurs at the tail of an arrow in the matching for ℓ and m is matched to a node which is not matched back to it. That is, $A_m^{(\ell')} = 1$ and either
  (a) $P_{\ell'}^{(\ell)} = 1$ and $M_{\ell',m}^{(\ell)} = (\alpha, \ell'', m'')$, but either $A_{m''}^{(\ell'')} = 0$ or $M_{\ell'',m''}^{(\ell)} \neq (\beta, \ell', m)$, where β is the opposite sign of α, or
  (b) $\ell = \ell'$ and $M_{\ell,m}^{(\ell)} = (-, m'', \ell'')$ for some $m'' \in [N], \ell'' < \ell$ and $M_{\ell'',m''}^{(\ell)} \neq (+, \ell', m)$ or $P_{\ell''}^{(\ell)} = 0$.

We also add the following solution[5], which requires that the node in the final line occurs at the tail of an arrow (is negative) in $M^{(L)}$.

- *Final Pool Solution.* A pair $(L, 1)$ such that $M_{L,1}^{(L)} = (+, \ell', m)$ for some $\ell' \leq \ell$ and $m \in [N]$.

One can obtain a characterization theorem of uDSA by *IND-SINK-OF-LINE* (analogous to Theorem 11) by combining by combining the proof of Theorem 11 with the proof of the characterization of uSA by *SINK-OF-LINE* from [25].

3 Communication TFNP and Monotone Circuit Complexity

In addition to proof system characterizations of black-box TFNP problems, the *communication* versions of TFNP problems have provided characterizations of monotone circuit models [26, 32, 45]. 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, bounds on the $\mathbb{F}_2$-Nullstellensatz proof system, which is characterized by black-box PPA were lifted to communication-PPA lower bounds, which characterizes $\mathbb{F}_2$-monotone span programs [40]. Converseley, as described in the introduction, a black-box and communication characterization of the same TFNP subclass generically gives rise to a monotone interpolation theorem, translating small proofs in the associated proof system into efficient computations in the associated model of computation.

In this section, we give generic conditions under which a monotone circuit model has a communication-TFNP characterization. We will formalize monotone circuit models as complexity measures on *partial* monotone functions. 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 will allow us to give an exact characterization of when a complexity measure on partial functions has a TFNP characterization, proving Theorem 3. Since complexity measures on *total* functions induce complexity measures

[5] Note that we could have added this final pool solution to our definition of *IND-END-OF-LINE* without changing its complexity. Indeed, this solution just enforced that the final line is -1 in the uPC proof, which can be assumed without loss of generality, and thus *IND-END-OF-LINE* with the final pool solution reduces to *IND-END-OF-LINE*.

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 in this section 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.

3.1 Communication TFNP

For n bit strings x and x', we say that x' dominates x, written $x \leq x'$, if $x_i \leq x'_i$ for every $i \in [n]$. A *partial* Boolean function f on n bit strings is described by two disjoint sets of inputs, No_f which is the set of strings that f rejects, and Yes_f, the strings that it accepts. f is *total* if $\mathsf{No}_f \cup \mathsf{Yes}_f = \{0,1\}^n$. A partial Boolean function f is monotone if whenever $x \in \mathsf{No}_f$ and $x' \leq x$, then $x' \in \mathsf{No}_f$ and whenever $y \in \mathsf{Yes}_f$ and $y \leq y'$ then $y' \in \mathsf{Yes}_f$. For partial functions f and g, we say f is *solved* by g if $\mathsf{No}_f \subseteq \mathsf{No}_g$ and $\mathsf{Yes}_f \subseteq \mathsf{Yes}_g$. That is, g contains f as a sub-function.

Let $h : \{0,1\}^n \to \{0,1\}^{n'}$, and let f be a partial function on n'-bit inputs. Then $f \circ h$ is the partial function where $\mathsf{Yes}_{f \circ h} = \{x | h(x) \in \mathsf{Yes}_f\}$ and $\mathsf{No}_{f \circ h} = \{x | h(x) \in \mathsf{No}_f\}$. If h is monotone in its input, and f is monotone, then $f \circ h$ is monotone.

3.1.0.1 Monotone Partial Function Complexity Measures.

A monotone partial function complexity measure mpc is a map from partial monotone functions to non-negative integers that is *Monotone Under Solutions*: whenever g solves f, $\mathsf{mpc}(g) \geq \mathsf{mpc}(f)$.[6] 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.

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 communication problem where one player has $x \in \mathsf{No}_f$ the other has $y \in \mathsf{Yes}_f$ and the output is a position i so that $x_i \neq y_i$. Similarly, for a 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 [32] showed that communication complexity of KW_f (mKW_f) is an exact characterization of the (monotone) circuit depth needed to compute f, or equivalently communication-FP.

3.1.0.2 Communication TFNP.

Consider relational communication problems defined by a predicate $R \subseteq X \times Y \times [\ell]$. The corresponding communication problem has one player given $x \in X$, the other $y \in Y$, and the goal being to output an index 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 $R \subseteq X \times Y \times [\ell]$ *mapping reduces* to another $R' \subseteq X' \times Y' \times [\ell']$ with communication t if there are functions $M_X : X \to X'$, $M_Y : Y \to Y'$ and a

[6] Recall that a partial function g solves f if $\mathsf{No}_f \subseteq \mathsf{No}_g$ and $\mathsf{Yes}_f \subseteq \mathsf{Yes}_g$.

t-bit communication protocol $S(x, y, i')$ which outputs i so that

$$R'(M_X(x), M_Y(y), i') \implies R(x, y, S(x, y, i')).$$

In particular this means that R requires at most t more bits of communication than R' to solve. We say that two communication problems R, R' are *equivalent* under t-bit mapping reductions if they t-bit mapping reduce to each other.

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

▶ **Lemma 12.** *For any search problem $R \subseteq X \times Y \times [\ell]$ in t-bit communication* TFNP*, there is a partial function F, on $2^t\ell$ many variables, such that R is equivalent to mKW_F under t-bit mapping reductions.*

Proof. Let $S(x, y, j)$ be a t-bit protocol that verifies that $j \in [\ell]$ is a valid solution on input (x, y). We define a partial function F on $N = 2^t\ell$ input bits. We think of each coordinate as representing a solution $j \in [\ell]$ and a communication pattern for $S(x, y, j)$. We then construct the accepting and rejecting sets for F; for each $x \in X$ we construct an input $\alpha^{(x)} \in \{0, 1\}^N$ in No_F as follows: for each $j \in [\ell]$ and t-bit communication pattern $p \in \{0, 1\}^t$ we set

$$\alpha^{(x)}_{(j,p)} = \begin{cases} 1 & \text{if there is a } y \in Y \text{ such that } S(x, y, j) \text{ evolves according to } p \text{ and } S(x, y, j) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

To construct Yes_F we build an input $\beta^{(y)} \in \{0, 1\}^N$ in the same way, except we reverse 0 and 1:

$$\beta^{(y)}_{(j,p)} = \begin{cases} 0 & \text{if there is a } x \in X \text{ such that } S(x, y, j) \text{ evolves according to } p \text{ and } S(x, y, j) = 1, \\ 1 & \text{otherwise.} \end{cases}$$

We claim that mKW_F is equivalent to R, using this construction as the map. Let j be a solution to R on input (x, y). We simulate $S(x, y, j)$ and output j together with the communication pattern p for the simulation. This gives an index (j, p) such that $\alpha^{(x)}_{(j,p)} = 1 > 0 = \beta^{(y)}_{(j,p)}$, which is a solution to mKW_F on input $(\alpha^{(x)}, \beta^{(y)})$. In the reverse direction, if we are given a bit (j, p) such that $\alpha^{(x)} > \beta^{(y)}$, then we know that $S(x, y, j)$ accepts, and we can return j. ◀

Thus, we can restrict attention to instances of the monotone Karchmer-Wigderson search problem. Analogous to black-box TFNP, we measure the *complexity* of reducing one search problem to another as the amount of communication needed together with the logarithm of the number of bits of the resulting input (up to a constant). Formally, let $R_n \subseteq X_n \times Y_n \times [\ell_n]$ be a sequence of TFNP^{cc}-problems where $X_n, Y_n \subseteq \{0, 1\}^{\mathsf{poly}(n)}$ and $\ell_n = \mathsf{poly}(n)$. Define the *complexity measure* R^{cc} on monotone partial Boolean functions f as

$$R^{cc}(\mathsf{mKW}_f) := \min \log n + t,$$

over the set of n, t so that mKW_f mapping reduces to R_n with t-bits of communication. We say that a family of TFNP^{cc} problems R *characterizes* a mpc if $R^{cc}(\mathsf{mKW}_f) = \log^{\Theta(1)} \mathsf{mpc}(f)$ for every monotone function f.

We will also need the following notion which will essentially allow us to pad a search problem. Say that the sequence R_n is *paddable* if there is a quasi-polynomial function p and a function $t(n) = \mathsf{polylog}(n)$ so that R_n is $t(n')$-communication reducible to $R_{n'}$ for all $n' \geq p(n)$. The condition that the sequence R_n be paddable looks a bit artificial at first. However, if we drop it, we would allow totally unrelated TFNP subclasses to be used in a characterization, e.g., a class that is essentially PPA for infinitely many sizes and suddenly switches to the pigeon-hole principle, and back again. Or have all of TFNP by slowly introducing TFNP problems into the sequence in a non-computable way. So we think natural subclasses of TFNP with complete problems will have the paddable property.

In the remainder of this section we will prove Theorem 3. We will first give conditions for a TFNP^{cc} characterization which involve a stronger notion of a universal family of functions, which we will call *complete families* (Theorem 13). Using this, we then weaken the requirement of having a complete family to admitting a *universal family* (Theorem 17), which gives Theorem 3. In between, we explore sufficient conditions for TFNP^{cc}-characterizations of total functions.

3.2 Complete Problems give TFNP Characterizations

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

i) *Closed Under Reductions.* Say that an mpc is closed under reductions if for any $h : \{0,1\}^n \to \{0,1\}^{n'}$ that is computable by monotone Boolean circuits of depth d, and any partial monotone function f on n' bit inputs, $\mathsf{mpc}(f \circ h) \leq \mathsf{poly}(n, n', \mathsf{mpc}(f), 2^d)$.

ii) *Admits a Complete Family.* A complete family for an mpc is a family F_m of partial functions on $N(m) \leq \mathsf{quasipoly}(m)$ bit inputs such that for every partial monotone function f with $\mathsf{mpc}(f) \leq m$, there is a $\mathsf{polylog}(m)$-depth monotone circuit computing a function h so that $F_m \circ h$ solves f, and $\mathsf{mpc}(F_m) \leq \mathsf{quasipoly}(m)$.[7]

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

▶ **Theorem 13.** *Let* mpc *be a complexity measure. Then there is a paddable sequence of* TFNP *communication problems R_n which characterizes* mpc *iff (i) and (ii) hold. Moreover, the sequence R_n can be made explicit (i.e., computably described) iff the sequence of complete functions for f can be made explicit.*

To prove this, we will use the following lemma which says that reductions between monotone Karchmer Wigderson games and monotone reductions between functions are identical. Note that while this is intuitive and has a simple proof, the proof does not seem to extend to non-monotone complexity. This might be an important distinction between monotone and non-monotone circuit complexity.

▶ **Lemma 14.** *Let f and g be monotone partial Boolean functions. Then mKW_f has a communication-t mapping reduction to mKW_g iff there is a function h computable by a depth-t monotone circuit so that $g \circ h$ solves f.*

Proof. As before, let $\mathsf{Yes}_f, \mathsf{No}_f$ and $\mathsf{Yes}_g, \mathsf{No}_g$ be the set of accepting and rejecting inputs of f and g respectively.

For the if direction, suppose that there is a function h computable by depth-t monotone circuits such that $g \circ h$ solves f. From this, we define a reduction from mKW_f to mKW_g as follows: first, we let h be both M_X and M_Y; it remains to define S. Since $g \circ h$ solves f, for every $(x, y) \in \mathsf{No}_f \times \mathsf{Yes}_f$, we have $(h(x), h(y)) \in \mathsf{No}_g \times \mathsf{Yes}_g$. Thus, $(h(x), h(y))$ is a valid input to mKW_g. A solution to

[7] Note that in the definition of admitting a complete family are insisting that f reduce to F_m for an m only dependent on its complexity, not its input size. Most natural notions of circuit complexity have circuit size be always at least the number of bits the function actually depends on, and the reduction can ignore the irrelevant bits, so this should not usually be a problem.

mKW_g on this input is a bit position i such that $h(x)_i < h(y)_i$. Let h_i be the partial function, defined on inputs in $\mathsf{No}_f \cup \mathsf{Yes}_f$, which outputs the i-th bit of h. Since h is computable by depth-t monotone circuits, so is h_i. Thus, by the Karchmer-Wigderson transformation [32], there is a t-bit communication protocol $S_i(x, y)$ for mKW_{h_i}. Following this protocol on any input (x, y) for which $h(x)_i < h(y)_i$ will output a position j such that $x_j < y_j$, which is a solution to mKW_f. Thus, we can define S as follows: on input (x, y, i) it runs $S_i(x, y)$ and outputs the answer.

Conversely, suppose that we have a t-bit communication reduction $M_X, M_Y, S(x, y, i)$ from mKW_f to mKW_g. From the protocol S, which maps solutions i to mKW_g on input $M_X(x), M_Y(y)$ back to solutions $S(x, y, i)$ to mKW_f on input (x, y), we construct a function h computable with depth-t monotone circuits such that $g \circ h$ solves f. For each i, consider the monotone partial function H_i whose *no*-inputs are the x for which there is an $x \leq x'$ with $x' \in \mathsf{No}_f$ and $M_X(x')_i = 0$, and whose *yes*-inputs are those y for which there is $y \leq y'$ with $y' \in \mathsf{Yes}_f$ and $M_X(y')_i = 1$; we call such an input pair a *dominating and dominated pair* for H_i.

By the definition of reduction, whenever $x' \in \mathsf{No}_f, M_X(x')_i = 0, y' \in \mathsf{Yes}_f$ and $M_Y(y')_i = 1$, the communication protocol $S(x', y', i)$ returns a position j with $x'_j < y'_j$. Given any input pair (x, y) to mKW_f where there is a dominating and dominated pair (x', y') for H_i as above, the parties can, without communication, find x' and y' respectively and then run the protocol $S(x', y', i)$ to obtain the index j. By definition, $x_j \leq x'_j < y'_j \leq y_j$, so this modified protocol solves the mKW_{H_i} game. Therefore, by the Karchmer-Wigderson transformation [32], there is a depth-t monotone circuit computing a function h_i that rejects all $x \in \mathsf{No}_f$ with $M_X(x)_i = 0$ and accepts all $y \in Y_f$ with $M_Y(y)_i = 1$; it follows that $h_i(x) \leq M_X(x)_i$ for all $x \in \mathsf{No}_f$, and if $y \in \mathsf{Yes}_f$ then $M_Y(y)_i \leq h_i(y)$. Letting $h = (h_1, \ldots, h_n)$, where n is the number of input bits to f, we have that for each $x \in \mathsf{No}_f$, $h(x) \leq M_X(x) \in \mathsf{No}_g$, so by monotonicity of g, $h(x) \in \mathsf{No}_g$. Similarly, if $y \in \mathsf{Yes}_f$, $M_X(y) \leq h(y)$ and $h(y) \in \mathsf{Yes}_g$. Thus, $g \circ h$ solves f and g is computable by depth-t monotone circuits. ◀

We will now use the lemma to prove the theorem.

Proof of Theorem 13. Let mpc be a complexity measure with properties (i) and (ii) and let F_m be the complete family of partial monotone functions guaranteed by (ii). Let $R_m := \mathsf{mKW}_{F_m}$ be the monotone Karchmer-Wigderson game for F_m. Observe that as F_m is complete, it reduces to $F_{m'}$ for all $m' \geq \mathsf{mpc}(F_m) = \mathsf{quasipoly}(m)$ via depth-polylog(m') reductions. Thus by Lemma 14, $R_n = \mathsf{mKW}_{F_m}$ reduces to $R_{m'} = \mathsf{mKW}_{F_{m'}}$ with communication-polylog(m') for all such m', and so R is paddable.

We claim $R^{cc}(\mathsf{mKW}_f) = \log^{\Theta(1)} \mathsf{mpc}(f)$ for every monotone partial function f. Letting $m = \mathsf{mpc}(f)$, f reduces to F_m with a polylog(m)-depth monotone circuit, as F_m is complete. Then by Lemma 14, mKW_f reduces to mKW_{F_m} with polylog(m) bits of communication. It follows by definition that $R^{cc}(\mathsf{mKW}_f) \leq \mathsf{polylog}(m) = \mathsf{polylog}(\mathsf{mpc}(f))$. In the other direction, let $R^{cc}(\mathsf{mKW}_f) = M$. Then there are n, t with $t + \log n = M$ so that mKW_f is t-communication reducible to mKW_{F_n}. By Lemma 14, it follows that $F_n \circ h$ solves f for some depth-t circuit h. Then by monotonicity under solutions, and closure under reductions,

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

Next we prove the converse direction of the theorem. Let R_n be any paddable sequence of communication TFNP problems and define a monotone partial function complexity measure mpc as

$$\mathsf{mpc}(f) := 2^{R^{cc}(\mathsf{mKW}_f)}$$

for every monotone partial function f. By construction, mpc is monotone under solutions. We will show that mpc has the properties (i) and (ii). First, assume $g \circ h$ solves f and h is computable by depth-t monotone circuits. Then by Lemma 14, mKW_f has a t-bit reduction to mKW_g. As well, mKW_g has a t' bit reduction to R_n where $t' + \log n = R^{cc}(\mathsf{mKW}_g)$. Stringing these together, f has a $t + t'$ bit reduction to R_n, and so $R^{cc}(\mathsf{mKW}_f) \le t + t' + \log n = t + R^{cc}(\mathsf{mKW}_g)$, and $\mathsf{mpc}(f) \le 2^t \mathsf{mpc}(g)$. Therefore, mpc is closed under reductions.

Finally, we give a complete family for mpc. Let F_N be the sequence of partial monotone functions given by Lemma 12 such that R_N is equivalent to mKW_{F_N}. Note that by definition F_N has at most $N2^t$ many input bits where $t = \mathsf{polylog}(N)$ is the number of bits that need to be communicated in order to verify solutions to R_N, and also that $\mathsf{mpc}(F_N) = 2^{R^{cc}(\mathsf{mKW}_{F_N})} \le 2^t = \mathsf{quasipoly}(N)$.

We will show that for each m, there is an $N' = \mathsf{quasipoly}(m)$ so that every partial function f with $\mathsf{mpc}(f) \le m$ reduces to $F_{N'}$ via a $\mathsf{polylog}(m)$-depth reduction. Fix some f with $\mathsf{mpc}(f) \le m$ and let $M = \log \mathsf{mpc}(f) = R^{cc}(\mathsf{mKW}_f)$. Then mKW_f reduces to some R_n in t bits of communication, where $t + \log n = M$; in particular, t is at most M and $\log n \le M$. Then by paddability, we can reduce this to some $R_{N'}$ where $N' = \mathsf{quasipoly}(n) \le \mathsf{quasipoly}(M)$ is a fixed function of m, and the further communication is at most $\mathsf{polylog}(M)$. Then by Lemma 14, f has a $\mathsf{polylog}(M)$-depth circuit reduction to $F_{N'}$ as desired. Thus, mpc is closed under reductions and admits a complete family. ◀

A Partial Characterization for Complexity Measures on Total Functions

Analogous to measures on partial functions, let a *monotone (total function) complexity measure* mc map total monotone functions to non-negative integers. From any mc we can extract a monotone complexity measure mpc on partial functions 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 $\mathsf{mc}(f) \le m$, there is a $\log m$-depth monotone circuit computing a function h so that $F_m \circ h$ solves f, and $\mathsf{mc}(F_m) \le \mathsf{poly}(m)$.

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 15.** mpc *is closed under reductions and has a complete (partial function) family if and only if* mc *is closed under reductions and has a complete total function family.*

An immediate consequence is the following.

▶ **Corollary 16.** *If a monotone complexity measure* mc *is closed under reductions and has a complete family, then it has a* TFNP^{cc} *characterization by a sequence of paddable relations. If not,* mc *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 15. To prove the lemma, we will first assume mc is closed under reductions, e.g., $\mathsf{mc}(f \circ h) \le \mathsf{poly}(\mathsf{mc}(f), 2^d)$ 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 $\mathsf{mpc}(F \circ h) \le \mathsf{mc}(f \circ h) \le \mathsf{poly}(\mathsf{mc}(f), 2^d) = \mathsf{poly}(\mathsf{mpc}(F), 2^d)$. Conversely, since $\mathsf{mpc}(f) = \mathsf{mc}(f)$ for total functions, it follows immediately that if mpc is closed under reductions, then so is mc.

If F_m is a family of complete partial functions for mpc, let f_m be the corresponding minimal complexity total functions solving F_m. Note that $\mathsf{mc}(f_m) = \mathsf{mpc}(F_m) = \mathsf{quasipoly}(m)$. Let g be any total function and let $m = \mathsf{mpc}(g) = \mathsf{mc}(g)$. Then there is a function h computable by $\mathsf{polylog}m$-depth monotone circuits such that $F_m \circ h$ solves h. Furthermore, $f_m \circ h$ solves $F_m \circ h$, and so $f_m \circ h$ solves g. However, the only way for one total function to solve another is if they are equal, so $f_m \circ h = g$. It follows that f_m is also complete and, by assumption, is total.

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

3.3 Universal Functions vs. Complete Functions

We can simplify the condition that there be complete functions in the class to having *universal families* of functions, replacing (ii) in Theorem 17 by the following:

ii†) *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 17.** *Let* mpc *be a monotone partial function complexity measure satisfying (i) and (ii). Then* mpc *admits a universal family if and only if it admits a complete family.*

Using Lemma 15, we can derive an analogous statement to Corollary 16 for total functions as well. Next, we state Theorem 3 formally, which follows immediately from Theorem 17 and Theorem 13.

▶ **Theorem 3.** Let mpc be a complexity measure. Then there is a paddable sequence of TFNP communication problems R_n which characterizes mpc iff (i) and (ii†) hold. Moreover, the sequence R_n can be made explicit (i.e., computably described) iff the sequence of complete functions for f can be made explicit.

Proof of Theorem 17. If there is a universal family F_m for mpc then we can let $G_m = F_m$ since as mentioned above, F_m is complete under depth 0 reductions.

Conversely, say that a monotone partial complexity measure mpc admits a complete family under $d(m)$-depth reductions if there exists a family G_m of functions such that $\mathsf{mpc}(G_m) \leq 2^{d(m)}$ and for every partial monotone function f with $\mathsf{mpc}(f) \leq m$, there is a depth-$d(m)$ monotone circuit computing a function h so that $G_m \circ h$ solves f. Suppose that $G_m(x)$ is complete under depth $d(m)$ reductions, where the input size $|x| = M \leq \mathsf{poly}(m)$. 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 computable by monotone circuits of 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 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. 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?
4. 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.)

References

1 Albert Atserias and Moritz Müller. Automating resolution 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. doi:10.1137/130914085.

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. doi:10.1006/jcss.1998.1575.

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. doi:10.1145/3060145.

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. doi:10.2307/2275569.

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. doi:10.1109/CCC.2004.1313795.

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. doi:10.1017/jsl.2013.37.

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. doi:10.1145/2811255.

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. doi:10.1145/3532737.3532746.

13 Susanna F. de Rezende, Massimo Lauria, Jakob Nordström, and Dmitry Sokolov. The power of negative reasoning. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 40:1–40:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CCC.2021.40.

14 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. doi:10.1109/FOCS46700.2020.00011.

15 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. URL: https://eccc.weizmann.ac.il/report/2021/006.

16 Noah Fleming. *The Proof Complexity of Integer Programming*. PhD thesis, University of Toronto, Canada, 2021. URL: http://hdl.handle.net/1807/108797.

17 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. doi:10.4230/LIPIcs.ITCS.2022.69.

18 Noah Fleming, Pravesh Kothari, and Toniann Pitassi. Semialgebraic proofs and efficient algorithm design. *Found. Trends Theor. Comput. Sci.*, 14(1-2):1–221, 2019. doi:10.1561/0400000086.

19 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. doi:10.1145/3486680.

20 Anna Gál. A characterization of span program size and improved lower bounds for monotone span programs. *Comput. Complex.*, 10(4):277–296, 2001. doi:10.1007/s000370100001.

21 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. doi:10.1145/3188745.3188838.

22 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.

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

24 Paul W. Goldberg and Christos H. Papadimitriou. Towards a unified complexity theory of total functions. *Electron. Colloquium Comput. Complex.*, page 56, 2017. URL: https://eccc.weizmann.ac.il/report/2017/056.

25 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. arXiv:2205.02168, doi:10.48550/arXiv.2205.02168.

26 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. doi:10.4230/LIPIcs.ITCS.2019.38.

27 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. doi:10.1145/3357713.3384248.

28 Mika Göös, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles are nonnegative juntas. *SIAM J. Comput.*, 45(5):1835–1869, 2016. doi:10.1137/15M103145X.

29 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM J. Comput.*, 47(6):2435–2450, 2018. doi:10.1137/16M1059369.

30 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. doi:10.1109/FOCS.2017.20.

31 Pritish Kamath. *Some hardness escalation results in computational complexity theory*. PhD thesis, Massachusetts Institute of Technology, 2019.

32 Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM J. Discret. Math.*, 3(2):255–265, 1990. doi:10.1137/0403021.

33 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. doi:10.1145/3055399.3055438.

34 Jan Krajícek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. Symb. Log.*, 62(2):457–486, 1997. doi:10.2307/2275541.

35 Jan Krajícek. Interpolation by a game. *Math. Log. Q.*, 44:450–458, 1998. doi:10.1002/malq.19980440403.

36 Jan Krajícek. Randomized feasible interpolation and monotone circuits with a local oracle. *J. Math. Log.*, 18(2):1850012:1–1850012:27, 2018. doi:10.1142/S0219061318500125.

37 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. doi:10.1145/2746539.2746599.

38 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. doi:10.1137/S0895480192233867.

39 Shachar Lovett, Raghu Meka, Ian Mertz, Toniann Pitassi, and Jiapeng Zhang. Lifting with sunflowers. 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 104:1–104:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.104.

40 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. doi:10.1145/3188745.3188914.

41 Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997. doi:10.2307/2275583.

42 Pavel Pudlák. On the complexity of finding falsifying assignments for herbrand disjunctions. *Arch. Math. Log.*, 54(7-8):769–783, 2015. doi:10.1007/s00153-015-0439-6.

43 Pavel Pudlák and Jirí Sgall. Algebraic models of computation and interpolation for algebraic proof systems. In Paul Beame 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. doi:10.1090/dimacs/039/15.

44 Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Comb.*, 19(3):403–435, 1999. doi:10.1007/s004930050062.

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

46 Robert Robere. Separations in proof complexity and TFNP. Talk at the Satisfiability: Theory, Practice, and Beyond Reunion, Simons Institute, Berkeley, 2022.

47 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. doi:10.1109/FOCS.2016.51.

Appendix: Proof of Theorem 11

In this appendix we prove Theorem 11, which we break into the following two lemmas. Recall that the *length* of a uPC proof is the number of lines (deductions) in the proof.

▶ **Lemma 18.** *Let F be an unsatisfiable CNF formula on n variables. If there is a* uPC *proof of F with size-s, length-L, and degree-d then there is a depth-$O(d)$ decision-tree reduction from S_F to an instance of IND-END-OF-LINE on $O(sL)$ many variables.*

Proof. Fix a unary Polynomial Calculus proof Π of some unsatisfiable CNF formula F. For each monomial m, let c_m be the maximum absolute value of any coefficient of m that occurs in Π, and define $N := \sum_m c_m$. We will have c_m nodes for monomial m and implicitly identify any of these c_m nodes with the monomial m. We define an *IND-END-OF-LINE* instance on L pools and N nodes in much the same way as we did for $\mathbb{F}_2$-PC.

For each $\ell \in [L]$, we define the active nodes $m \in [N]$ for pool ℓ as follows. If monomial m occurs in the ℓ-th line of Π with coefficient c, let $m_1, \ldots, m_c$ be the first c nodes corresponding to copies of monomial m and set $A^{(\ell)}_{m_i} = m(x)$ for all $i \in [c]$. Fix $A^{(\ell)}_{m'} = 0$ for the remaining nodes $m' \in [N] \setminus \{m_1, \ldots, m_c\}$. Note that as m is a monomial of degree $\leq d$, $m(x)$ can be computed by a depth-d decision tree.

If line ℓ is derived by addition from two lines ℓ', ℓ'', set $P^{(\ell)}_{\ell'} = P^{(\ell)}_{\ell''} = 1$ and $P^{(\ell)}_{\ell^*} = 0$ for all $\ell^* \neq \ell', \ell''$. If ℓ was derived from ℓ' by multiplication by some variable x_i set $P^{(\ell)}_{\ell'} = x_i$ and $P^{(\ell)}_{\ell^*} = 0$ for all $\ell^* \neq \ell'$.

Finally, for each $\ell \in [L]$ we define the matching $M^{(\ell)}$ as follows. For this it will be convenient to think of each line ℓ in Π as a multi-set of monomials, each with an associated positive or negative coefficient, and a corresponding node in N. There are three cases:

Case 1. If ℓ was derived by *addition* from some $\ell', \ell'' < \ell$ then every monomial m in line ℓ comes from one of ℓ', ℓ'' — suppose that m comes from ℓ' — and so we match m to the copy of m in ℓ'. If m has a positive coefficient in ℓ, then we set $M^{(\ell)}_{\ell,m} = (+, \ell', m)$ and $M^{(\ell)}_{\ell',m} = (-, \ell, m)$, and if it has a negative coefficient we set $M^{(\ell)}_{\ell,m} = (-, \ell', m)$ and $M^{(\ell)}_{\ell',m} = (-, \ell, m)$.

It remains to define the matchings for all monomials m which occur in ℓ' or ℓ'' but not in ℓ; suppose that m belongs to ℓ'. For this to happen, m must have cancelled with a negative coefficient copy of itself in ℓ'' and so we match them. That is, if m occurs positively in ℓ' then we set $M^{(\ell)}_{\ell',m} = (-, \ell'', m)$ and $M^{(\ell)}_{\ell'',m} = (+, \ell', m)$, and if it occurs negatively then we set $M^{(\ell)}_{\ell',m} = (+, \ell'', m)$ and $M^{(\ell)}_{\ell'',m} = (-, \ell', m)$. The matching variables for the remaining nodes (which do not correspond to monomials occurring in lines ℓ, ℓ', ℓ'') can be set arbitrarily.

Case 2. If ℓ was derived by *multiplication* by a variable x_i from some $\ell' < \ell$ then for every monomial m in line ℓ, there must be a monomial $m' = m \setminus x_i$ or $m' = m$ belonging to ℓ' from which it was derived. If m is positive in ℓ then match $M^{(\ell)}_{\ell,m} = (+, \ell', m')$ and $M^{(\ell)}_{\ell',m'} = (-, \ell, m)$, and if m is negative in ℓ then $M^{(\ell)}_{\ell,m} = (-, \ell', m')$ and $M^{(\ell)}_{\ell,m} = (+, \ell, m)$. Finally, we match the remaining nodes corresponding to monomials in ℓ' that have yet to be matched. Each of these remaining monomials must have cancelled after multiplication by x_i so as to not appear in ℓ. The only cancellations which can occur are pairs (m, mx_i) such that m does not contain x_i and m and mx_i occur with different signs in ℓ'. Suppose that m occurs positively in ℓ' then we match $M^{(\ell)}_{\ell',m} = (-, \ell', mx_i)$ and $M^{(\ell)}_{\ell',mx_i} = (+, \ell', m)$, and similarly if m occurred negatively then we match $M^{(\ell)}_{\ell',m} = (+, \ell', mx_i)$ and $M^{(\ell)}_{\ell',mx_i} = (-, \ell', m)$. The remaining nodes (which do not correspond to nodes in ℓ or ℓ') may be matched arbitrarily.

Case 3. If ℓ is an axiom of F — that is, ℓ is $\overline{C}$ for some $C \in F$ — then for each monomial $m \in \overline{C}$, the matching $M^{(\ell)}_{\ell,m}$ is defined by querying the $\leq d$ variables in $\overline{C}$. If we discover that $\overline{C}(x) = 0$ (that is, C is satisfied) then we fix an arbitrary matching between the positive and negative monomials in $\overline{C}$ which are not set to 0 under x such that each negative monomial is at the tail of some arrow and each positive monomials is at the head of some arrow. Otherwise, if $\overline{C}(x) \neq 0$ then we fix the matching variables arbitrarily (there will always be a solution in this case).

Observe that the only solutions to the constructed *IND-END-OF-LINE* instance occur at the pools $\ell \in [L]$ corresponding to an axioms $C \in F$ for which $C(x) = 0$. Thus, any solution to *IND-END-OF-LINE* will be in a violated clause of F, a solution to S_F. Using this, we can define the output decision trees: for any solution s belonging a pool $\ell \in [L]$ which corresponds to an initial clause $C_i \in F$, the output decision tree T^o_s outputs i. The output decision trees corresponding to the remaining solutions (which do not occur in this instance of *IND-END-OF-LINE*) can be set arbitrarily. ◀

▶ **Lemma 19.** *Let F be an unsatisfiable CNF formula. If S_F reduces to an instance of* IND-END-OF-LINE *on n variables using depth-d decision trees, then there is an degree-$O(d)$ and size $n^3 2^{O(d)}$* uPC *proof of F.*

Proof. Let F be an unsatisfiable CNF formula and suppose that S_F reduces by depth-d decision trees to an *IND-END-OF-LINE* instance on n variables. For each variable x of the *IND-END-OF-LINE* let T_x be the decision tree computing x. As before, we will associate T_x with the polynomial formed by taking a sum over the *accepting* paths in T_x. As well, for each solution s of the *IND-END-OF-LINE* instance let T^o_s be the output decision tree. We will say that a node m which active for ℓ is *positive* if it appears at the head of an arrow in $M^{(\ell)}$ and *negative* otherwise. Recall that for a function f element o in the range of f, $[\![f = o]\!]$ denotes the *indicator polynomial* which is 1 on input x if $f(x) = o$ and 0 otherwise.

For $\ell \in [L]$ define the polynomial

$$q_\ell := \sum_{m \in [N]} A^{(\ell)}_m \left(\sum_{m^* \in [N], \ell^* \leq \ell} \left[\!\!\left[M^{(\ell)}_{\ell,m} = (+, \ell^*, m^*) \right]\!\!\right] - \sum_{m^* \in [N], \ell^* \leq \ell} \left[\!\!\left[M^{(\ell)}_{\ell,m} = (-, \ell^*, m^*) \right]\!\!\right] \right)$$

which records the difference between the number of positive and negative nodes for pool ℓ. We will derive by induction on $\ell = 1, \ldots, L$ that $q_\ell = 0$ and $-q_\ell = 0$. This will complete the proof as for pool L, $A^{(L)}_1 = 1$ and $A^{(L)}_m = 0$ for all $m \neq 1$ and so

$$0 = q_L = \sum_{m^* \in [N], \ell^* \leq L} \left[\!\left[M^{(L)}_{L,1} = (+, \ell^*, m^*) \right]\!\right] - \sum_{m^* \in [N], \ell^* \leq L} \left[\!\left[M^{(L)}_{L,1} = (-, \ell^*, m^*) \right]\!\right].$$

From which we can derive the $1 = 0$ by the following claim, noting that the terms of q_L are exactly the paths in the decision tree for $M^{(L)}_{L,1}$.

▷ Claim 3. Let T be any depth-d decision tree and let $q(x) = \sum_{p \in T} \alpha_p p(x)$, where the sum is taken over (the polynomial representation of) each root-to-leaf path p in T, and $\alpha_p \in \{\pm 1\}$. Then there is a uPC degree-$2d$ and size $O(|T|)$ derivation of $1 = 0$ from $q(x) = 0$ and $-q(x) = 0$.

Proof. From $q = 0$ we will derive $p = 0$ for each $p \in T$. This completes the proof as $\sum_{p \in T} p = 1$ for any decision tree T. For any path $p' \in T$ with $\alpha_{p'} = 1$ observe that $p'q = \sum_{p \in T} \alpha_p p' p = p'$ as any pair of paths $p \neq p'$ contain an opposing literal (i.e., x and $(1 - x)$ for some variable x) and thus sum to 0. Similarly, we can derive $p' = 0$ for any $p' \in T$ with $\alpha_{p'} = -1$ by multiplying $-q = 0$ by p'. ◀

It remains to show that $q_\ell = 0$ can be derived from $q_{\ell'} = 0$ for $\ell' < \ell$. Note that we can derive $-q_\ell = 0$ by a symmetric argument by using $-A(x) = 0$ for each axiom $A(x) = 0$ used in the derivation of $q_\ell = 0$. Our induction will rely on (i) the matching $M^{(\ell)}$, and (ii) the consistencies of polarities — if m is a node of ℓ' which occurs at one end of an arrow in the matching for ℓ', then it must occur at the other end of an arrow in the matching for ℓ, if ℓ' is a predecessor of ℓ. We will represent (i) by the following polynomial which records the difference between the number of positive and negative nodes involved in the matching for pool ℓ

$$\mathsf{deriv}^{(\ell)} := \sum_{\ell' \leq \ell} P^{(\ell)}_{\ell'} \sum_{m \in [N]} A^{(\ell')}_m \left(\sum_{m^* \in [N], \ell^* \leq \ell'} \left[\!\left[M^{(\ell)}_{\ell,m} = (+, \ell^*, m^*) \right]\!\right] - \left[\!\left[M^{(\ell)}_{\ell,m} = (-, \ell^*, m^*) \right]\!\right] \right),$$

where, for convenience of notation, we have introduced an additional variable $P^{(\ell)}_\ell$ which is fixed to 1.

We will represent (ii) by the polynomial

$$\mathsf{consist}^{(\ell)}_{\ell'} = P^{(\ell)}_{\ell'} \sum_{m \in [N]} A^{(\ell')}_m \sum_{\ell^* \leq \ell} \left(\left[\!\left[M^{(\ell)}_{\ell',m} = (-, \ell^*, m^*) \right]\!\right] - \left[\!\left[M^{(\ell)}_{\ell',m} = (+, \ell^*, m^*) \right]\!\right] \right) - P^{(\ell)}_{\ell'} q_{\ell'}.$$

The equation $\mathsf{consist}^{(\ell)}_{\ell'} = 0$ states that the active nodes for line ℓ' must occur with the same polarity in the matching for pool ℓ' as in the matching for pool ℓ. The following claims give short uPC derivations of these polynomials from the axioms.

▷ Claim 4. For any $\ell \in [L]$, $\mathsf{deriv}^{(\ell)} = 0$ has a degree-$O(d)$ and size-$NL2^{O(d)}$ uPC proof from the axioms.

▷ Claim 5. For any $\ell \in [L]$ and $\ell' < \ell$, $\mathsf{consist}^{(\ell)}_{\ell'}$ has a degree-$O(d)$ and size-$NL2^{O(d)}$ uPC proof from the axioms.

Assuming these claims, we show how to derive $q_\ell = 0$ from $q_{\ell'} = 0$ for all $\ell' < \ell$. For each $\ell' < \ell$, sum the polynomial $P^{(\ell)}_{\ell'} q_{\ell'} = 0$ with $\mathsf{consist}^{(\ell)}_{\ell'}$ to deduce

$$P^{(\ell)}_{\ell'} \sum_{m \in [N]} A^{(\ell')}_m \sum_{\ell^* \leq \ell} \left(\left[\!\left[M^{(\ell)}_{\ell',m} = (-, \ell^*, m^*) \right]\!\right] - \left[\!\left[M^{(\ell)}_{\ell',m} = (+, \ell^*, m^*) \right]\!\right] \right) = 0.$$

Summing these polynomials with $\mathsf{deriv}^{(\ell)} = 0$ gives $q_\ell = 0$. We apply Claim 4 $\ell \leq L$ times and Claim 5 once. Thus, this induction step can be performed in degree $O(d)$ and size $NL^2 2^{O(d)}$. ◀

Proof of Claim 4. For $\ell' \leq \ell$, $m \in [N]$ and $\alpha \in \{-,+\}$ define

$$\mathsf{match}^{(\ell)}_{\alpha,m,\ell'} := \sum_{\substack{m^* \in [N],\\ \ell^* \in [\ell]}} \left[\!\left[M^{(\ell)}_{m,\ell'} = (\alpha, m^*, \ell^*)\right]\!\right] \sum_{\gamma,\delta \in \{0,1\}} \left[\!\left[P^{(\ell)}_{\ell^*} = \gamma\right]\!\right] \left[\!\left[A^{(\ell^*)}_{m^*} = \delta\right]\!\right] \cdot \sum_{\substack{\hat{m} \in [N], \hat{\ell} \in [\ell]\\ \beta \in \{-,+\}}} \left[\!\left[M^{(\ell)}_{m^*,\ell^*} = (\beta, \hat{m}, \hat{\ell})\right]\!\right],$$

which records whether node m belonging to ℓ' is at the head or tail of an arrow, and whether it is correctly matched in the matching $M^{(\ell)}$ for ℓ. Note that

$$\sum_{\gamma,\delta \in \{0,1\}} \left[\!\left[P^{(\ell)}_{\ell^*} = \gamma\right]\!\right] \left[\!\left[A^{(\ell^*)}_{m^*} = \delta\right]\!\right] \sum_{\substack{\hat{m} \in [N], \hat{\ell} \in [\ell]\\ \beta \in \{-,+\}}} \left[\!\left[M^{(\ell)}_{m^*,\ell^*} = (\beta, \hat{m}, \hat{\ell})\right]\!\right] = 1, \tag{1}$$

as it is the polynomial obtained from summing over all paths in the stacked decision tree obtained by running the decision trees for $P^{(\ell)}_{\ell^*}$, $A^{(\ell^*)}_{m^*}$ and then $M^{(\ell)}_{m^*,\ell^*}$.

Now, consider the polynomial $P^{(\ell)}_{\ell'} A^{(\ell')}_m \mathsf{match}^{(\ell)}_{\alpha,m,\ell'}$ and partition its terms into two sets, a set $C^{(\ell',m)}_\alpha$ which corresponds to *correct* matchings — that is, m is matched to a node $m^* \in [N]$ belonging to a pool $\ell^* \leq \ell$ ($M^{(\ell)}_{\ell',m} = (\alpha, \ell^*, m^*)$) with $P^{(\ell)}_{\ell^*} = 1$ and $A^{(\ell^*)}_{m^*} = 1$ which is matched back to m, meaning that $M^{(\ell)}_{\ell^*,m^*} = (\gamma, \ell', m)$, where γ is the opposite sign of α — and $E^{(\ell',m)}_\alpha$ which will contain the remaining terms, corresponding to *erroneous* matchings. Using these polynomials, define

$$\mathsf{match}^{(\ell)} := \sum_{\ell' \in [\ell]} \sum_{m \in [N]} A^{(\ell')}_m P^{(\ell)}_{\ell'} \left(\mathsf{match}^{(\ell)}_{+,m,\ell'} - \mathsf{match}^{(\ell)}_{-,m,\ell'}\right),$$

which records the matching for pool ℓ. By (1), this polynomial is equivalent to $\mathsf{deriv}^{(\ell)}$, and therefore it suffices to show that this polynomial has a low-degree derivation from the axioms. To do so, partition the terms of $\mathsf{match}^{(\ell)}$ into three sets, C_+, C_-, E as above, where $C_\alpha = \bigcup C^{(\ell',m)}_\alpha$ for $\alpha \in \{-,+\}$, and $E = \bigcup E^{(\ell',m)}_+ \cup E^{(\ell',m)}_-$ where the unions are taken over $\ell' \leq \ell$ and $m \in [N]$. Observe that because the matchings in C_+ and C_- are correct, for every node at the head of an arrow, a node occurs at the tail of that arrow. It follows that $\sum_{t \in C_+} t - \sum_{t' \in C_-} t' = 0$.

Next, consider a term $t \in E$. This term corresponds to a node m in some pool $\ell' \leq \ell$ that is incorrectly matched; let s be this incorrect matching. We will denote by t_s that the term t witnesses s. Let T^o_s be the output decision tree for solution s and abuse notation by letting T^o_s also denote the polynomial formed by taking the sum over all of the paths in the decision tree T^o_s. Recalling that the sum over all paths in a decision tree is 1,

$$\mathsf{match}^{(\ell)} = \sum_{t \in C_+} t - \sum_{t' \in C_-} t' + \sum_{t_s \in E} t_s = 0 + \sum_{t_s \in E} t_s = \sum_{t_s \in E} t_s \cdot T^o_s.$$

An incorrect matching is a solution to *IND-END-OF-LINE*. Therefore, because this instance solves S_F, any truth assignment x which satisfies t_s must falsify the $T^o_s(x)$-th clause of F. It follows that each term of $t_s \cdot T^o_s$ that is not identically 0 must contain the polynomial $\overline{C} = 0$ for some clause C of F. Thus, $t_s \cdot T^o_s$ can be derived by multiplication from the axioms $\overline{C} = 0$ and $-\overline{C} = 0$. It follows that $\mathsf{deriv}^{(\ell)}$ has a proof of degree at most the degree and size of $\mathsf{match}^{(\ell)}$, which are $6d$ and $NL2^{O(d)}$ respectively. ◀

Proof of Claim 4. For $\alpha \in \{-,+\}$, define the *polarity* polynomial

$$\mathsf{pol}_\alpha^{(\ell')} := P_{\ell'}^{(\ell)} \sum_{m\in[N]} A_m^{(\ell')} \sum_{\ell^* \le \ell', m^* \in [N]} \left[\!\!\left[M_{\ell',m}^{(\ell')} = (\alpha, \ell^*, m^*) \right]\!\!\right] \sum_{\substack{\hat{\ell}\le \ell, \hat{m}\in[N]\\ \beta\in\{-,+\}}} \left[\!\!\left[M_{\ell',m}^{(\ell)} = (\beta, \hat{\ell}, \hat{m}) \right]\!\!\right],$$

which records for each node at the α-end of an arrow in the matching for ℓ', which end of an arrow it occurs at in the matching for pool ℓ'. We will partition the set of terms of this polynomial into two sets, $C_\alpha^{(\ell')}$ and $E_\alpha^{(\ell')}$ $C_\alpha^{(\ell')}$. $C_\alpha^{(\ell')}$ will be the terms t which are the indicators of *correct* assignments of polarities of the nodes in pool ℓ' in the matchings $M^{(\ell)}$ and $M^{(\ell')}$ — that is, if m is an active node for ℓ' and m occurs at the head of an arrow in the matching for $M^{(\ell')}$ then it is at the tail of an arrow in the matching for $M^{(\ell)}$ if ℓ' is a predecessor of ℓ. $E_\alpha^{(\ell')}$ $C_\alpha^{(\ell')}$ will be the remaining terms which correspond to *erroneous* assignments of polarities. As well, observe that

$$\mathsf{pol}_\alpha^{(\ell')} = P_{\ell'}^{(\ell)} \sum_{m\in[N]} A_m^{(\ell')} \sum_{\ell^* \le \ell', m^* \in [N]} \left[\!\!\left[M_{\ell',m}^{(\ell')} = (\alpha, \ell^*, m^*) \right]\!\!\right] \cdot 1,$$

as $\sum_{\hat{\ell}\le\ell, \hat{m}\in[N], \beta\in\{-,+\}} [\![M_{\ell',m}^{(\ell)} = (\beta, \hat{\ell}, \hat{m})]\!]$ is the polynomial obtained by taking a sum over all paths in the decision tree for $M_{\ell',m}^{(\ell)} = (\beta, \hat{\ell}, \hat{m})$, which sums to 1.

Similarly, let

$$\mathsf{pol}_\alpha^{(\ell)} := P_{\ell'}^{(\ell)} \sum_{m\in[N]} A_m^{(\ell')} \sum_{\ell^* \le \ell, m^* \in [N]} \left[\!\!\left[M_{\ell',m}^{(\ell)} = (\alpha, \ell^*, m^*) \right]\!\!\right] \sum_{\substack{\hat{\ell}\le \ell', \hat{m}\in[N]\\ \beta\in\{-,+\}}} \left[\!\!\left[M_{\ell',m}^{(\ell')} = (\beta, \hat{\ell}, \hat{m}) \right]\!\!\right],$$

be the polynomial which records for each active node of ℓ' which occurs at the α-end of an arrow in $M^{(\ell)}$, which end of an arrow it occurs at in $M^{(\ell')}$. Define $C_\alpha^{(\ell)}$ and $E_\alpha^{(\ell)}$ analogously, and note that

$$\mathsf{pol}_\alpha^{(\ell)} = P_{\ell'}^{(\ell)} \sum_{m\in[N]} A_m^{(\ell')} \sum_{\ell^* \le \ell, m^* \in [N]} \left[\!\!\left[M_{\ell',m}^{(\ell)} = (\alpha, \ell^*, m^*) \right]\!\!\right] \cdot 1,$$

by the same reasoning as above.

Putting these together, we have

$$\mathsf{consist}_{\ell'}^{(\ell)} = \mathsf{pol}_-^{(\ell)} - \mathsf{pol}_+^{(\ell)} - \mathsf{pol}_+^{(\ell')} + \mathsf{pol}_-^{(\ell')}.$$

We will derive $\mathsf{pol}_+^{(\ell)} - \mathsf{pol}_-^{(\ell')} = 0$ and $\mathsf{pol}_-^{(\ell)} - \mathsf{pol}_+^{(\ell')} = 0$ separately from the axioms, beginning with $\mathsf{pol}_+^{(\ell)} - \mathsf{pol}_+^{(\ell')} = 0$. Consider any term t in $C_+^{(\ell')}$ and observe that since t is *correct*, it records that an active monomial m of ℓ' which occurs at the head of an arrow in $M^{(\ell')}$ occurs at the tail of an arrow in $M^{(\ell)}$. Thus, t occurs also in $C_-^{(\ell)}$. By a symmetric argument, any term t occurring in $C_-^{(\ell)}$ occurs in $C_+^{(\ell')}$. Thus, $\sum_{t\in C_+^{(\ell')}} t - \sum_{t\in C_-^{(\ell)}} t = 0$, and also $\sum_{t\in C_-^{(\ell')}} t - \sum_{t\in C_+^{(\ell)}} t = 0$ by a similar argument. Denoting the union of all of the error sets by $E := E_+^{(\ell)} \cup E_-^{(\ell)} \cup E_+^{(\ell')} \cup E_-^{(\ell')}$, we have

$$\mathsf{consist}_{\ell'}^{(\ell)} = \Big(\sum_{t\in C_+^{(\ell')}} t - \sum_{t\in C_-^{(\ell)}} t \Big) + \Big(\sum_{t\in C_-^{(\ell')}} t - \sum_{t\in C_+^{(\ell)}} t \Big) + \sum_{t\in E} t = 0 + \sum_{t\in E} t.$$

It remains to show that each term $t \in E$ can be derived from the axioms with a low-degree uPC proof. As each $t \in E$ witnesses a node which switched polarity between the matching for line ℓ' and the matching for line ℓ, this is a solution s to *IND-END-OF-LINE*; we will denote denote t by t_s to record the fact that t witnesses solution s. Let T_s^o be the output decision tree corresponding to solution s, and abuse notation by identifying it with polynomial formed by taking the sum over all paths in T_s^o. As the sum over all paths in a decision tree gives the 1 polynomial, we have $t_s = t_s \cdot T_s^o$. As t_s witnesses solution s, it follows that any assignment x such that $t_s(x) = 1$ must falsify the $T_s^o(x)$-th clause C of F. Thus, $t_s \cdot T_s^o$ can be derived from the axioms $\overline{C} = 0$ and $-\overline{C} = 0$. It follows that

$$\mathsf{consist}_{\ell'}^{(\ell)} = 0 + \sum_{t_s \in E} t_s \cdot T_s^o = 0$$

has a uPC proof from the axioms of degree at most $4d$ and size $NL2^{O(d)}$. ◀