TFNP Characterizations of Proof Systems and Monotone Circuits

Sam Buss
UCSD

Noah Fleming
Memorial University

Russell Impagliazzo
UCSD

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 *lifting* theorems, translate efficient proofs into small circuits and vice versa, enabling tools from one area to be applied to the other.

Recently, the theory of TFNP has emerged as a unifying framework for these connections. For proof systems which admit such a connection there is a TFNP problem which *characterizes* it: the class of problems which are black-box reducible to this TFNP problem is *equivalent* to the set of tautologies which admit short proofs in this proof system. Such characterizations have resulted in proof complexity becoming one of the main tools for analyzing black-box TFNP. Similarly, for certain models of monotone circuitry, the class of functions that can be computed efficiently is equivalent to the total functions that can be reduced to a certain TFNP problem via communication-efficient reduction. Whenever a TFNP problem is characterized by both a proof system and a circuit model, one immediately obtains an interpolation theorem. Similarly, the major lifting theorems are between proof systems and circuit models which characterize the same TFNP problem. This suggests that TFNP could provide a roadmap for the development of further interpolation theorems and lifting theorems.

In this paper we develop a systematic understanding of these connections by uncovering the exact conditions for such characterizations to occur. 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 is characterized by a Cook-Reckhow 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 is characterized by a monotone circuit model with a universal family of functions.

As a proof of concept, we use this to uncover a characterization of Polynomial Calculus and a dag-like variant of Sherali-Adams.

A preliminary version of this work appeared at the 14th *Innovations in Theoretical Computer Science* (ITCS).

1 Introduction

Recently, connections between proof complexity and monotone circuit complexity have been central to resolving long-standing open problems in a variety of areas. These connections take the form of:

- *Interpolation Theorems*, which translate small proofs into efficient computations in an associated model of monotone circuit [6, 17, 18, 21, 32, 37–39, 45, 47, 49].
- *Query-to-Communication Lifting Theorems*, which translate efficient monotone computations into small proofs in an associated proof system [10, 15, 16, 23, 29–31, 36, 40, 43, 44, 48, 50].

Recently, the landscape of *total functional NP* (TFNP) has emerged as an organizing principle for connections between proof systems and models of monotone circuitry [13, 28] in the following sense. For many of the proof systems which admit an interpolation theorem or lifting theorem there is a subclass of TFNP which *characterizes* it: the set of theorems efficiently provable in this proof system is equivalent to the total functions contained within this class, in the *black-box* setting. This has resulted in proof complexity becoming a major tool for proving black-box/oracle separations in TFNP. As well, the framework of TFNP has provided a number of unique results for proof complexity, including complete tautologies for any proof system admitting a TFNP-characterization, and intersection theorems showing that some proof systems are the intersection several others.

An analogous phenomenon exists for circuit complexity. For many models of monotone computation, the set of functions which can be computed efficiently is equivalent to the set of total search problems which can be reduced to a complete problem for a corresponding TFNP subclass, via *communication*-efficient reductions.

When a proof system and circuit model are characterized by the same TFNP subclass, then we immediately get an interpolation theorem! As well, 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 that understanding when TFNP classes admit such characterizations is 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 subclass. 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 subclass — simply the search problem associated with its reflection principle. This gives a recipe for constructing a TFNP subclass which characterizes a given proof system, simply use the search problem for its reflection principle as the complete problem! Conversely, we show that every TFNP class which has a complete 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 subclass with a complete problem, we give a proof system which it characterizes. As an example, we provide a TFNP characterization of the Polynomial Calculus, as well as several variants, answering a question from [27], and show that it can prove its own soundness. For circuit complexity, we show that every well-behaved model of monotone circuitry which admits a *universal family* of functions is characterized by a TFNP class. Conversely, every TFNP class with a complete problem gives rise to a monotone circuit model with a universal problem.

[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.

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.

- *False Clause.* SEARCH_F for an unsatisfiable CNF formula $F = C_1 \wedge \cdots \wedge C_m$: given an assignment $x \in \{0,1\}^n$ output the index $i \in [m]$ of a clause such that $C_i(x) = 0$.
- *Monotone Karchmer-Wigderson.* MKW_f for a monotone boolean function f: given $x, y \in \{0,1\}^n$ such that $f(x) = 1$ and $f(y) = 0$ output $i \in [n]$ such that $x_i > y_i$.

The theory of TFNP considers the total search problems for which solutions can be efficiently verified, such as SEARCH and MKW. There is believed to be no complete problem for TFNP [46], and therefore much of the work on this subject has focused on identifying subclasses 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, game theory, circuit complexity, and proof complexity. These classes are typically defined as everything that can be efficiently reduced to a certain existence principle on an exponential-size universe. For example, PPA is the class of search problems that can be reduced to finding an unmatched vertex in a perfect matching on a graph with an odd (exponential) number of vertices. These exponential size objects are given in a *white-box* fashion: they are represented succinctly 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 subclasses relate. However, a separation between any pair of subclasses would imply $\mathsf{P} \neq \mathsf{NP}$. 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, we are now limited to accessing the defining circuit via an oracle. We may query the oracle for each bit of the input, but we may no longer observe its representing circuit (nor are we guaranteed that a representing circuit exists for this instance).

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 SEARCH_F for some unsatisfiable CNF formula F. Using this connection, Göös et al. [28] observed that many prominent TFNP^{dt} subclasses are *characterized* by associated proof systems in the sense that the unsatisfiable formulas that are efficiently provable in that proof system are exactly the problems SEARCH_F that are reducible to the the complete problem for that TFNP^{dt} subclass. This has led to the characterization of many well-studied TFNP^{dt} subclasses:

- $\mathsf{FP}^{dt} = \mathsf{TreeRes}$ [42].
- $\mathsf{PLS}^{dt} = \mathsf{Res}$ [9].
- $\mathsf{PPA}^{dt} = \mathbb{F}_2\text{-NS}$ [28].
- $\mathsf{PPA}_q^{dt} = \mathbb{F}_q\text{-NS}$ for any prime q [34]
- $\mathsf{PPAD}^{dt} = \mathsf{unary\text{-}NS}$ [27].
- $\mathsf{PPADS}^{dt} = \mathsf{unary\text{-}SA}$ [27].
- $\mathsf{EOPL}^{dt} = \mathsf{RevResT}$ [27].
- $\mathsf{SOPL}^{dt} = \mathsf{RevRes}$ [27].
- $\mathsf{CPLS}^{dt} = \mathsf{Res(polylog)}$ [12]

Via these characterizations, separations between these proof systems translate into separations between their corresponding TFNP^{dt} subclasses. This has resulted in a complete picture of how the prominent TFNP^{dt} subclasses relate [2, 7, 19, 27, 28].

This relationship has led to a number of striking results for proof complexity as well. These include:

– *Complete Problems:* Any proof system $\mathcal{P}$ which is characterized by a TFNP^{dt} subclass has a complete formula F, in the following sense: if a proof system can efficiently prove F (and can simulate decision-tree reductions) then that proof system can efficiently simulate $\mathcal{P}$. This complete tautology F states that a complete problem for this subclass is total. Previously, complete problems were known only for strong proof systems which could prove their own reflection principle.

– *Intersection Theorems:* Proof systems which can efficiently prove a formula iff that formula has short proofs in several other proof systems [27, 33, 41].

– *Coefficient Separations:* Separations between the complexity of certain *algebraic* proof system when their coefficients are represented in unary versus binary [27].

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

Communication TFNP and Monotone Circuit Complexity. Karchmer and Wigderson [35] showed that the monotone formula complexity of any monotone function f is equal to the communication complexity of MKW_f. Building on this, Razborov [49] 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: analogously 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 [22, 28].

Following these results, a number of TFNP^{cc} sub-classes have been characterized by models of monotone circuits [18, 28]. 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 and monotone circuit models are characterized by dt and cc TFNP subclasses. Göös et al. [28] observed that in all known examples of TFNP sub-class 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 SEARCH_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 them — this follows immediately by the observation that communication protocols can simulate decision trees! Thus, the landscape of TFNP, together with characterizations of TFNP subclasses 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 an exact characterization of when a proof system admits a characterization by a TFNP^{dt} class. We show that this occurs for any proof system P which meets the following two criteria:

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

ii) *Closure under decision-tree reductions.* Whenever there is a small P-proof of a formula H, and SEARCH_F efficiently reduces to SEARCH_H, then there is also a small P-proof of F.

Conversely, we show that every *syntactic* TFNP^{dt} subclass (meaning that it has a complete problem) has a proof system which characterizes it. Furthermore, this proof system satisfies conditions (i) and (ii).

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

- *For any syntactic* TFNP^{dt} *subclass* $\mathcal{C}$ *there is a proof system* P *satisfying (i) and (ii), such that* $\mathcal{C}$ *characterizes* P*:* P *has short proofs of* F *iff* SEARCH_F *is efficiently reducible to a* $\mathcal{C}$*-complete problem.*
- *For any proof system* P *which satisfies (i) and (ii) there is a* TFNP^{dt} *subclass which characterizes* P*.*

It follows that the false-clause search problem for the reflection principle provides a somewhat a somewhat systematic way of generating a TFNP^{dt} problem which characterizes that proof system. As an example, we introduce a new TFNP subclass, IND-PPA, which characterizes the $\mathbb{F}_2$-Polynomial Calculus proof system. This class contains problems which can be solved by *inductive* parity arguments. Furthermore, we show that the $\mathbb{F}_2$-Polynomial Calculus can prove its own soundness.

Theorem 2 (Informal)**.** $\mathsf{IND\text{-}PPA}^{dt}$ *characterizes* $\mathbb{F}_2$-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 method we use to discover IND-PPA can also be used to discover TFNP^{dt} subclasses which characterize the dynamic versions of all static proof systems for which we currently have TFNP characterizations. In subsection 3.4, we provide TFNP^{dt} subclasses for the $\mathbb{F}_q$*-Polynomial Calculus*, the *unary Polynomial Calculus*, and *unary dag-like Sherali-Adams*. Finally, we observe that these characterizations hold the PCR systems as well. Indeed, from the perspective of TFNP, there is no difference between the standard and dual-variable encodings of CNF formulas (Observation 14).

Our second main result is an exact characterization of the conditions under which models of monotone circuitry admit characterization by a TFNP^{cc} subclass. 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} subclass is characterized by a mpc iff the mpc meets the following criteria:

i) *Closure under low-depth reductions:* 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* $\mathcal{C} \subseteq \mathsf{TFNP}^{cc}$ *such that* $\mathcal{C}^{cc}$ *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 27) 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 [25]. On the other hand, Garlik [24] 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 reflection principle about itself. There are two main differences between our encodings of the reflection principle and previous ones which have appeared in the literature. The first is that our reflection principles are parameterized by a *complexity* parameter c (see Section 2) rather than the typical size parameter. The complexity parameter measures both the logarithm of the size and width/degree of the proof simultaneously. The second is that the reflection principle must be *efficiently verifiable*, meaning that an error in the purported proof in the can always be verified by examining in a small number of bits. Thus for example, our results show that $\mathsf{polylog}(n)$-width Resolution can prove its own reflection principle, which we show explicitly in Appendix A. As well, this means that the bound of Garlik [24] 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 $i \in \mathcal{O}_n$ such that $(x,i) \in R_n$. A total search problem is in TFNP^{dt} if its solutions are *verifiable*: for each n sufficiently large and every $i \in \mathcal{O}_n$ there is a $\mathsf{polylog}(n)$-depth decision tree V_i such that

$$V_i(x) = 1 \iff (x,i) \in R_n.$$

Note that if $R \in \mathsf{TFNP}^{dt}$, then $\log|\mathcal{O}_n| \leq \mathsf{polylog}(n)$ as we must be able efficiently output a solution. Henceforth, for readability, we will suppress the subscript n.

A *decision tree reduction* from $Q \in \{0,1\}^n \times \mathcal{O}_Q$ to $R \subseteq \{0,1\}^s \times \mathcal{O}_R$ is a set of decision trees $T_i : \{0,1\}^n \to \{0,1\}$ for $i \in [s]$ and $T^o_j : \{0,1\}^n \to \mathcal{O}_Q$ for $j \in \mathcal{O}$ such that for any $x \in \{0,1\}^n$,

$$((T_1(x), \ldots, T_s(x), j) \in R \implies (x, T^o_j(x)) \in Q.$$

That is, the T_i's map inputs to from Q to R, and the T^o_j'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 s. The *complexity* of the reduction is $\log s + d$ and the complexity of reducing Q to R is the minimum complexity of any decision tree reduction from Q to R. We say that Q reduces to R, denoted $Q \leq_{dt} R$, if there is a $\mathsf{polylog}(n)$-complexity reduction from Q to R for all sufficiently large n. We say that $Q =_{dt} R$ if $Q \leq_{dt} R$ and $R \leq_{dt} Q$.

The intimate connection between black-box TFNP and proof complexity is summarized by the following claim from [27, 28]. Recall that the *width* of a CNF formula is the maximum number of literals in any clause of the formula.

Claim 4. *If $R \in \mathsf{TFNP}^{dt}$ then there is an unsatisfiable $\mathsf{polylog}(n)$-width CNF F such that $R =_{dt} \textsc{Search}_F$.*

Proof. Let $R \subseteq \{0,1\}^n \times \mathcal{O}$ be a search problem in TFNP^{dt}. We build a CNF formula F asserting that R is *not* total.

As $R \in \mathsf{TFNP}^{dt}$, there are decision trees V_i for each $i \in \mathcal{O}$ such that $V_i(x) = 1$ iff $(x,i) \in R$. For each V_i, we build a CNF formula saying that i is not a solution to R. Say that a root-to-leaf path in V_i is a 1-*path* if its leaf is labelled 1. If we follow a 1-path of V_i on input x then this path verifies that $(x,i) \in R$. Thinking of each path in this decision tree as a conjunction of the literals which appear along it, we define the CNF

$$\mathcal{C}_i := \bigwedge_{\text{1-path } p \in V_i} \neg p$$

which states that no input $x \in \{0,1\}^n$ outputs solution i. We state that R is not total with the CNF formula $F := \bigwedge_{i \in \mathcal{O}} \mathcal{C}_i$. As $R \in \mathsf{TFNP}^{dt}$ the depth of each V_i, and hence the width of F is at most $\mathsf{polylog}(n)$.

To reduce R to $\textsc{Search}_F$, let $T_i(x) := x_i$ and $T_j^o(x) := i$ if clause C_j of F is one of the clauses of $\mathcal{C}_i$. Conversely, we can reduce $\textsc{Search}_F$ to R by letting $T_i(x) := x_i$ and setting T_i^o to be the decision tree which queries V_i and outputs the index of the clause $\neg p$ for the root-to-leaf path p followed in V_i. □

The upshot is that black-box TFNP is *exactly* the study of the false clause search problem! Therefore, without loss of generality, we focus on the search problems $\textsc{Search}_F$ associated with $R \in \mathsf{TFNP}^{dt}$ instead of R itself. Using this connection, Göös et al. [28] observed that many important proof systems are characterized by an associated TFNP^{dt} class in the sense that the CNF formulas F that are efficiently provable in that proof system are exactly the search problems $\textsc{Search}_F$ that are efficiently reducible to this TFNP^{dt} class.

TFNP Characterizations. 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

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

where width denotes an associated *width* measure of the proof system. For Nullstellensatz and Sherali-Adams, this width measure is the maximum degree of any polynomial in their proofs, while for Resolution, width is the maximum number of variables in any clause in a proof. 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} subclass $\mathcal{C}$ *characterizes* a proof system P if

$$\mathcal{C}^{dt} = \Big\{\textsc{Search}_F : P(F) = \mathsf{polylog}(n)\Big\}.$$

That is, the false-clause search problems (and hence by Claim 4, the problems in $\mathcal{C}$) are exactly the $\mathsf{polylog}(n)$-width CNF formulas provable in P. This is reflexive and so we also say that P characterizes $\mathcal{C}$.

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) its *reflection principle*.

2.1 What is a Reflection Principle?

The first 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,

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

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, 26]) can be efficiently verified.

For a proof system P there are many ways to encode its proofs, with the choice of the encoding potentially affecting the complexity of proving the associated reflection principle. Rather than worrying about the particular encoding, we will instead define one reflection principle for each efficiently verifiable way of encoding P-proofs, which we 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 two sets of variables 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. Note that we can always specify clauses of smaller width by repeating the same literal, and specify a formula with few clauses by repeating the same clause multiple times.

- An assignment to the variables Π specifies a (purported) P-proof of H, such that any purported 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 the number of variables is exponential in $\Theta(c)$ by the third condition, the second condition ensures that $V_{n,m,c}$ has width $\mathsf{poly}(c)$ and can be verified by looking at $\mathsf{poly}(c)$-many variables. The third condition can be relaxed, and larger numbers of variables can be tolerated at the cost of worse bounds in Theorem 10. We give a concrete example of a verification procedure for the Polynomial Calculus proof system in Section 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 $\log 2n_H$-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 formula stating that α is a satisfying assignment for H:

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

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

Sat can be encoded as a CNF formula of width $O(c \log n_H)$ and size $m_H \exp(O(c \log n_H))$. To see this, construct the following decision tree T_i for each $i \in [m_H]$: for each $j \in [c]$, query the $2n_H$-ary variables $C_{i,j}$ to determine the literal ℓ_k or for $C_{i,j}$ and then query the value of $\alpha_k \in \{0,1\}$. Label this leaf with 1 if α_k agrees with the literal specified by $C_{i,j}$ and 0 otherwise. Using this, we build the width-$O(c \log n_H)$ formula

$$\mathsf{Sat} := \bigwedge_{i \in [m_H]} \ \bigwedge_{1\text{-path } p \in T_i} \neg p.$$

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.

2.2 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 substitutions of decision trees for variables, as TFNP^{dt} subclasses are closed under such 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 $\textsc{Search}_H$ reduces to $\textsc{Search}_F$ by depth-d decision trees, then there is a P-proof of H of complexity $\mathsf{poly}(cd)$.

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. For example, if $\textsc{Search}_H$ reduces to $\textsc{Search}_F$ 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 (CNF formula corresponding to the negation of the accepting paths of) corresponding decision tree from the reduction. If

these are depth d decision trees, and the Resolution proof has width w and size s, then after substitution we have a Resolution proof of width at most dw and size at most $s\exp(dw)$. Hence, the proof width and size remain poly-logarithmic and quasi-polynomial, respectively, if $d, w, \log s = \mathsf{polylog}(n)$.

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

Theorem 5. *The following hold:*

i) For any syntactic TFNP^{dt} *subclass* $\mathcal{C}$ *there is a proof system* P *such that* $\mathcal{C}$ *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} *subclass* $\mathcal{C}$ *which characterizes* P*.*

In fact, we prove a tighter characterization over the following two subsections, from which Theorem 5 will follow. Part (i) is proven in Theorem 7, with the "furthermore" part proven in Theorem 9, and part (ii) is proven in Theorem 10.

2.3 A Proof System for any TFNP Problem

We show how to construct, from any TFNP^{dt} subclass with a complete problem R, a proof system which characterizes it. The key insight is that one can view decision tree reductions to R as proofs — indeed, the correctness of a reduction can be verified efficiently! This efficient verifiability can then combined with the characterization of problems in TFNP^{dt} by instances of the false-clause search problem in order to obtain a proof system for UNSAT. More concretely, let $F = \{F_n\}_{n\in\mathbb{N}}$ be the unsatisfiable CNF formula such that $\textsc{Search}_F$ is equivalent to R. A P_F-proof Π will consist of an unsatisfiable CNF formula H consists of a decision-tree reduction from $\textsc{Search}_H$ to $\textsc{Search}_F$. In order to define this formally, we will use the following notion of a reduction between CNF formulas.

Suppose that 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 $i \in \mathsf{Vars}(C)$ and rewriting the result as a CNF formula. Formally, $C(T)$ is formed by creating a decision tree T^C that sequentially runs the trees T_i for each variable x_i used in C. Say that a path p in T^C is *violating* if corresponds to queried trees T_i outputting an assignment which falsifies C; that is, p satisfies the constraint

$$\bigwedge_{i\in\mathsf{Vars}^+(C)} [\![T_i(p) = 0]\!] \wedge \bigwedge_{i\in\mathsf{Vars}^-(C)} [\![T_i(p) = 1]\!],$$

where $\mathsf{Vars}^+(C)$, $\mathsf{Vars}^-(C)$ are the indices variables which occur positively and negatively in C. That is, $C(T(p)) = 0$ for every violating path. Label the leaves of the violating paths $p \in T^C$ by 0, and label the remaining leaves by 1. Then, $C(T)$ is the CNF formula

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

where a path p is identified with the conjunction of the literals set true along the path, and $\neg p$ is its negation. $C(T)$ asserts that C is not falsified. Another way to see this is to say that a root-to-leaf path in T_i is a

$t \in \{0,1\}$-path if it ends at a leaf labelled t. Then

$$C(T) := \neg\Big(\bigwedge_{i \in \mathsf{Vars}^+(C)} \Big(\bigvee_{\text{0-path } p \in T_i} p \Big) \wedge \bigwedge_{i \in \mathsf{Vars}^-(C)} \Big(\bigvee_{\text{1-path } p \in T_i} p \Big) \Big),$$

stating that we do not follow path in the trees T_i for $i \in \mathsf{Vars}(C)$ which falsifies C.

Reductions Between CNF Formulas. 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 is a depth-d decision tree reduction $T = \{T_i\}_{i \in n}$ where $T_i : \{0,1\}^{n_H} \to \{0,1\}$, and $T^o = \{T_i^o\}_{i \in [m]}$ with $T_i^o : \{0,1\}^{n_H} \to [m_H]$ from $\textsc{Search}_H$ to $\textsc{Search}_F$. We can interpret this syntactically as follows. Define the *reduced formula* F_H as the CNF formula

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

where $p \in T_i^o$ denotes that p ranges over all root-to-leaf 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, since (T, T^o) is a valid reduction, each clause of $C_i(T) \vee \neg p$ must either be tautological (contains a literal and its negation) or be a weakening of the clause of H indexed by the label of the leaf reached by taking the path p in T_i^o.

Letting the width and size of a CNF formula be the maximum number of literals in any clause, and the number of clauses in the formula respectively, the following hold:

- $\mathsf{width}(F_H) \leq d \cdot \mathsf{width}(F)$,
- $\mathsf{size}(F_H) \leq \mathsf{size}(F) \cdot 2^{d \cdot \mathsf{width}(F)}$.

In particular, if F is a quasi-polynomial size, $\mathsf{polylog}(n)$-width CNF formula and $d = \mathsf{polylog}(n)$, then F_H has quasi-polynomial size and $\mathsf{polylog}(n)$-width.

Observe that a depth-d decision tree reduction of $\textsc{Search}_H$ to $\textsc{Search}_F$ introduces a new false clause search problem $\textsc{Search}_{F_H}$ which is a refinement of $\textsc{Search}_H$. Clearly, if F is unsatisfiable, then so is F_H and consequently also H is unsatisfiable.

Canonical Proof System. Let $\textsc{Search}_F \in \mathsf{TFNP}^{dt}$. The canonical proof system P_F for $\textsc{Search}_F$ 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 $T^o = \{T_i^o\}_{i \in [m]}$ of the reduction, along with the reduced formula F_H.

The *size* of Π is the number of bits needed to write down the proof $\Pi = (T, T^o, F_H)$, which is polynomially related to $s(F_H)$, and the *width* is $w(F_H)$, which is polynomially related to the maximum depth among any of the decision trees in T, T^o. The *complexity* of proving an unsatisfiable CNF formula H is then the minimum over all P_F-proofs of H,

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

This proof system is sound as any substitution of an unsatisfiable CNF formula is also unsatisfiable, and it is complete as $\textsc{Search}_H$ for any unsatisfiable CNF formula H can be solved by a decision tree of depth n. The following claim shows that P_F is polynomial-time verifiable.

Claim 6. *There is a polynomial-time Turing Machine M such that $M(H, \Pi) = 1$ if and only if Π is a P_F-proof of H.*

Proof. Let (T, T^o, F_H) be a P_F-proof of an unsatisfiable CNF formula H. Our verifier M checks the following conditions:

- F_H is the CNF formula obtained by the reduction (T, T^o) applied to F.
- Each clause of F_H is either a weakening of a clause in H or tautological, meaning that it contains a variable and its negation.

As the number of clauses of F_H is bounded above by the size of the proof, this can be done in polynomial-time in the size of the proof. □

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

The next theorem states that P_F has a short proof of H iff $\textsc{Search}_H$ efficiently reduces to $\textsc{Search}_F$. This is almost immediate from the definitions.

Theorem 7. *If $\textsc{Search}_F, \textsc{Search}_H \in \mathsf{TFNP}^{dt}$ then $\textsc{Search}_H \leq_{dt} \textsc{Search}_F$ iff $P_F(H) \leq \mathsf{polylog}(n)$.*

This theorem establishes part of (i) in Theorem 7; the "furthermore" part is proven in the following subsection. This theorem follows from the next lemma by setting $c \leq \mathsf{polylog}(n)$.

Lemma 8. *Let $\textsc{Search}_F \in \mathsf{TFNP}^{dt}$. For any unsatisfiable CNF formula H, there is a complexity c reduction from $\textsc{Search}_H$ to $\textsc{Search}_F$ iff $P_F(H) \leq c \cdot \mathsf{polylog}(n)$.*

Proof. Suppose that there is a complexity c reduction from $\textsc{Search}_H$ to $\textsc{Search}_F$. Letting $T = \{T_i\}_{i \in [n]}$ and $T^o = \{T^o_j\}_{j \in [m]}$ be the decision trees in this reduction, construct the CNF formula F_H as above. Note that since $\textsc{Search}_F \in \mathsf{TFNP}^{dt}$, F contains at most quasipolynomially-many clauses and has width at most $\mathsf{polylog}(n)$. Hence $\mathsf{size}(F_H) \leq \exp(c \cdot \mathsf{polylog}(n))$ and $\mathsf{width}(F_H) = O(\mathsf{polylog}(n))$, and $P_F(H) = c \cdot \mathsf{polylog}(n)$.

It remains to verify that (F_H, T, T^o) is a P_F-proof of H. Fix any clause L of F_H. L belongs to some CNF formula $C_i(T) \vee \neg p$ for some clause C_i of F and root-to-leaf p of T^o_i. If L is tautological, then we are done. Otherwise we will argue that it is a weakening of a clause of H. Fix any assignment which falsifies L. As L is a clause of $C_i(T) \vee \neg p$ it has $\neg p$ as a sub-clause, and so $p(x)$ is satisfied. Thus, by the correctness of the reduction from $\textsc{Search}_H$ to $\textsc{Search}_F$, the $T^o_i(x)^{th}$ clause D of H must be falsified by $T(x)$ (as $T^o_i(x)$ is a solution to $\textsc{Search}_H$). Hence, we have that $\neg L \implies \neg D$ and so D is a weakening of L.

For the converse direction, suppose that there is a proof (T, T^o, F_H) of H in P_F. We claim that (T, T^o) constitutes a reduction from $\textsc{Search}_H$ to $\textsc{Search}_F$. To see this, fix any input x to $\textsc{Search}_H$. As F is unsatisfiable, $T(x)$ falsifies some clause C_i of F, and x follows some root-to-leaf path p in T^o_i. Hence, x falsifies some clause L in the sub-formula $C_i(T(x)) \vee \neg p(x)$ of F_H. As F_H is a proof of H, and x follows path p in $T^o_i(x)$, L must be a weakening of clause $T^o_i(p) = T^o_i(x)$ of H. Thus, the $T^o_i(x)^{th}$ clause of H is falsifies on input x and we conclude that $(x, T^o_i(x)) \in \textsc{Search}_H$.

Finally, let c be the complexity of this proof. Then, each decision tree in (T, T^o) has depth at most c and there are at most 2^c of them. Hence, this is a complexity c reduction. □

2.4 Canonical Proof Systems Prove their own Soundness

In this section we define a natural formulation of the reflection principle for any canonical proof system P by defining a verification procedure for its proofs. Then, we show that P can prove this encoding of its reflection principle.

To construct this verification procedure we must define a CNF formula which generically encodes any complexity-c P-proof in the sense that all complexity-c proofs can be recovered from variable instantiations of this formula. As proofs in a canonical proof system are decision tree reductions, the following notion of a *generic decision tree* will be useful. One should think of a generic decision tree as a template for a decision tree, such that any decision tree of depth d can be recovered from it by correctly instantiating its variables.

Generic Decision Tree. Let $\mathcal{O}$ be a set and $z_1, \ldots, z_n$ be Boolean variables. A generic n-ary decision tree $\mathcal{T}$ of depth d specifying a decision trees over the variables $\vec{z}$ with output in $\mathcal{O}$ is the following complete binary tree.

- *Non-leaves.* For each non-leaf vertex v of $\mathcal{T}$, the left outgoing edge is labelled 0 and the right outgoing edge is labelled 1. The vertex v is labelled by an n-ary variable x_v.
- *Leaves.* Each leaf ℓ is labelled with an $|\mathcal{O}|$-ary variable x_ℓ.

An assignment to the variables of a generic n-ary decision tree specifies a decision tree over the variables $z_1, \ldots, z_n$ as follows. For each vertex v, the assignment to x_v specifies the z-variable which should be queried at vertex v. For example, if $x_v = i$ then v is labelled by z_i. Similarly, for each leaf ℓ, the assignment to x_ℓ specifies the element $o \in \mathcal{O}$ which is the outcome of the decision tree at this leaf. For any root-to-leaf path $p \in \mathcal{T}$, let $\mathsf{Lits}(p)$ be z-literals named by this root-to-leaf path.

A generic n-ary decision tree of depth d can be used to specify any decision tree T of depth at most d by naming the index of the variable queried at each vertex v using the x_v variables. If T not a complete binary tree then in order to shorten a root-to-leaf path, we can name the same variable repeatedly along this path. For example, if p is a root-to-leaf path in T of depth 1, involving only a single variable z_i, then for for every v in the sub-tree rooted at the leaf of p in $\mathcal{T}$, we set $x_v = i$.

A generic n-ary decision tree may be converted into a *generic decision tree* $\mathcal{T}$ over binary variables by the following recursive process (see Figure 1): beginning at the root of $\mathcal{T}$. Let v be the current non-leaf vertex. Replace each x_v by $(\log n + 1)$-many variables $x_v = [x_{v_1}, \ldots, x_{v_{\log n}}, s_v]$, where s_v is a sign bit indicating the polarity of variable x_v. Replace v by a depth-$\log n + 1$ complete binary tree T_v which sequentially queries $x_{v_1}, \ldots, x_{v_{\log n}}$ to obtain the value variable $x_v \in [n]$ and then the sign bit s_v to determine whether the variable indexed by x_v should be positive or negative. If v had left and right children u and w in $\mathcal{T}$, then at each leaf of T_v where $s_v = 0$ we recurse on the child u and at leaves where $s_v = 1$ we recurse on v. At each leaf ℓ we replace x_ℓ by $\log |\mathcal{O}|$-many variables, replace ℓ with a binary tree querying these variables, and label each leaf with the outcome in $\mathcal{O}$.

Verification Procedure for P_F. Fix some $\textsc{Search}_F \in \mathsf{TFNP}^{dt}$. We define a verification procedure $V = V_{n_H, m_H, (d, n_F)}$ for P_F, which encodes a complexity $c = \mathsf{poly}(d, \log n_F)$ P_F-proof Π of a CNF formula H on n_H variables and m_H clauses as follows. The proof Π is specified by n_F-many depth-d generic decision trees $\mathcal{T}_1, \ldots, \mathcal{T}_{n_F}$ which specify decision trees over the variables of H with output in $\{0, 1\}$ and m_F depth-d generic decision trees $\mathcal{T}^o_1, \ldots, \mathcal{T}^o_{m_F}$ which specify decision trees over the variables of H with output in $[m_H]$. The variables of V will be

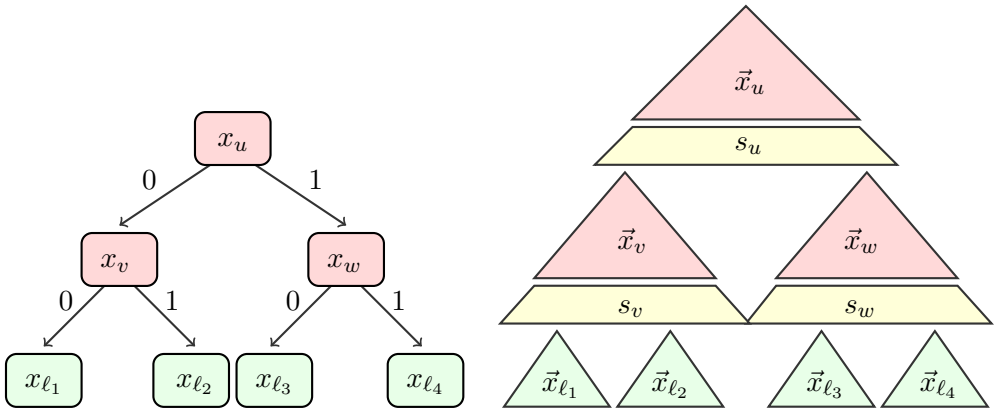


Figure 1: A generic n-ary decision tree (left) and its corresponding generic decision tree (right).

– $\{C_{i,j} : i \in [m_H], j \in [c]\}$, where $C_{i,j} \in [2n_H]$ (encoded by binary variables) names the j^{th} literal in the i^{th} clause of H.

– x_v for each vertex v of $\mathcal{T}_i, \mathcal{T}_j^o$ where x_v names the variable of H being queried at vertex v or if v is a leaf, is either a boolean variable if v is from $\mathcal{T}_i$ or names a clauses of H if it belongs to $\mathcal{T}_j^o$.

The constraints of the CNF formula V will enforce that each clause of the reduced CNF formula F_H is a weakening of a clause of H. In particular, the constraints will assert that for each clause C_i of $F = F_{n_F}$ and root-to-leaf path $p \in \mathcal{T}_i^o$, the clauses of $C_i(\mathcal{T}) \vee \neg p$ are a weakening of the $\mathcal{T}_i^o(p)^{th}$ clause of H.

The CNF formula $C_i(\mathcal{T})$ is formed as before, except that we now query the variables of the generic decision tree. In particular, we create a stacked decision tree $\mathcal{T}^{C_i}$ formed by sequentially running the trees $\mathcal{T}_i$ for each $i \in \mathsf{Vars}(C_i)$, querying the variables $x_v^{(j)}$, $s_v^{(j)}$, and $x_\ell^{(j)}$ of $\mathcal{T}_i$. For any root-to-leaf path p in $\mathcal{T}^{C_i}$ let $\{x_\ell^{(j)}\}_{j \in [\mathsf{Vars}(C_i)]}$ be the leaf variables queried along p — these determine the assignment to clause C_i. Say that a path p is *violating* if p satisfies

$$\bigwedge_{j \in \mathsf{Vars}^+(C_i)} [\![x_\ell^{(j)} = 0]\!] \wedge \bigwedge_{j \in \mathsf{Vars}^-(C_i)} [\![x_\ell^{(j)} = 1]\!],$$

where $\mathsf{Vars}^+(C_i), \mathsf{Vars}^-(C_i)$ are the variables which occur positively and negatively in C_i. That is, p is violating if it corresponds to an assignment which falsifies the clause C_i. Then the CNF formula

$$C_i(\mathcal{T}) := \bigwedge_{\text{violating } \hat{p} \in \mathcal{T}^{C_i}} \neg \hat{p}$$

The constraints of V enforce that each clause $\neg \hat{p} \vee \neg p$ of

$$\bigwedge_{p \in \mathcal{T}_i^o} C_i(\mathcal{T}) \vee \neg p = \bigwedge_{\text{violating } \hat{p} \in \mathcal{T}^{C_i}} \bigwedge_{p \in \mathcal{T}_i^o} \left(\neg \hat{p} \vee \neg p\right)$$

is a weakening of the $\mathcal{T}_i^o(p)^{th}$-clause of H. Let $\mathsf{Lits}_H(p)$ be the literals of H named along the a path p of one of these generic decision trees. As well, let $\mathcal{H}_k$ be the decision tree which sequentially queries the variables $C_{k,j} \in [2n_H]$ for each $j \in [c]$. Each root-to-leaf path p^* in this decision tree determines the

variables which appear in the k^{th} clause of H. Say that p^* is *bad* for a pair of paths $(\hat{p}, p) \in \mathcal{T}_i^o \times \mathcal{T}^{C_i}$ if $\mathsf{Lits}_H(p^*) \not\subseteq \mathsf{Lits}_H(\hat{p} \wedge p)$, meaning that $C_i(\mathcal{T}) \vee \neg p$ are not a weakening of the k^{th} clause of H.

The constraint corresponding to $C_i(\mathcal{T}) \vee \neg \hat{p}$ are

$$W_i := \bigwedge_{\text{violating } \hat{p} \in \mathcal{T}^{C_i}} \bigwedge_{p \in \mathcal{T}_i^o} \bigwedge_{p^* \in \mathcal{H}_{T_i^o(p)} \text{bad for } (p,\hat{p})} (\neg \hat{p} \vee \neg p \vee \neg p^*)$$

stating that if we follow path $\hat{p}$ of $\mathcal{T}^{C_i}$ and path p of $\mathcal{T}_i^o$ then the clause $C_i(\mathcal{T}) \vee \neg \hat{p}$ of F_H is a weakening of the $\mathcal{T}_i^o(p)^{th}$ clause of H. The CNF formula

$$V(H, \Pi) := \bigwedge_{i \in m_F} W_i,$$

is satisfied only when Π is a valid P_F-proof of H. As each generic decision tree has depth at most d, F has width $O(\mathsf{polylog}(n_F)$, and H has width at most c, this is a CNF formula of width at most $\mathsf{polylog}(n_F)(d \log(n_H) + \log m_H) + c \log n_H$.

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

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

where Sat is defined as in the definition of the reflection principle and $\mathsf{Proof} := V^P_{n_H, m_H, (d, n_F)}$. In total, this is a CNF formula of width $d \log n_F + \log m_H + c \log n_H$ over $n = m_F 2^{d+1} + n_F 2^d \log n_H + c m_H \log 2 n_H$ 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.[2] Thus, $\textsc{Search}_{\mathsf{Ref}_F} \in \mathsf{TFNP}^{dt}$.

The following theorem establishes the "furthermore" part of (i) of Theorem 7.

Theorem 9. *For any* $\textsc{Search}_F \in \mathsf{TFNP}^{dt}$, $P_F(\mathsf{Ref}_F) = O(\mathsf{polylog}(n))$.

Proof. Fix an instance of $\textsc{Search}_{\mathsf{Ref}_F}$. By Theorem 7, it suffices to show that $\textsc{Search}_{\mathsf{Ref}_F}$ is reducible to an instance of $\textsc{Search}_F$. Let the parameters of Ref_F be $(n_H, m_H, (d, n_F))$, where $c = d + \log n_F$, and let $\mathcal{T}_i, \mathcal{T}_j^o$ be the generic decision trees of Ref_F.

We will define the decision trees $T_1, \ldots, T_{n_F}, T_1^o, \ldots, T_{m_F}^o$ specifying a reduction from $\textsc{Search}_{\mathsf{Ref}_F}$ to an instance of $\textsc{Search}_F$ on n_F variables. To do so, we will say that a decision tree T *evaluates* one of these generic decision tree $\mathcal{T}$ by the following recursive procedure, beginning at the root of $\mathcal{T}$: Let v be the current vertex. If v is not a leaf then query x_v and let $j \in [n_H]$ be the outcome. Then query α_j for the value of the j^{th} variable of H. If $\alpha_j = 0$ then proceed to the left child of v in $\mathcal{T}_i$ and to the right child if $\alpha_j = 1$. Otherwise, if v is a leaf query x_v and output its value.

– T_i evaluates $\mathcal{T}_i$ and outputs the value in $\{0, 1\}$ to should assign to the i^{th} variable of F.

[2] Note that in this definition of Ref, and in particular V, we have excluded the formula F_H from the input, and let it be defined implicitly from the decision trees and n_F. We could instead explicitly include F_H as part of the input. However, as F has width c and each decision tree has depth at most c, F_H is a CNF formula of size at most $m_H \cdot 2^{dc}$. Hence, F_H would be encoded generically in Π with that many variables. This would only change the number of variables of Ref by a quasi-polynomial amount, and hence the complexity of verifying and reducing to Ref by a $\mathsf{polylog}(n)$ amount. Thus, these formulas are equivalent up to polylog-reductions.

– T_j^o first sequentially evaluates $\mathcal{T}_i$ to obtain the value of the variable $i \in \mathsf{Vars}(C_j)$, where C_j is the j^{th} clause of F. If this truth assignment satisfies the C_j, then T_j^o outputs something arbitrary. Otherwise, if C_j is falsified, then T_j^o evaluates $\mathcal{T}_j^o$ to obtain the index $k \in [m_H]$ of a clause of H. The assignment discovered thus far must falsify some clause D of the formula F_H. D should be a weakening k^{th} clause of H. To check whether this is the case, T_j^o queries the values of the variables $C_{k,\ell}$ for each $\ell \in [c]$ to determine the variables which belong to k^{th} clause. If D is not a weakening of this clause, then we have found a falsified clause of Ref, which we output. Otherwise, if D is a weakening of the k^{th} clause, then as D is falsified, so is the k^{th} clause of H. Hence, we must have discovered a falsified clause of SAT, which we output.

Each T_i has depth $O(d \log n_H)$ and each T_j^o has depth $O(w(F) \cdot d \log n_H + \log m_H + c \log n_H)$, which are both poly-logarithmic in the number of variables of Ref_F. □

2.5 TFNP Problems for Proof systems which Prove their own Soundness

In this section we identify the following two conditions as being necessary for a proof system to be characterized by a TFNP^{dt} problem:

1. Closed under decision-tree reductions, and
2. Admits short proofs of a reflection principle about itself.

The first condition is required because sub-classes of TFNP^{dt} are closed under decision tree reductions. For the second condition, we will see that *any* verification procedure for this proof system will suffice.

Theorem 10. *Let P be a proof system that is closed under decision tree reductions and that has $\mathsf{polylog}(n)$-complexity proofs of a reflection principle Ref about itself, then P is characterized by $\textsc{Search}_{\mathsf{Ref}}$.*

We begin with a high-level sketch of the proof. If there is a reduction from some $\textsc{Search}_H$ to $\textsc{Search}_{\mathsf{Ref}}$ then, as P is closed under decision tree reductions, there is a proof of roughly the same complexity. Conversely, if we have a P-proof Π of an unsatisfiable formula H then we can construct a reduction from $\textsc{Search}_H$ to $\textsc{Search}_{\mathsf{Ref}}$ of similar complexity by hard-wiring the description of Π and H into Ref, leaving only the variables of H free. Hence solutions to this instance of $\textsc{Search}_{\mathsf{Ref}}$ are falsified clauses of H. The following lemma is a technical statement of Theorem 10.

Lemma 11. *Let P be a proof system that is closed under decision tree reductions, and let V be a verification procedure for P, and denote by $\mathsf{Ref} = \mathsf{Ref}_{P,V}$. Then for any unsatisfiable CNF formula H,*

i) If there is a complexity c reduction from $\textsc{Search}_H$ to $\textsc{Search}_{\mathsf{Ref}}$ then $P(H) = \mathsf{poly}(c \cdot P(\mathsf{Ref}))$.

ii) There is a complexity $O(P(H))$ reduction from $\textsc{Search}_H$ to $\textsc{Search}_{\mathsf{Ref}}$.

Theorem 10 follows immediately from Lemma 11 assuming that $P(\mathsf{Ref}) = \mathsf{polylog}(n)$. Indeed, (i) implies that if there is a $\mathsf{polylog}(n)$-complexity reduction from $\textsc{Search}_H$ to $\textsc{Search}_{\mathsf{Ref}}$ then $P(H) = \mathsf{polylog}(n)$. Conversely, (ii) implies that if $P(H) = \mathsf{polylog}(n)$ then $\textsc{Search}_H \leq_{dt} \textsc{Search}_{\mathsf{Ref}}$.

Proof of Lemma 11. Let H be any unsatisfiable CNF formula and suppose that there is a complexity c reduction from $\textsc{Search}_H$ to $\textsc{Search}_{\mathsf{Ref}}$. Then, as P is closed under decision tree reductions, there is a P-proof of H with complexity $P(H) = \mathsf{poly}(c \cdot P(\mathsf{Ref}))$.

For the other direction, let Π be a complexity $P(H)$ proof of H in P. We construct a reduction $T = (T_1, \ldots, T_n), T^o = (T^o_1, \ldots, T^o_m)$ from SEARCH_H to $\text{SEARCH}_{\mathsf{Ref}}$ as follows. Let n_H, m_H be the number of variables and number of clauses of H respectively. The decision trees T will hardwire the input (H, Π) into the instance of $\mathsf{Ref} = \mathsf{Proof} \wedge \mathsf{SAT}$ with parameters $n_H, m_H, P(H)$, using constant decision trees, and map the input variables $x_1, \ldots, x_{n_H}$ of SEARCH_H to the variables $\alpha_1, \ldots, \alpha_{n_H}$ of Ref. For each clause i of Proof we let T^o_i output the index of an arbitrary clause of H. For each clause ℓ of SAT, corresponding to some clause $i \in [m_H]$ of H, the tree T^o_ℓ outputs i. This is a constant-depth decision tree reduction to an instance of $\text{SEARCH}_{\mathsf{Ref}}$ on $\exp(c)$-many variables. Hence the complexity of this reduction is $O(P(H))$.

It remains to argue that this reduction is correct. Let $\alpha \in \{0,1\}^{n_H}$ be any assignment to SEARCH_H. As Π is a valid P-proof of H, all of the clauses of $\mathsf{Proof}(T(\alpha))$ are satisfied. The formula $\mathsf{SAT}(T(\alpha))$ assets that H is satisfied by α, and hence some clause of SAT must be falsified. By construction, if ℓ is a falsified clause of SAT then it must be part of the constraints encoding that some clause $i \in [n_h]$ of H is satisfied. Hence clause i of H must be falsified, and T^o_ℓ correctly outputs the index i of a falsified clause of H. □

3 Characterizations of the Polynomial Calculus

As an instantiation of the generic connection between TFNP^{dt} and proof complexity, we develop a total search problem which characterizes the Polynomial Calculus. Using this, we show that the Polynomial Calculus can prove its own reflection, establishing Theorem 2. We do this for every finite field, as well as PC over the integers when coefficients are measured in unary. This answers an open question from [27].

The Polynomial Calculus (PC). The Polynomial Calculus proves that a system of polynomials $\{p_i(x)\}_{i\in[m]}$ has no common root over $\{0,1\}^n$. An unsatisfiable CNF formula $F = C_1 \wedge \ldots \wedge C_m$ is encoded as a system of polynomial by mapping each clause C_i to the conjunct $\overline{C}_i$. For example, $(x_1 \vee \neg x_2 \vee x_3)$ is represented as $(1 + x_1)x_2(1 + x_3) = 0$. We will operate exclusively with multilinear arithmetic; that is, x_i^2 and x_i are represented by the same function; that is, we work over the ideal $\langle x_i^2 = x_i \rangle_{i\in[n]}$.

A PC *derivation* of a polynomial q from a set of polynomials $\{p_i(x)\}_{i\in[m]}$ using the following two inference rules:

- *Addition.* From two previously derived polynomials p, q, deduce $p + q$.
- *Multiplication by a Variable.* From a previously derived polynomial p, deduce $x_i p$ for some $i \in [n]$.

The *size* of a proof is the number of monomials (counted 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. A PC *proof* of a system of polynomials $\{p_i(x)\}_{i\in[m]}$ is a derivation of the polynomial 1, certifying that the set of polynomials has no roots over $\{0,1\}^n$. The *complexity* of proving an unsatisfiable CNF formula F in PC is

$$\min\{\mathsf{size}(\Pi) + \log \mathsf{degree}(\Pi) : \Pi \text{is a PC proof of } F\}$$

For any prime q, $\mathbb{F}_q$-PC is obtained from PC by replacing addition by addition over $\mathbb{F}_q$.

We begin by characterizing the $\mathbb{F}_2$-PC. Our characterization will extend the characterization of $\mathbb{F}_2$-Nullstellensatz by PPA [28], the class of total search problems reducible to LEAF.

Leaf. An instance of LEAF_n is given by a function $N : [n] \to [n] \times [n]$. We view N as defining a degree-≤ 2 undirected graph, where an edge (u, v) is present in the graph iff $u \in N(v)$ and $v \in N(u)$. A vertex $v \in [n]$ is a solution if:

- *Root Violations.* If $v = 1$ and $|N(1)| \neq 1$. That is, 1 is not a leaf of the graph.
- *Leaf Violations.* If $v \neq 1$ and $|N(v)| \neq 2$. That is, v is a leaf other than the designated leaf 1.

This problem is total by a parity argument: each vertex is *matched* with its two neighbours. As the 1-vertex only one neighbour, there must be another vertex with only a single neighbour.

The IndPPA class is defined by its complete problem INDLEAF. At a high level, an instance of INDLEAF is formed from an instance of LEAF by partitioning the vertices $[n]$ into groups (for simplicity, in the actual problem each pool actually gets a copy of $[n]$), which we call *pools*, $P_1, \ldots, P_L$, where we require that $P_1 = \{1\}$. These pools are arranged in a rooted dag, with P_1 being the root, and we require that for every P_i with children $P_c, P_{c'}$, there is a matching between the vertices in $P_i \cup P_c \cup P_{c'}$. If the edges of this dag are fixed then this problem remains PPA-complete — essentially it is an instance of LEAF in which we have constrained where each vertex can be matched. However, if we allow the edges of this dag to be defined by variables, then we are able to simulate the ability of PC to dynamically simplify a polynomial after a multiplication step.

Inductive Leaf. An instance of the IndPPA-complete problem INDLEAF for parameters L, N, is given by functions $E : [L] \to [L] \times [L]$ and functions $M^{(v)} : [L] \times [N] \to [L] \times [N]$ for each $v \in [L]$. We will interpret E and $M^{(v)}$ as forming the following structure:

- *The dag E.* We view $[L]$ as the vertices of a rooted dag with the ≤ 2 outgoing edges of vertex v given by $E(v) = (u, w)$. If one (both) of (u, w) are equal to v then we will interpret v as having one (no) child. We will ensure that E forms a dag with root vertex 1 by making v a solution if it is not the case that $u, w \geq v$.
- *The Pools.* We call each vertex $v \in [L]$ a *pool* and associate a subset $P_v \subseteq [N]$ with it; we call these the *nodes* of pool V. To identify this subset, we will say that a node $m \in [N]$ does not appear in pool v if $M^{(v)}(v, m) = (v, m)$ and appears otherwise. That is,

$$P_v := \{m \in [N] : M^{(v)}(v, m) \neq (v, m)\}.$$

- *The Matching.* For each pool $v \in [L]$ with children $E(v) = (u, w)$, we will have a matching between the nodes $P_v \cup P_u \cup P_w$. This matching is given by the function $M^{(v)}$, and we say that $m \in P_i$ and $\gamma \in P_j$ are *matched*, for $i, j \in \{u, v, w\}$, if $M^{(v)}(i, m) = (j, \gamma)$ and $M^{(v)}(j, \gamma) = (i, m)$. The structure of these matchings can be seen in Figure 2.
- *The Root.* The root pool 1 contains only the 1-node, $P_1 = \{1\}$.

There are four types of solutions, which can be seen in Figure 3:

- *Dag Violations.* A pool $v \in [L]$ such that $E(v) = (u, w)$ and either $u < v$ or $w < v$.
- *Root Violations.* Any violation to $P_1 = \{1\}$. That is,

i) $(1,1)$ is a solution if $M^{(1)}(1,1) = (1,1)$, and

ii) $(1,m)$ is a solution if $M^{(1)}(1,m) \neq (1,m)$.

– *Matching Violations.* A node that is incorrectly matched in the matching constraint for a pool v. That is, (v,u,m) is a solution if $u \in E(v) \cup \{v\}$, and $m \in P_u$, and either

i) m is matched to a node in a pool $w \notin E(v) \cup \{v\}$: $M^{(v)}(u,m) = (w,m')$ for some m', or

ii) The node which m is matched to is not matched to m: $M^{(v)}(M^{(v)}(u,m)) \neq (u,m)$

– *Consistency Violations.* If $u \in E(v)$ then node m of pool u occurs in the matching for v iff it occurs in the matching for u. That is, (v,u,m) is a solution if $E(v) = (u,z)$ for some $z \in [L]$ and either

i) node m of u is inactive in v but active in u: $M^{(v)}(u,m) = (u,m)$ but $M^{(u)}(u,m) \neq (u,m)$

ii) node m of u is active in v but inactive in u: $M^{(v)}(u,m) \neq (u,m)$ but $M^{(u)}(u,m) = (u,m)$

Encodings. We think of the function $E : [L] \to [L]^2$ as being encoded as follows: for each $v \in L$, we an $2L$-ary variable E_v naming u, w such that $E(v) = \{u, w\}$. These can be encoding binary by replacing E_v by $2 \log L$-many variables coding E_v. These two encodings are equivalent up to polylog-depth decision tree reductions: from polylog-depth decision trees $T_1, \ldots, T_{2 \log L}$ specifying the values of the variables of the binary encoding we can construct a polylog-depth decision tree T specifying the value of E_v by stacking the (logarithmically-many) trees $T_1, \ldots, T_{2 \log L}$. Conversely, given a polylog-depth decision tree T specifying the value of E_v we can construct decision trees $T_1, \ldots, T_{2 \log L}$, where $T_i(x)$ outputs the i^{th} bit of $T(x)$. For ease of notation we will work with these $2L$-ary variables while implicitly working with these binary variables. Similarly, we have $2 \log L$-many variables E_v naming u, w such that $E(v) = \{u, w\}$. Similarly, for each $v, u \in [L]$ and $m \in [N]$, we have an LN-ary variable $M^{(v)}_{u,m}$ naming $M^{(v)}(u,m)$, which can be encoded with $\log L + \log N$-many variables.

Claim 12. INDLEAF $\in \mathsf{TFNP}^{dt}$.

Proof. The totality of INDLEAF follows from a parity argument. Suppose that there is an instance of INDLEAF that does not have a solution. Observe that for each leaf $l \in [L]$ (with $E(v) = (v,v)$) we have a matching $M^{(l)}$ on the nodes in P_l. Hence, $|P_l|$ is even. We also have that $|P_1|$ is odd. We claim that if $|P_v|$ is odd and $E(v) = (u,w)$ then either $|P_u|$ or $|P_w|$ is odd. Indeed, the matching $M^{(v)}$ between the vertices of $P_v \cup P_u \cup P_w$, as well as the fact that we do not have any inconsistent appearances, ensures that $|P_v| + |P_u| + |P_w| \equiv 0 \bmod 2$.

Beginning at 1, we walk down the dag maintaining that $|P_v|$ is odd for the current vertex v, until we reach a leaf. This is a contradiction, as $|P_v|$ is even for every leaf. Hence, there must be a solution to this instance of INDLEAF. Finally, observe that any solution to INDLEAF can be efficiently verified by a constant number of calls to E and $M^{(m)}$. □

The following theorem states that $\mathbb{F}_2$-PC is characterized by INDLEAF.

Theorem 13. *Let F be any unsatisfiable CNF formula. There is a complexity c reduction from* SEARCH$_F$ *to* INDLEAF *iff* $\mathbb{F}_2$-$\mathsf{PC}(F) = O(c)$.

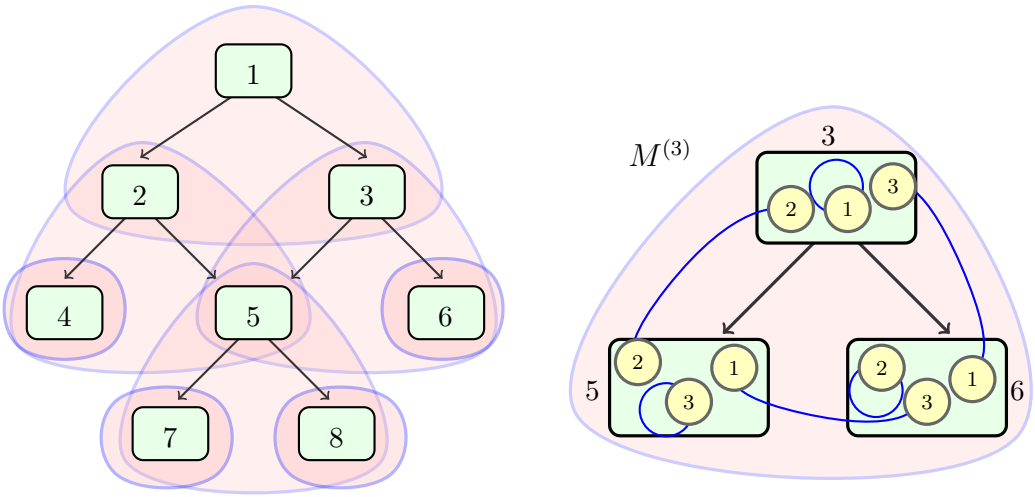


Figure 2: The structure of an IndLeaf instance on $L = 8$ and $N = 3$. Left is the dag in which every pink blob indicates a matching. Here we have, for example, $E(2) = \{4, 5\}$ and $E(7) = \{7\}$. Right is an example of a matching for $M^{(3)}$, where $P_3 = \{2, 3\}, P_5 = \{2, 1\}$, and $P_6 = \{1, 3\}$ as the remaining nodes self-loop, meaning that $M^{(3)}(3, 1) = (3, 1)$, $M^{(5)}(5, 3) = (5, 3)$, and $M^{(6)}(6, 2) = (6, 2)$, and hence these nodes do not appear in their respective pools. The matching $M^{(3)}$ is indicated by the blue edges. Here, $M^{(3)}(3, 2) = (5, 2)$, $M^{(3)}(5, 2) = (3, 2)$, $M^{(3)}(5, 1) = (6, 3)$, $M^{(6)}(6, 3) = (5, 1)$, $M^{(3)}(3, 3) = (6, 1)$, and $M^{(6)}(6, 1) = (3, 3)$.

Dual Variable Encodings. As a corollary, we observe that this theorem holds also for $\mathbb{F}_2$-PCR. This is the extension of $\mathbb{F}_2$-PC to include variables $\overline{x}_i$ for each variable x_i. The benefit of these additional *dual variables* is that it allows us to reduce the number of monomials needed to encode CNF formulas. A clause $C = \bigvee_{i \in I} x_i \vee \bigvee_{j \in J} \overline{x}_j$ is encoded as the polynomial $D_C = \prod_{i \in I} \overline{x}_i \prod_{j \in J} x_j$. The *dual variable encoding* of a CNF formula $F = C_1 \wedge \ldots \wedge C_m$ is the system of polynomials

$$D_F := \{D_{C_i} : i \in [m]\} \cup \{x_i + \overline{x}_i = 1 : i \in [n]\}.$$

The set of polynomials D_F is *unsatisfiable* in the sense that there is no $x \in \{0, 1\}^{2n}$ such that for every $p \in D_F$, $p(x) = 0$.

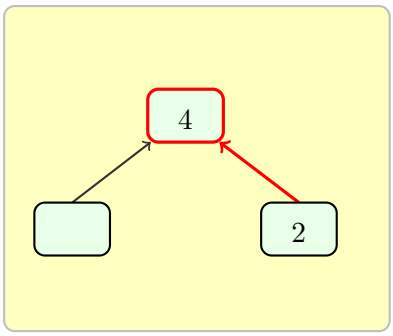
For an unsatisfiable system of polynomials $D = \{p_1, \ldots, p_m\}$ over n variables, the *false polynomial search problem* $\textsc{Search}_D \subseteq \{0, 1\}^n \times [m]$ is defined as

$$(x, i) \in \textsc{Search}_D \iff p_i(x) \neq 0.$$

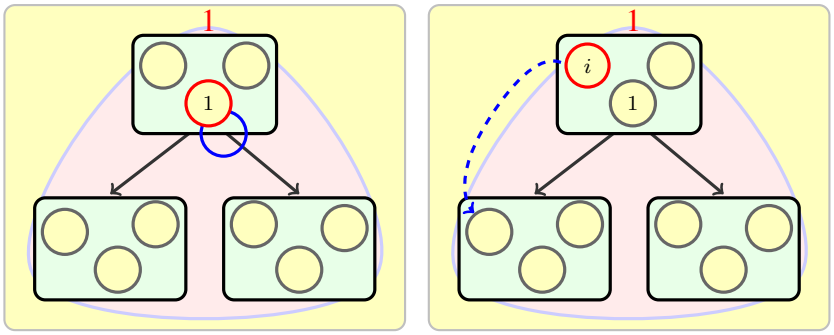
We observe that for any proof system which admits a TFNP^{dt}-characterization, using the dual variable encoding or the regular encoding does not affect complexity. This is in contrast to measuring proofs by their size (number of bits), where we know that dual and regular encodings can be exponentially separated [14].

Observation 14. For any unsatisfiable CNF formula F over n variables, $\textsc{Search}_F =_{dt} \textsc{Search}_{D_F}$.

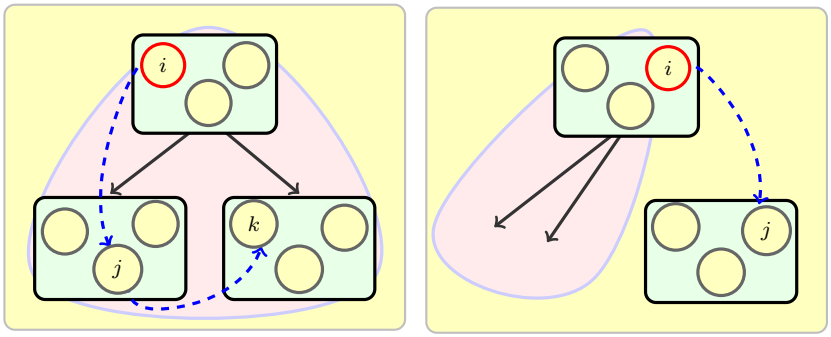
Proof. Let $F = C_1 \wedge \ldots \wedge C_m$. We define a reduction $\{T_i\}_{i \in [2n]}, \{T^o_j\}_{j \in [m+n]}$ from $\textsc{Search}_F$ to $\textsc{Search}_{D_F}$ as follows. For each $i \leq n$ let $T_i(x) = x_i$ and for each $i > n$ let $T_i(x) = 1 - x_i$. For each $i \in [m]$ let $T^o_i(x) = i$ and for each $i > m$ let T^o_j be arbitrary. Observe that this reduction is correct:



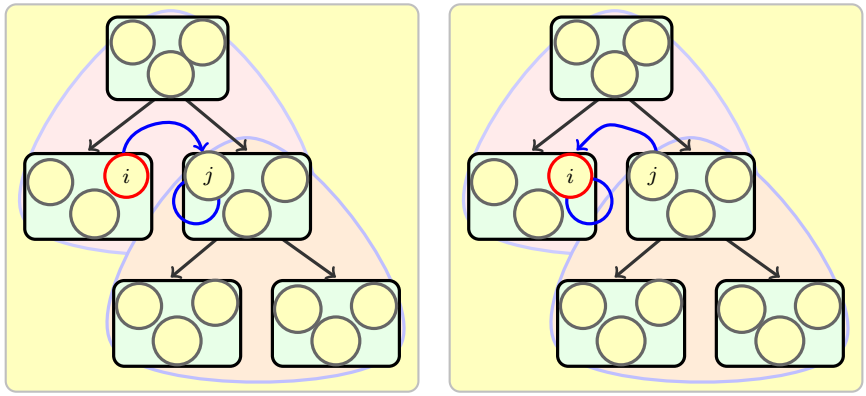
(a) Dag violations.



(b) Root violations. Left: type (i). Right: type (ii).



(c) Matching violations. Left: type (i). Right: type (ii).



(d) Consistency violations. Left: type (i). Right: type (ii).

Figure 3: The solution types of IndLeaf marked in red. A dashed blue line indicates that i points to j ($M^{(v)}(v,i) = (u,j)$, while a solid blue line indicates that there is an edge between i and j ($M^{(v)}(v,i) = (u,j)$ and $M^{(v)}(u,j) = (v,i)$).

for any $x \in \{0,1\}^n$, $T(x) = (x, \overline{x})$ and hence we satisfy every $x_i + \overline{x}_i = 1$. Hence the only falsified polynomials of $D_F(x)$ are of the form D_{C_i} and $D_{C_i}(x, \overline{x}) \neq 0$ iff $C_i(x) = 0$.

For the converse direction, we define the reduction $\{T_i\}_{i\in[n]}, \{T^o_j\}_{j\in[m]}$ as follows. For each $i \in [n]$, T_i queries x_i and $\overline{x}_i$. If $x_i + \overline{x}_i \neq 1$ then we output the constraint $x_i + \overline{x}_i \neq 1$ as a solution to $\textsc{Search}_{D_F}$. Otherwise, $T_i(x, \overline{x}) = x_i$. Finally, $T^o_j(x, \overline{x}) = j$. □

We will now prove Theorem 13, which is broken into Lemma 16 and Lemma 15, and proven over the following subsections.

3.1 PC Proofs Imply Reductions

We begin with the easier direction of Theorem 13.

Lemma 15. *Let F be an unsatisfiable CNF formula. If there is a $\mathbb{F}_2$-Polynomial Calculus proof of F with size L and degree-d then there is a depth $O(d)$-reduction from $\textsc{Search}_F$ to an instance of $\textsc{IndLeaf}$ on $O(L^2)$-many variables.*

An example of this construction is given in Figure 4.

Proof. Let Π be a $\mathbb{F}_2$-PC proof of F and fix a topological ordering of the lines in Π with the first line being the 1 polynomial. Let N be the number of distinct monomials that occur in Π and let L be the number of lines. The $\textsc{IndLeaf}$ instance that we construct will have pools $[L]$ and nodes $[N]$.

Fix any assignment $x \in \{0,1\}^n$ to F. The edges of the dag in the $\textsc{IndLeaf}$ instance will be in one-to-one correspondence with the inferences in Π, with one twist. Let v be a line in Π there are three cases depending on how v was derived:

- If v is an *axiom*, then the decision tree $E(v)(x) = (v, v)$ always. That is, v becomes a leaf of the dag.
- If v was derived by *addition* from u, w in Π then $E(v)(x) = (u, w)$ always.
- If v was derived from u by *multiplication* by a variable x_i, then the decision tree $E(v)$ queries x_i and outputs v if $x_i = 0$ and otherwise it outputs u. That is, v becomes a leaf in the dag if $x_i = 0$.

The intuition behind the multiplication case is that if $x_i = 0$ then v becomes the 0-polynomial under this assignment, as every monomial in v contains x_i.

For every line $v \in [L]$ of Π, there will be a one-to-one correspondence between the monomials in line v that are non-zero under x and the nodes that appear in P_v, and hence we will abuse notation refer to nodes as "monomials". That is,

$$P_v = \{m : \ m \text{ is a monomial in line } v \text{ and } m(x) \neq 0\}. \tag{1}$$

To ensure that this is the case, and that we maintain consistency (i.e., avoid matching violations of type (iii)), for every pool $u \in [L]$ (including v) and every monomial $m \in [N]$ that does not occur in u, we will fix $M^{(v)}(u, m) = (u, m)$ always. Otherwise, if the monomial m occurs in line u, then the decision tree $M^{(v)}(u, m)$ will query the at most d-many bits of x which occur in m in order to check whether $m(x) = 0$. If $m(x) = 0$ then $M^{(v)}(u, m) = (u, m)$. Otherwise, if $m(x) \neq 0$, then we set the matching so that $M^{(v)}(u, m) \neq (u, m)$ as follows:

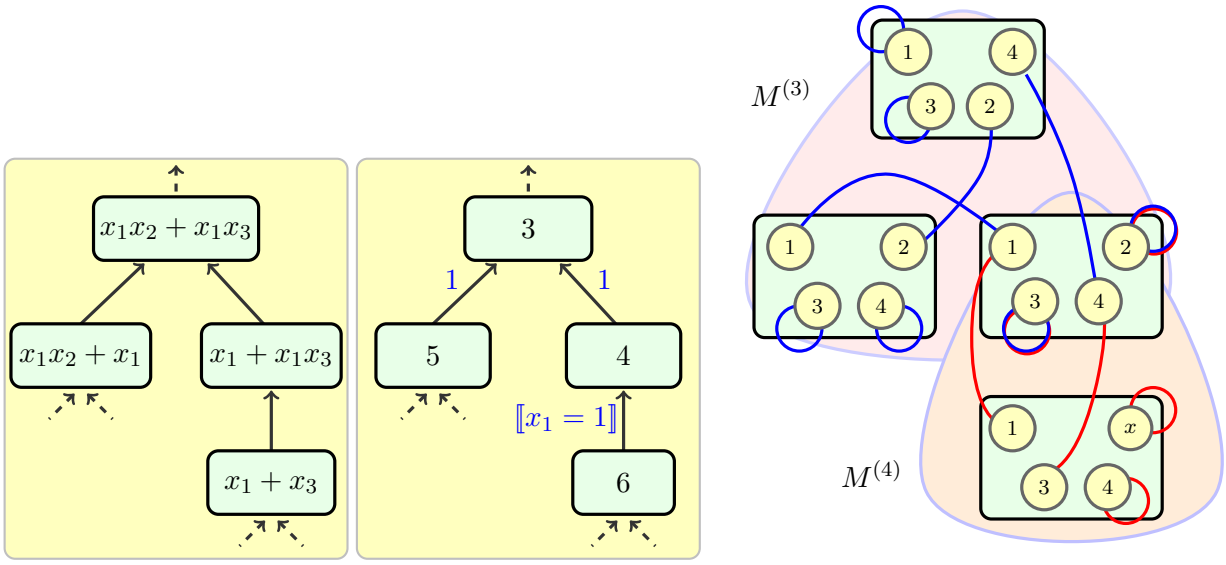


Figure 4: An example part of an $\mathbb{F}_2$-PC proof and the relevant part of the corresponding IndLeaf instance. Left: part of a $\mathbb{F}_2$-PC proof. Middle: the dag of the IndLeaf instance when $x_1 = 1$. Right: the The matching for pool 1 indicated in blue, and the matching for pool 2 indicated in red when $x_1 = 1$.

– *Addition:* If v was obtained by addition from lines u and w then in the matching for v, we will match each of the monomials m in v to the corresponding monomial in u or w that it came from. That is, we set $M^{(v)}(v, m) = (u, m)$ and $M^{(v)}(u, m) = (v, m)$ if m came from u and otherwise, we set it to w.

  Any monomial m that occurs in u (w) but not in v must have cancelled $\bmod 2$ during addition, and so there must be a copy of m in w (u). Hence, the decision trees $M^{(v)}(u, m) = (w, m)$ and $M^{(v)}(w, m)$.

– *Multiplication:* If v is derived from u by multiplication by a variable x_i, then every monomial in v contains the variable x_i. For each $z \in \{u, v\}$ and each monomial m, $M^{(v)}(z, m)$ begins by querying the variable x_i. If $x_i = 0$ then we set $M^{(v)}(z, m) = (z, m)$. Note that this does not cause any consistency violations for u because if $x_i = 0$ then $E(v) = (v, v)$. Therefore, u is not a child of v, and so $M^{(v)}(u, m)$ may be arbitrary.

  Suppose that $x_i = 1$. Observe that for every monomial $x_i m$ in v, exactly one of m or $x_i m$ occurs in u — they cannot both occur, as they would have cancelled $\bmod 2$ when multiplied by x_i and so $x_i m$ would not be in v. Let m' be the one which occurs in v and match $M^{(v)}(v, x_i m) = (u, m')$ and $M^{(v)}(u, m') = (v, x_i m)$. Finally, for any monomial m in u that was not matched in this way, if x_i is not in m then $m x_i$ must also be in u, by the same argument as above. Hence, we match $M^{(v)}(u, m) = (u, m x_i)$ and $M^{(v)}(u, x_i m) = (u, x_i)$. Similarly, if x_i is in m then we match it to the occurrence of $m \setminus x_i$.

– *Axioms:* If v is an axiom in Π, then $v = \overline{C}$ for some clause C of F. Let m be some monomial in $\overline{C}$. The decision tree $M^{(v)}(v, m)$ queries the at most d-many variables in $\overline{C}$ to get some assignment $\alpha \in \{0, 1\}^{\mathsf{Vars}(C)}$. If $C(\alpha) = 1$, and hence $\overline{C}(\alpha) = 0$, then either (1) $m(\alpha) = 0$, or (2) there is another monomial m' in $\overline{C}$ such that $m \restriction \alpha = m' \restriction \alpha$. In the first case, $M^{(v)}(v, m) = (v, m)$ and in the second case $M^{(v)}(v, m) = (v, m')$.

Otherwise, if $C(\alpha) = 1$ then we have found a solution to SEARCH_F. As $C(\alpha) = 1$, there will be an odd number of monomials m in C such that $m \restriction \alpha = 1$ and the remaining monomials m' in C will satisfy $m' \restriction \alpha = 0$. Hence, no matter how we set the matching there will be a matching violation, but for concreteness we pick some monomial m^* such that m^* is not in C, and we match $M^{(v)}(v, m) = (v, m^*)$ for every monomial m in C such that $m \restriction \alpha = 1$.

Any decision tree which has not been defined by this process can be set arbitrarily without causing a violation. This completes the description of the INDLEAF instance. The correctness of this reduction will be guaranteed by the following claim.

Claim. The only solutions are matching violations for pools corresponding clauses C of F falsified by x.

Before proving this claim, we complete the description of the reduction by describing the output decision trees T^o_s for each solution s. By the claim, if s is not a matching violation for pool v corresponding to an axiom of F, then T^o_s can be defined arbitrarily. Let $s = (v, v, m)$ for $v = \overline{C}$ for some clause C of F and monomial m in $\overline{C}$. The decision tree T^o_s outputs the index of the clause C in F.

Proof of Claim. As the final line L of the proof contains only the 1-monomial, there will be no root violations. As we respect the topology of the proof there will also be no dag violations. Consistency violations are avoided as the nodes in each pool are in one-to-one correspondence between the non-zero monomials in the corresponding line in Π. Matching violations at non-leaf nodes are avoided because we match monomials to the corresponding monomial in the line that they came from in Π. Therefore, the only potential solutions are matching violations at the leaves. By construction, the matchings corresponding to clauses C of F that are satisfied under x are correct, and hence the only violations occur at pools corresponding to falsified clauses of F. □

3.2 Reductions Imply PC Proofs

We begin with some terminology. A *conjunct* $\bigwedge_i x_i \wedge \bigwedge_j \neg x_j$ is represented as the polynomial $\prod_i (1 - x_i) \prod_j x_j$. For conjuncts C, C', we say that $C \subseteq C'$ if every literal of C is contained in C'. For a decision tree T with output in $[k]$, we associate it with the sum of conjuncts

$$[\![T = k]\!] := \sum_{k\text{-path } p \in T} p,$$

where a k-path is a root-to-leaf path p ending in a leaf labelled k, and p is associated with the product of the literals occurring along the path p. For a conjunct C, say that $C \subseteq [\![T = k]\!]$ if $C \subseteq p$ for all k-paths $p \in T$. If T_i is the decision tree for some variable x of INDLEAF, we write $[\![x = k]\!] := [\![T_i = k]\!]$. We will often make use of the fact that summing over every root-to-leaf path in a decision tree gives the 1-polynomial:

$$\sum_{j \in [k]} [\![x_i = j]\!] = \sum_{p \in T_i} p = 1.$$

Lemma 16. *Let F be an unsatisfiable CNF formula. If SEARCH_F is reducible to an instance of* INDLEAF *on n variables using decision trees of depth at most d then there is an $O(d)$-degree and size-$n^2 2^{O(d)}$ $\mathbb{F}_2$-PC proof of F.*

Proof. Suppose that $F = C_1 \wedge \ldots \wedge C_m$ is an unsatisfiable CNF formula and $\textsc{Search}_F$ reduces to $\textsc{IndLeaf}_n$ using decision trees $\{T_i\}, \{T^o_j\}$ of depth at most d. Let L be the number of pools and N the number of nodes of the $\textsc{IndLeaf}_n$ instance. For each pool $v \in [L]$ consider the polynomial

$$P_v := \sum_{m \in [N]} [\![M^{(v)}(v,m) \neq (v,m)]\!] = \sum_{m \in [N]} \sum_{(u,\gamma) \neq (v,m)} [\![M^{(v)}(v,m) = (u,\gamma)]\!],$$

As we are working over $\mathbb{F}_2$, the value of $P_v(x)$ records whether the number of monomials in v, $|P_v|$, is even or odd. To prove F using this reduction we will follow the proof of totality of $\textsc{IndLeaf}$. Beginning from L, we will derive that $P_v = 0$ for $v = L, \ldots, 1$.

Claim 17. *For each $v \in [L]$, there is a degree $O(d)$, size $NL2^{O(d)}$ $\mathbb{F}_2$-PC derivation of the polynomial P_v from F.*

Using this claim, we derive that $P_1 = 0$. This suffices in order to complete the proof, as the axioms of $\textsc{IndLeaf}$ enforce that $|P_1|$ is odd.

Claim 18. *There is a degree $O(d)$, size $N2^{O(d)}$ $\mathbb{F}_2$-PC derivation of the polynomial $P_1 - 1$ from F.*

Having derived the polynomials P_1 and $P_1 - 1$, summing them over $\mathbb{F}_2$ gives 1, completing the proof. □

We will now prove the outstanding claims, which will rely on the following Key Observation. Let β be a solution to $\textsc{IndLeaf}$ (either a dag, root, matching, or consistency violation), the *indicator* of the solution β is the minimal conjunct I_β such that $I_\beta(x) = 0$ iff $x \restriction \mathsf{Vars}(\beta) = \beta$.

Key Observation: If P is any conjunct such that there is a solution β with $I_\beta \subseteq P$ then P has a $\mathbb{F}_2$-PC derivation of degree $O(d + \deg(P))$ and size $O(|P|2^d)$ from the axioms of F.

Proof. Let T^o_β be the output decision tree for the solution β. As $I_\beta \subseteq P$, any assignment x which falsifies P must witness the solution β. Let k the index of the clause of F output by $T^o_\beta(x)$. By the soundness of the reduction, $\overline{C}_k(x) = 0$. Hence, whenever $P \cdot [\![T^o_\beta = k]\!]$ is falsified, $\overline{C}_k$ must be falsified as well, and so we must have $\overline{C}_k \subseteq P \cdot [\![T^o_\beta = k]\!]$. Putting this together,

$$\begin{aligned} P &= \sum_{k \in [m]} P \cdot [\![T^o_\beta = k]\!] && \text{(Summing all the root-to-leaf paths in } T^o_\beta \text{ gives 1)} \\ &= \sum_{k \in [m]} \overline{C}_k = 0. && (\overline{C}_k \text{ are axioms of } F) \end{aligned}$$

As T^o_β has depth at most d, it has at most 2^d-many leaves and the bound on the degree and size follows. □

Proof of Claim 18. It suffices to express

$$P_1 - 1 := \sum_{m \in [N]} \sum_{(u,\gamma) \neq (v,m)} [\![M^{(1)}(1,m) = (u,\gamma)]\!] - 1$$

as a low-degree polynomial in which every term involves one of the axioms $\overline{C}_i$ of F. Consider any term of the form $[\![M^{(1)}(1,m) = (u,\gamma)]\!]$ for $u \neq 1$ or $\gamma \neq m$, or $[\![M^{(1)}(1,1) = (1,1)]\!]$. Any assignment to the variables of F which satisfies one of these terms is a root violation, and hence a $(1,m)$ is a solution to

INDLEAF. Since summing over all root-to-leaf paths in the decision tree $M^{(1)}(1,1)$ gives the 1 polynomial, we have

$$\begin{aligned}
0 &= \sum_{u\in[L]}\sum_{\gamma\in[N]} [\![M^{(1)}(1,1)=(u,\gamma)]\!] - 1 \\
&= \sum_{(u,\gamma)\neq(1,1)} [\![M^{(1)}(1,1)=(u,\gamma)]\!] + [\![M^{(1)}(1,1)=(1,1)]\!] - 1 \\
&= \sum_{(u,\gamma)\neq(1,1)} [\![M^{(1)}(1,1)=(u,\gamma)]\!] + 0 - 1 && \text{(Key Observation)} \\
&= \sum_{(u,\gamma)\neq(1,1)} [\![M^{(1)}(1,1)=(u,\gamma)]\!] + \sum_{m\neq 1}\sum_{(u,\gamma)\neq(1,m)} [\![M^{(1)}(1,m)=(u,\gamma)]\!] - 1 && \text{(Key Observation)} \\
&= \sum_{m\in[N]}\sum_{(u,\gamma)\neq(1,m)} [\![M^{(1)}(1,m)=(u,\gamma)]\!] - 1 \\
&= P_1 - 1.
\end{aligned}$$

This polynomial is composed of a sum of $O(N)$-many pairs of decision trees (the second decision tree coming being the output decision tree from the Key Observation), each of depth d and therefore at most 2^d-many leaves. Hence, this is a proof of degree $O(d)$ and size at most $N2^{O(d)}$. □

Proof of Claim 17. We will give a size $N2^{O(d)}$ and degree $O(d)$ proof of the polynomial

$$P_v + \sum_{u,w>v} [\![E(v)=(u,w)]\!]\Big(P_u + P_w\Big) \tag{2}$$

from the clauses of F, expressing that the matching for pool v does not contain any violations. Having derived (2) for all $v \le L$ the claim follows by induction on $v = L,\dots,1$. For $v = L$, observe that (2) is simply P_L as there are no $u, w > L$, and hence by (2) we have a small proof of our base case from the clauses of F. For $v < L$, suppose that we have derived $P_u = 0$ for all $u > v$. Multiplying them we obtain

$$\sum_{u,w>v} [\![E(v)=(u,w)]\!]\Big(P_u + P_w\Big).$$

Adding this to (2) over $\mathbb{F}_2$ gives P_v.

It remains to give a small proof of (2) from the clauses of F. Consider any term $[\![E(v)=(u,w)]\!]$ where at least one of u, w is greater than v. This is a dag-violation and so by the Key Observation we have a small proof of this term from F. Using this,

$$
\begin{aligned}
(2) &= P_v\Big(\sum_{u,w\in[L]} [\![E(v)=(u,w)]\!]\Big) + \sum_{u,w>v} [\![E(v)=(u,w)]\!]\Big(P_u+P_w\Big) && \text{(As } \textstyle\sum_{p\in E(v)} p = 1\text{)}\\
&= P_v\Big(\sum_{u<v\vee w<v} [\![E(v)=(u,w)]\!]\Big) + \sum_{u,w>v} [\![E(v)=(u,w)]\!]\Big(P_v+P_u+P_w\Big)\\
&= 0 + \sum_{u,w\geq v} [\![E(v)=(u,w)]\!]\Big(P_v+P_u+P_w\Big), && \text{(Key Observation)}\\
&= \sum_{u,w\in[L]} [\![E(v)=(u,w)]\!] \sum_{z\in\{u,v,w\}} P_z\\
&= \sum_{u,w\in[L]} [\![E(v)=(u,w)]\!] \sum_{z\in\{u,v,w\}} \sum_{m\in[N]} [\![M^{(z)}(z,m)\neq(z,m)]\!], && (3)
\end{aligned}
$$

where the last line follows by the definition of P_z.

In order to derive (3) from the axioms we will define the following polynomial which can be read as the sum of all of the indicators of valid matchings for pool v.

$$
\mathsf{match}^v := \sum_{u,w\in[L]} [\![E(v)=(u,w)]\!] \sum_{m\in[N]} \sum_{z\in\{u,v,w\}} [\![M^{(v)}(v,m)\neq(v,m)]\!]\cdot \mathsf{match}^v_{z,m}.
$$

We will define the polynomial $\mathsf{match}^v_{z,m}$ shortly. However, for now we note that $\mathsf{match}^v_{z,m} \equiv 1$ as it will be the sum of all root-to-leaf paths of in a sequence of decision trees. Then we can write

$$
(3) = \sum_{u,w\in[L]} [\![E(v)=(u,w)]\!] \sum_{m\in[N]} \sum_{z\in\{u,v,w\}} [\![M^{(v)}(v,m)\neq(v,m)]\!]\cdot \mathsf{match}^v_{z,m} = \mathsf{match}^v.
$$

Hence, it suffices to show that there is an efficient proof of match^v from the axioms of F.

Define

$$
\mathsf{match}^v_{z,m} :=
$$
$$
\sum_{a\in[L]} \sum_{\alpha\in[N]} [\![M^{(v)}(z,m)=(a,\alpha)]\!] \sum_{z^*\in[L]} \sum_{m^*\in[N]} [\![M^{(v)}(a,\alpha)=(z^*,m^*)]\!] \sum_{b\in[L]} \sum_{\beta\in[N]} [\![M^{(a)}(a,\alpha)=(b,\beta)]\!]
$$

which records the possible matchings of node m in pool z in the matching for v. The first term asks for the node (a,α) that the node m from pool z is matched to in the matching for z, the second term asks what the node α from pool a is matched to, and the last term checks whether node α is *active* in pool a. Note that each term is a sum over all of the paths in a decision tree, $\mathsf{match}^v_{z,m} = 1$.

To show that match^v has a proof from F, we will partition match^v into two multisets of conjuncts, V which contains the conjuncts which witness *violations*, and C which correspond to *correct* matchings — those without violations. That is, C is the multiset of conjuncts witnessing $(z^*,m^*) = (a,\alpha)$ and $M^{(a)}(a,\alpha)\neq(a,\alpha)$, and E are the remaining conjuncts in match^v. The conjuncts in V can be removed by the Key Observation, while we argue that each conjunct in C occurs an even number of times in match^v, as

(z, m) is matched to (a, α) iff (a, α) is matched to (z, m).

$$\begin{aligned}
\mathsf{match}^v &= \sum_{t \in C} t + \sum_{t \in V} t \\
&= \sum_{t \in C} t + 0 && \text{(Key Observation)} \\
&= \sum_{u,w > v} [\![E(v) = (u, w)]\!] \sum_{m \in [N]} \sum_{z \in \{u,v,w\}} [\![M^{(v)}(v, m) \neq (v, m)]\!] \cdot \\
&\quad \sum_{a \in [L]} \sum_{\alpha \in [N]} 2 \cdot [\![M^{(v)}(z, m) = (a, \alpha)]\!][\![M^{(v)}(a, \alpha) = (z, m)]\!][\![M^{(v)}(a, \alpha) \neq (a, \alpha)]\!] \\
&= 0. && \text{(Working over } \mathbb{F}_2\text{)}
\end{aligned}$$

To bound the size and degree, observe that each term of $\mathsf{match}^v_{z,m}$ is the product of three decision trees each of depth at most d, and hence has size at most 2^{3d} and depth $3d$. The application of the key lemma introduces an additional depth$\leq d$ output decision tree. The polynomial match^v is formed from $\mathsf{match}^v_{z,m}$ for every $m \in [N]$ by multiplying it two decision trees of depth at most d. Hence, each match^v can be derived in size $O(N2^{6d})$ and degree $6d$. □

3.3 PC Proves its own Soundness

In this section we show that the $\mathbb{F}_2$-Polynomial Calculus can prove a reflection principle about itself.

Theorem 19. *There is a reflection principle* $\mathsf{Ref}_{\mathsf{PC}}$ *for the* $\mathbb{F}_2$*-Polynomial Calculus such that* $\mathsf{PC}(\mathsf{Ref}_{\mathsf{PC}}) \leq \mathsf{polylog}(n)$.

We define $\mathsf{Ref}_{\mathsf{PC}}$ by giving a natural verification procedure for $\mathbb{F}_2$-PC proofs. To do so, for simplicity it will be convenient to combine the $\mathbb{F}_2$-PC rules into a single inference rule: for any line v_1, v_2,

$$\frac{v_1, \quad v_2}{v_1 x + v_2 y},$$

for any $x, y \in \{0, 1, x_1, \ldots, x_n\}$. It is easy to see that this will change the complexity of the proof by at most 2.

A Verification Procedure for $\mathbb{F}_2$-PC. We define the reflection principle by giving a verification procedure $V^{\mathsf{PC}}(H, \Pi) := V^{\mathsf{PC}}_{n_H, m_H, c}(H, \Pi)$. For simplicity of exposition, we will have break the complexity of the proof into three parameters, $N = 2^c$ which denotes the number of monomials, $L = 2^c$ which denotes the number of lines, and $d = c$ which denotes the degree of the proof.[3] As before, n_H and m_H are the number of variables and clauses of the CNF formula H.

Variables: V^{PC} describes the proof Π with the following variables. Each line $v \in [L]$ is given by a length-N vector $\mathsf{mon}^{(v)} \in ([n_H + 1]^d)^N$, where the m^{th} entry $\mathsf{mon}^{(v)}_m \in [n_H + 1]^d$ names the at most d variables in the m^{th} monomial in the v^{th} line. The $(n_H + 1)$-st value is understood to indicate the constant 1. Not every

[3] Note that we could have used a size parameter s, rather than N and L, counting the bit-complexity of the proof (roughly the number of monomials and the number of lines). However, this would only change the complexity of the PC-proof in Theorem 19 by log-factors, and it is notationally simpler to work with parameters N and L.

line contains N monomials and so for each $m \in [N]$ we include a variable $a_m^{(v)} \in \{0,1\}$ which indicates whether the i^{th} monomial is *active* — that is, whether it occurs in line v.

The lines $L - m_H + 1, \ldots, L$ are reserved to describe the clauses of H. Every other line $v \leq L - m_H$ has a pair of predecessor pointers $p_1^{(v)}, p_2^{(v)} \in [v-1]$, naming the two previous lines from which v was derived, and a pair of variable pointers $v_1^{(v)}, v_2^{(v)} \in [n_H + 2]$ naming the respective variables multiplying previous lines in the derivation of v; the values $n_H + 1$ and $n_H + 2$ indicate the constants 1 and 0. Together, these variables state that the line v was derived as $p_1^{(v)} v_1^{(v)} + p_2^{(v)} v_2^{(v)}$.

To ensure that each inference in the proof is correct, for each line $v \leq L - m_H$ there is a matching between the active monomials in v, $p_1^{(v)}$, and $p_2^{(v)}$. The matching for line v is given by $3N$-many variables $\mathsf{match}^{(v)}(\delta, m) \in \{0,1,2\} \times [N]$ for $\delta \in \{0,1,2\}$ and $m \in [N]$, where $\delta \in \{0,1,2\}$ indicates whether m is belongs to v (0), the left child (1) of v, or the right child (2), and $\mathsf{match}^{(v)}(\delta, m)$ names the monomial that c's copy of m is matched to.

Constraints: The constraints of V^{PC} are the following:

- Final Line. There are constraints saying that the only monomial in line 1 is the 1-monomial. That is, $\mathsf{mon}_1^{(1)} = 1$, and for all $m \neq 1$, $a_m^{(1)} = 0$.
- Axioms. For $i \in [m_H]$ and $v = L - i$, we have constraints stating that the v^{th} line is exactly the i^{th} clause of H. In particular, let k be the number of positive literals in C_i of H (of which there are at most d), then we have constraints saying that exactly the first 2^k monomials are active; that is, $a_1^{(v)}, \ldots, a_{2^k}^{(v)} = 1$. As well, we have constraints saying that the m^{th} monomial of v is exactly the m^{th} monomial in the monomial-expansion of $\overline{C}_i$. To see that each of these can be expressed by a small CNF formula, observe that this constraints on the value of $a_i^{(v)}$ and $\mathsf{mon}_m^{(v)}$ can be defined by a decision tree querying the variables $C_{i,j}$ for $j \in [d]$ of Sat.
- Inferences. For each line $v \leq L - m_H$ we have constraints saying that $p_1^{(v)}, p_2^{(v)} \geq v$, and a set of constraints saying that v was correctly derived from $p_1^{(v)}, p_2^{(v)}$. In particular, that the active monomials of v are exactly the monomials in $p_1^{(v)} v_1^{(v)} + p_2^{(v)} v_2^{(v)}$ after cancelling $\mathrm{mod}2$. More concretely, we have constraints stating that if an active monomial m from one of $v, p_1^{(v)}, p_2^{(v)}$ is matched to another monomial m' then m' has to be matched back to m and the variables in m have to agree over $v_1^{(v)}, v_2^{(v)}$. That is, if m is a monomial from v and m' is a monomial from $p_1^{(v)}$ then $m = v_1^{(v)} m'$. We also have constraints forcing that active monomials are only matched to active monomials, and no active monomial is matched to itself.

The reflection principle for $\mathbb{F}_2$-PC will be $\mathsf{Ref}^{\mathsf{PC}}(H, \Pi) := \mathsf{Sat}(H, \Pi) \wedge V^{\mathsf{PC}}(H, \Pi)$. As every constraint mentions $O(d \log n_H)$-many variables, this can be encoded as a CNF formula of width $O(d \log n_H)$.

Proof of Theorem 19. By Theorem 13, it suffices to construct a reduction from $\textsc{Search}_{\mathsf{Ref}^{\mathsf{PC}}}$ to $\textsc{IndLeaf}$. Let $n_H, m_H, (d, N, L)$ be the parameters of $\mathsf{Ref}^{\mathsf{PC}}$. We construct an instance of $\textsc{IndLeaf}$ with L pools and N nodes. The high-level idea is simple: $\mathsf{Ref}^{\mathsf{PC}}$ is already almost an instance of $\textsc{IndLeaf}$, with only a few major differences:

- V^{PC} does not specify a matching for the axioms of H.
- $\mathsf{Ref}^{\mathsf{PC}}$ contains the additional Sat formula.

– Each node $m \in [N]$ no longer corresponds to a monomial, but rather has a set of variables $\mathsf{mon}_m^{(v)}$, naming which monomial it is.

Hence, given an assignment (H, Π) to $\textsc{Search}_{\mathsf{Ref}^{\mathsf{PC}}}$ we construct the decision trees of the reduction as follows:

– *Edges.* For any $v \in [L - m_H]$, the decision tree $E(v)$ queries the variables for $p_1^{(v)}, p_2^{(v)}$ and outputs $E(v) = (p_1^{(v)}, p_2^{(v)})$. For $L - m_H + 1 \leq v \leq L$, define $E(v) := (v + m_H, v + m_H)$. Finally for $L + 1 \leq L + m_H$ let $E(v) := (v, v)$.

– *Non-Leaf Matchings.* For every $v \leq L - m_H$ and $u \geq [L]$ and $m \in [N]$, the decision tree $M^{(v)}(u, m)$ queries $p_1^{(v)}, p_2^{(v)}$ to determine the two children u_1, u_2 of v, and $a_m^{(u)}$ to determine if m is active. If either $u \notin \{u_1, u_2\}$ or $a_m^{(u)} = 0$ then $M^{(v)}(u, m) = (u, m)$. Otherwise, decision tree asks for the value of $\mathsf{match}^{(v)}(u, m) = (c', m') \in \{0, 1, 2\} \times [N]$ and $\mathsf{match}^{(v)}(u_{c'}, m') = (c^*, m^*)$, where $u_0 := v$, and also for the variables in these monomials $\mathsf{mon}_m^{(u)}$ and $\mathsf{mon}_{m'}^{(u_{c'})}$, and $v_c^{(v)}$ for the variable used to derive m from m'. If either we discover that (u, m) is matched to itself ($m' = m$ and $u_{c'} = u$) or if the variables in m do not agree with the variables of the monomial it was derived from ($\mathsf{mon}_m^{(u)} \neq v_c^{(v)} \mathsf{mon}_{m'}^{(u_{c'})}$) then we introduce a matching violation by sending $M^{(v)}(u, m) = (w, m)$ for some $w \notin \{u_1, u_2\}$. Otherwise, we set $M^{(v)}(u, m) = (u_{c'}, m')$.

– *Leaf Matchings.* For any $i \in [m_H]$, $v = L - i + 1$, and $m \in [N]$, the decision tree $M^{(v)}(v, m)$ queries the variables $C_{i,j}$ for $j \in [d]$. Let k be the number of positive literals in C_i, and note that $\overline{C}_i$ has 2^k-many monomials. If $a_m^{(v)} = 0$ and $m > k$ then set $M^{(v)}(v, m) = (v, m)$. Otherwise, if $m > 2^k$ and $a_m^{(v)} = 1$ or if $m \leq 2^k$ but $a_m^{(v)} = 0$ then we pick some $w \neq v$ and set $M^{(v)}(v, m) = (w, m)$, forming a matching solution. If neither of these hold, then query $\mathsf{mon}_m^{(v)}$ to determine the variables in the m^{th} monomial. If these disagree with the variables in C_i, then we violate an *axiom constraint* of $\mathsf{Ref}^{\mathsf{PC}}$ and we set $M^{(v)}(v, m) = (w, m)$ for some $w \neq v$ in order to cause a solution. Finally, query the bits of the assignment α to Sat that are mentioned by C_i. If we discover that $C_i(\alpha) = 0$, then again we cause a solution by setting $M^{(v)}(v, m) = (w, m)$ for some $w \neq v$. Otherwise, if $C_i(\alpha) = 1$ then either $\mathsf{mon}_m^{(v)} \restriction \alpha = 0$, in which case we set $M^{(v)}(v, m) = (v, m)$, and otherwise, as $\overline{C}_i \restriction \alpha = 0$, there must exist another monomial in $\overline{C}_i \restriction \alpha$ which is equal to $\mathsf{mon}_m^{(v)} \restriction \alpha$; let this be the t^{th} monomial of $\overline{C}_i$. The decision tree outputs $M^{(v)}(v, m) = (v, t)$.

It remains to define the output decision tree T_s^o for each solution s to $\textsc{IndLeaf}$.

– *Dag Violations.* Let there be a dag violation at v. Then, as the edges of the $\textsc{IndLeaf}$ are exactly those in the inference structure of the $\mathsf{Ref}^{\mathsf{PC}}$ instance, there must one of $p_1^{(v)}, p_2^{(v)}$ which outputs a value less than v, and hence violates one of the inference constraints of V^{PC}. Hence, T_v^o queries $p_1^{(v)}, p_2^{(v)}$ and outputs the index of that falsified constraint.

– *Root Violations.* If $(1, m)$ is a root violation, then the decision tree $T_{(1,m)}^o$ queries $a_m^{(1)}$, $\mathsf{mon}_m^{(1)}$ and outputs the violated final line constraint.

– *Matching Violations.* If (v, u, m) is a matching violation, the decision tree $T_{(v,u,m)}^o$ queries the same variables as the decision tree $M^{(v)}(u, m)$ and outputs the appropriate violated inference constraint if v is not an axiom, or if v is an axiom then it outputs the appropriate violated axiom constraint or constraint of Sat.

– *Consistency Violations.* There will be no consistency violations in the reduction as it is defined, and so we let T^o_s be arbitrary for each potential consistency violation s.

This completes the reduction. □

3.4 Characterizing Dynamic Variants of Static Systems

We end this section by discussing how one can define *induction* variants of TFNP problems (such as LEAF) which characterize static proof systems (such as Nullstellensatz), in order to characterize their dynamic variants. Consider some TFNP class which characterizes a static proof system and let R be a complete problem for this class. We build a problem INDR to characterizes the dynamic variant of this proof system as follows: there are L *pools*, which correspond to the lines of the proof, and a dag structured defined by *edge variables* $E^{(m)}$ for each $m \in [L]$, which define the children $m_l, m_r \geq m$ of m. Each pool has a set of vertices, and we have an instance of the problem R defined over these vertices, where, as before, we restrict the vertices in each pool m to only interact with vertices in neighbouring pools. The crucial part is that the edges of this dag (and hence how vertices in different pools interact) is determined by variables, rather than being fixed a-priori.

Using this template, we develop characterizations of the $\mathbb{F}_q$-Polynomial Calculus, the unary Polynomial Calculus, and unary dag-like Sherali-Adams.

Polynomial Calculus over Finite Fields.

Kamath [34] showed that a $\mathbb{F}_q$-variant of PPA, where one uses q-matchings, rather than matchings, characterizes $\mathbb{F}_q$-Nullstellensatz. It is straightforward to generalize INDLEAF to characterize $\mathbb{F}_q$-PC. The INDLEAF$_q$ problem will be defined as INDLEAF, except one now matches q-tuples, rather than pairs. More specifically, in the matching for each pool $v \in [L]$, each active node m is either matched in a q-tuple with, or is matched in a 2-tuple, with the restriction that if m is matched in a 2-tuple with another node m' then m must belong to pool v and m' must belong to one of the children u of v. The meaning of the two edges is that the monomial m is being "carried forward" from line u to line v in the $\mathbb{F}_q$-PC proof.

Unary Polynomial Calculus.

The *unary Polynomial Calculus* (uPC) proof system is the Polynomial Calculus system 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. That is, if αm occurs in some line in the proof, for $\alpha \in \mathbb{Z}$, then it contributes $|\alpha|$ towards the size.[4]

Göös et al. [27] characterized the unary Nullstellensatz proof system by PPAD, the class of total search problems reducible to END-OF-LINE. We extend this to characterize uPC by IND-PPAD.

End-of-Line. An instance of END-OF-LINE$_n$ is given by *predecessor* and *successor* pointers $p : [n] \to [n]$ and $s : [n] \to [n]$. We view the pointers as defining a degree≤ 2 graph on the vertices $[n]$ where there is a directed edge (u, v) iff $s(p(v)) = v$ and $p(s(u)) = u$. A vertex v is a solution if:

– *Root Violations.* $p(1) \neq 1$ or $s(1) = 1$.

[4] Typically PC over $\mathbb{Z}$ is defined with a multiplication rule whereby one may derive, from previous lines p, q any line of the form $\alpha p + \beta q$ for $\alpha, \beta \in \mathbb{Z}$. As we are measuring coefficients in unary, this operation may be simulated by repeated addition.

– *Sink Violations.* $v \in V$ such that $p(s(v)) \neq v$, meaning that v is a sink.

– *Source Violations.* $v \neq 1$ such that $s(p(v)) \neq v$, meaning that v is a source.

The IND-PPAD class, which characterizes uPC is defined by its complete problem INDEND-OF-LINE. The major difference between this and INDLEAF is that the matchings for each pool are now directed. The direction of an edge will be indicated by the head-vertex being assigned "+" and the tail being assigned "−"; that is, if $M^{(v)}(u,m) = (u',m',+)$ and $M^{(v)}(u',m',-)$, then we interpret this as an arrow from u (the tail) to u' (the head). We will enforce consistency between pools: if a node m of pool u is at the head of an edge in the matching for v then it must be at the tail of an edge in the matching for u.

The direction of the edges corresponds to the polarity of the monomials in the uPC proof: let m be a node belonging to some pool v. Then, the sign of the monomial corresponding to m in line v in the uPC proof is determined by whether it appears at the head (positive) or tail (negative) of an edge in the matching for pool v. The consistency conditions ensure that the signs are maintained in each application of a uPC rule.

Induction End-of-Line. An instance of INDEND-OF-LINE (INDEOL) for parameters L, N is specified by functions $E : [L] \to [L] \times [L]$ and $M^{(v)} : [L] \times [N] \to [L] \times [N] \times \{-,+\}$ for each $v \in [L]$. We interpret E and $M^{(v)}$ as defining the same structure as in INDLEAF, with the following modification to the matching structure. In the following, $*$ denotes that the value is arbitrary.

– *The Matching.* For each pool v,

$$P_v := \{m \in [N] : M^{(v)}(v,m) \neq (v,m,*)\}$$

is the set of nodes which appear in v. Let $E(v) = (u,w)$ be the children of v. We have a directed matching between the nodes of $P_v \cup P_u \cup P_w$. This matching is given by the function $M^{(v)}$, and we say that $m \in P_i$ and $\gamma \in P_j$ are *matched*, for $i,j \in \{u,v,w\}$, if $M^{(v)}(i,m) = (j,\gamma,b)$ and $M^{(v)}(j,\gamma) = (i,m,\overline{b})$ for $b \in \{+,-\}$ and $\overline{b} = \{+,-\} \setminus b$. In this case, we say that (j,γ) is labelled b and (i,m) is labelled $\overline{b}$, and we view this as a directed edge from the node labelled "−" to the node labelled "+".

The solution-types are the same as INDLEAF, with an additional type of consistency violation.

– *Dag Violations.* A pool $v \in [L]$ such that $E[v] = (u,w)$ and either $u < v$ or $w < v$.

– *Root Violations.* Any violation to $P_1 = \{1\}$.

  i) $(1,1)$ is a solution if $M^{(1)}(1,1) = (1,1,*)$ or $M^{(1)}(1,1) = (u,m,+)$ for some $(u,m) \neq (1,1)$.
  ii) $(1,m)$ is a solution if $M^{(1)}(1,m) \neq (1,m,*)$.

– *Matching Violations.* (v,u,m) is a solution if $u \in E(v) \cup \{v\}$, and $m \in P_u$, and either

  i) m is matched to a node in a pool $w \notin E(v) \cup \{v\}$: $M^{(v)}(u,m) = (w,*,*)$.
  ii) The node which m is matched to is not matched to m: $M^{(v)}(u,m) = (u',m',b)$ and $M^{(v)}(u',m') \neq (u,m,\overline{b})$, for some $u' \in E[v] \cup \{v\}, m' \in [N], b \in \{+,-\}$.

– *Consistency Violations.* A node m of pool u occurs in the matching for v iff it occurs in the matching for v, and its sign is consistent. That is, (v,u,m) is a solution if $u \in E(v)$ and either

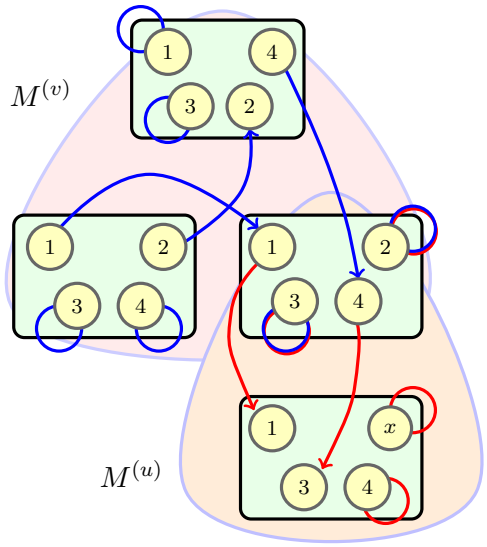


Figure 5: Two adjacent pools of an IndEOL instance. The red arrows indicate the $M^{(u)}$, while the blue arrows indicated $M^{(v)}$. There are no consistency violations present, as each node in P_u which is at the tail of an arrow in $M^{(v)}$ is at the head of one in $M^{(u)}$, and vice-versa.

i) node m of u is inactive in v but active in u: $M^{(v)}(u,m) = (u,m,*)$ but $M^{(u)}(u,m) \neq (u,m,*)$

ii) node m of u is active in v but inactive in u: $M^{(v)}(u,m) \neq (u,m,*)$ but $M^{(u)}(u,m) = (u,m,*)$.

iii) node m changes polarity between pools: $M^{(u)}(u,m) \neq (u,m,*)$ and $M^{(v)}(u,m) = (u',m',b)$ and $M^{(u)}(u,m) = (u'',m'',b)$ for $b \in \{-,+\}$, $u', u'' \in [L]$, and $m', m'' \in [N]$.

A portion of an instance of IndEOL is depicted in Figure 5.

Theorem 20. *Let F be any unsatisfiable CNF formula. There is a complexity c reduction from* $\textsc{Search}_F$ *to* IndLeaf *iff* $\mathbb{F}_2\text{-}\mathsf{PC}(F) = O(c)$.

We defer the proof of this theorem to the Appendix.

Unary DAG-Like Sherali-Adams.

The *unary dag-like Sherali-Adams* proof system generalizes both the uPC and (unary) Sherali-Adams proof systems (see e.g., [20] for a definition). Briefly, Sherali-Adams is an algebraic proof system which allows one to introduce additional conjuncts. Recall that a conjunct $\bigwedge_{i\in I} x_i \wedge \bigwedge_{j\in J} \overline{x}_j$ is encoded as the polynomial $C := \prod_{i\in I} x_i \prod_{j\in J}(1-x_j)$.

Dag-Like Sherali-Adams. The unary dag-like Sherali-Adams (uDSA) proves the unsatisfiability of a CNF formula F by deriving -1 from the polynomials $\{C_i = 0 : C_i \in F\}$ using the *addition* and *multiplication* rules of uPC along with the following additional rule:

– *Conjunct Rule.* From a previously derived polynomial $p \geq 0$, deduce $p + C$ for any conjunct C.

As before, we work over the ideal $\langle x_i^2 - x_i \rangle_{i \in [n]}$, multi-linearlizing implicitly.

We measure the degree of a uDSA proof by the maximum degree of any derived polynomial, and the size of the proof is measured as the number of monomials (with multiplicity) that appear in the proof (that is, we measure coefficients in unary).

Göös et al. [27] characterized unary Sherali-Adams by the class PPADS, whose complete problem is SINK-OF-LINE. The SINK-OF-LINE problem is identical to END-OF-LINE, except that we no longer have *source violations*. We extend this characterizing by defining the class IND-PPADS of everything reducible to the complete problem INDSINK-OF-LINE (INDSOL). Like SINK-OF-LINE, INDSOL restricts the solutions of INDEOL. In particular, INDSOL will permit nodes which occur at the head of arrows to be incorrectly matched. This corresponds to introducing conjuncts via the conjunct rule in a uDSA proof.

Induction Sink-of-Line. An instance of INDSOL is identical to an instance of INDEOL, with the matching and root violations replaced by the following:

- *Root Violations.* Any violation to 1 being the only node in the first pool and occurring at the tail of an arrow (1 is negative)
  - i) $(1,1)$ is a solution if $M^{(1)}(1,1) = (1,1,*)$ or $M^{(1)}(1,1) = (u,m,-)$ for any $u \in [L], m \in [N]$.
  - ii) $(1,m)$ is a solution if $M^{(1)}(1,m) \neq (1,m,*)$.
- *Matching Violations.* (v,u,m) is a solution if $m \in P_u$, and either
  - i) $u \in E(v) \cup \{v\}$ and u is matched to a node in a pool which is not in $E(v) \cup \{v\}$: $M^{(v)}(u,m) = (u',m',b)$ for $u' \notin E(v) \cup \{v\}$.
  - ii) $u \in E(v)$ and the node which m is matched to is not matched to m: $M^{(v)}(u,m) = (u',m',b)$ and $M^{(v)}(u',m') \neq (u,m,\bar{b})$, for some $u' \in E[v] \cup \{v\}, m' \in [N], b \in \{+,-\}$.
  - iii) $u = v$ and m appears at the *tail* of an arrow in $M^{(v)}$ and m is matched to a node which is not matched back to it: that is, $M^{(v)}(v,m) = (w,m',+) \neq (v,m,*)$ and $M^{(v)}(w,m') \neq (v,m,-)$.

The characterization of uDSA by INDSOL follows by a very similar argument to the proof of Theorem 20, combined with the characterization of Sherali-Adams by PPADS [27].

Theorem 21. *Let F be any unsatisfiable CNF formula. There is a complexity c reduction from* SEARCH_F *to* INDSOL *iff* $\text{uDSA}(F) = O(c)$.

4 Communication TFNP and Monotone Circuit Complexity

In addition to proof system characterizations of black-box TFNP subclasses, the *communication* versions of TFNP subclasses have provided characterizations of monotone circuit models [28, 35, 49]. 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 [44]. Conversely, a black-box and communication characterization of the same TFNP subclass automatically gives rise to a

monotone interpolation theorem. In this section, we uncover the exact conditions under which a monotone circuit model has a communication-TFNP characterization.

The first order of business is to define what is meant by a monotone circuit model. This is formalized by the following notion of a *monotone partial complexity measure*.

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, $\mathsf{mpc}(g) \geq \mathsf{mpc}(f)$.[5]

Typical such measures include the minimum monotone circuit, formula, or span program size of a total function that solves f, but we won't include a circuit model explicitly. 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 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. We explore this further in subsection 4.3.

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.

4.1 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.

We are now ready to develop the connection between monotone complexity and a variant of TFNP which uses communication-efficient reductions. Central to this connection is the *Karchmer-Wigderson game*.

Karchmer-Widgerson Game. Let f be a function. 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 $x_i < y_i$.

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

Karchmer and Wigderson [35] showed that communication complexity of KW_f (mKW_f) is an exact characterization of the (monotone) circuit depth needed to compute f, or equivalently FP^{cc}.

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 $R \subseteq X_m \times Y_m \times [\ell]$ *mapping reduces* to another $R' \subseteq X'_n \times Y'_n \times [\ell']$ with communication t if there are functions $M_X : X \to X'$, $M_Y : Y \to Y'$ and a t-bit communication protocol $S(x, y, i')$ which outputs i so that

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

In particular this means that R requires at most t more bits of communication than R' to solve. The *complexity* of this reduction is $t + \log n$, and we say that this reduction is efficient, denoted by $R \leq_{cc} R'$, if the complexity is $O(\mathsf{polylog}(m))$. We say that two communication problems R, R' are *equivalent*, denoted $R =_{cc} R'$ if $R \leq_{cc} R'$ and $R' \leq_{cc} R$.

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

Lemma 22. *For any $R \subseteq \mathsf{TFNP}^{cc}$, there is a partial monotone function f such that $R =_{cc} \mathrm{mKW}_f$.*

Proof. Let $R \subseteq X \times Y \times [\ell]$ and 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. □

We will also need the following notion which will allow us to pad a search problem.

Paddable. 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. For example, it would allow for a class that is essentially PPA for infinitely many sizes and then suddenly switches to the pigeon-hole principle, and back again. Another example of a potential issue is that our class could contain 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, giving the exact conditions under which an mpc characterizes a TFNP^{cc} subclass.

Characterizes. We say that TFNP^{cc} class $\mathcal{C}$ with a complete problem $R = \{R_n\}_{n\in\mathbb{N}}$ *characterizes* a mpc if for every partial monotone function f there is a complexity c reduction from $\textsc{mkw}_f$ to R iff $\mathsf{mpc}(f) = O(\mathsf{polylog}(c))$. As well, say that R characterizes a mpc if the class whose complete problem is R characterizes that mpc.

Before proving 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 23). Using this, we then weaken the requirement to admitting a *universal family* (Theorem 27). In between, we explore sufficient conditions for TFNP^{cc}-characterizations of total functions.

4.2 Complete Problems give TFNP Characterizations

Our first characterization of mpc measures with TFNP^{cc} connections involves two 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 $\mathsf{mpc}(F_m) \leq \mathsf{quasipoly}(m)$.[6]

The following theorem states that these conditions suffice for a TFNP^{cc} characterizations.

Theorem 23. *Let* mpc *be a complexity measure. Then there is a paddable sequence of* TFNP^{cc} *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.*

A key component of the proof is 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.

[6]Note that in the definition of admitting a complete family we 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.

Lemma 24. *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. Define the both of the function M_X and M_Y as h; 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 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 [35], 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 [35], 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 23. 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 $R_m := \text{MKW}_{F_m}$ 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-$\mathsf{polylog}(m')$ reductions. Thus by Lemma 24, $R_n = \text{MKW}_{F_m}$ reduces to $R_{m'} = \text{MKW}_{F_{m'}}$ with communication-$\mathsf{polylog}(m')$ for all such m', and so R is paddable.

We claim that there is a complexity-c reduction from MKW_f to R iff $\mathsf{mpc}(f) = \mathsf{polylog}(c)$. Letting $m = \mathsf{mpc}(f)$, f reduces to F_m with a $\mathsf{polylog}(m)$-depth monotone circuit, as F_m is complete. Hence by Lemma 24, there is a $\mathsf{polylog}(m)$-complexity reduction from MKW_f to MKW_{F_m} with $\mathsf{polylog}(m)$. It follows by definition that $R^{cc}(\text{MKW}_f) \leq \mathsf{polylog}(m) = \mathsf{polylog}(\mathsf{mpc}(f))$. In the other direction, suppose that there is a complexity c reduction from MKW_f to MKW_F. Then there are n, t with $t + \log n = c$ so that

$\textsc{mkw}_f$ is t-communication reducible to $\textsc{mkw}_{F_n}$. By Lemma 24, $F_n \circ h$ solves f for some depth-t circuit h. By monotonicity under solutions, and closure under reductions,

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

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 by letting

$$\mathsf{mpc}(f) := 2^c$$

for every monotone partial function f, where c is a the complexity of reducing $\textsc{mkw}_f$ to R_n. 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 24, $\textsc{mkw}_f$ has a t-bit reduction to $\textsc{mkw}_g$. As well, $\textsc{mkw}_g$ has a t' bit reduction to R_n where $t' + \log n$ is the complexity of reducing $\textsc{mkw}_g$ to R. Stringing these together, f has a $t + t'$ bit reduction to R_n, and so the complexity of reducing $\textsc{mkw}_f$ to R is $\leq t + t' + \log n = t + c$, and $\mathsf{mpc}(f) \leq 2^t \mathsf{mpc}(g)$. 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 22 such that R_N is equivalent to $\textsc{mkw}_{F_N}$. 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. As well, letting c be the complexity of reducing $\textsc{mkw}_{F_N}$ to R, we have $\mathsf{mpc}(F_N) = 2^c \leq 2^t = \mathsf{quasipoly}(N)$.

We will show that for each m, there is an $N' = \mathsf{quasipoly}(m)$ so that every partial function f with $\mathsf{mpc}(f) \leq m$ reduces to $F_{N'}$ via a $\mathsf{polylog}(m)$-depth reduction. Fix some f with $\mathsf{mpc}(f) \leq m$ and let $c = \log \mathsf{mpc}(f)$. Then by definition of mpc, $\textsc{mkw}_f$ reduces to some R_n in t bits of communication, where $t + \log n = c$; in particular, t is at most M and $\log n \leq c$. Then by paddability, we can reduce this to some $R_{N'}$ where $N' = \mathsf{quasipoly}(n) \leq \mathsf{quasipoly}(c)$ is a fixed function of m, and the further communication is at most $\mathsf{polylog}(c)$. Then by Lemma 24, f has a $\mathsf{polylog}(c)$-depth circuit reduction to $F_{N'}$ as desired. Thus, mpc is closed under reductions and admits a complete family. □

4.3 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 $\mathsf{mc}(f) \leq m$, there is a $\log m$-depth monotone circuit computing a function h so that $F_m \circ h$ solves f, and $\mathsf{mc}(F_m) \leq \mathsf{poly}(m)$.

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

Lemma 25. 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 26. *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 25. To prove the lemma, we will first assume mc is closed under reductions, e.g., $\mathsf{mc}(f \circ h) \leq \mathsf{poly}(\mathsf{mc}(f), 2^d)$ when h is computable in depth d. Let F be a partial function, and let f be a total function of minimal complexity solving F. Then $f \circ h$ solves $F \circ h$, so $\mathsf{mpc}(F \circ h) \leq \mathsf{mc}(f \circ h) \leq \mathsf{poly}(\mathsf{mc}(f), 2^d) = \mathsf{poly}(\mathsf{mpc}(F), 2^d)$. Conversely, since $\mathsf{mpc}(f) = \mathsf{mc}(f)$ for total functions, it follows immediately that if mpc is closed under 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. □

4.4 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 27 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 $g(x)$, and $\mathsf{mpc}(F_m) \leq \mathsf{quasipoly}(m)$.

Observe that such a universal family F_m can be viewed as complete under depth 0 reductions.

Theorem 27. *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 25, we can derive an analogous statement to Corollary 26 for total functions as well. Next, we state Theorem 3 formally, which follows immediately from Theorem 27 and Theorem 23.

Theorem 28. *Let* mpc *be a complexity measure. Then there is a paddable sequence of* TFNP^{cc} *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 27. If there is a universal family F_m for mpc then we can let $G_m = F_m$ since 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. □

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.)

5. Our characterization of the Polynomial calculus looks superficially similar to $\mathsf{PLS}^{\mathsf{PPA}}$: PLS provides the dag-like structure, while each inference is a matching (a PPA instance). Can one prove a separation between $\mathbb{F}_2$-PC and $\mathsf{PLS}^{\mathsf{PPA}}$? If so, what is the proof system corresponding to $\mathsf{PLS}^{\mathsf{PPA}}$?

Acknowledgements

Noah Fleming was supported by NSERC. Russell Impagliazzo was supported by NSF CCF 2212135 and the Simons Foundation. The authors thank Robert Robere and William Pires for comments on an earlier version of this paper.

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.

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

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

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

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

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

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

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

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

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

[12] Ben Davis and Robert Robere. Colourful TFNP and propositional proofs. In Amnon Ta-Shma, editor, *38th Computational Complexity Conference, CCC 2023, July 17-20, 2023, Warwick, UK*, volume 264 of *LIPIcs*, pages 36:1–36:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

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

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] Noah Fleming, Stefan Grosser, Toniann Pitassi, and Robert Robere. Black-box PPP is not turing-closed. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1405–1414. ACM, 2024.

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

[21] 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.

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

[23] 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.

[24] 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.

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

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

[27] 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.

[28] 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.

[29] 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.

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

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

[32] 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.

[33] Pavel Hubácek, Erfan Khaniki, and Neil Thapen. TFNP intersections through the lens of feasible disjunction. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPIcs*, pages 63:1–63:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

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

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

[36] 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.

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

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

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

[40] 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.

[41] Yuhao Li, William Pires, and Robert Robere. Intersection classes in TFNP and proof complexity. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPIcs*, pages 74:1–74:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[42] 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.

[43] 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.

[44] 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.

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

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

[47] 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.

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

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

[50] 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.

A Bounded-Width Resolution Proves its Reflection Principle

In this section we verify Theorem 10 by showing that resolution can indeed prove its own reflection principle, when measuring this complexity parameter. This corresponds to showing that $\mathsf{polylog}(n)$-width resolution can prove its own reflection principle. A *resolution proof* of an unsatisfiable set of Clauses F is a derivation of the empty clause $\perp$, which contains no literals, by the following inference rule:

– *Resolution rule.* From $C \vee x$ and $C \vee \overline{x}$ derive C.

The *size* of a resolution proof Π is the number of applications of the resolution rule, while the *width* is the maximum number of literals in any clause in the proof. The *complexity* of proving F in resolution is

$$\mathsf{Res}(F) := \min_{\Pi} \log \mathsf{size}(\Pi) + \mathsf{width}(\Pi).$$

Theorem 29. *There is a reflection principle* $\mathsf{Ref}^{\mathsf{Res}}$ *for resolution such that* $\mathsf{Res}(\mathsf{Ref}^{\mathsf{Res}}) \leq \mathsf{polylog}(n)$.

We define $\mathsf{Ref}^{\mathsf{Res}}$ by giving a natural verification procedure for resolution proofs.

A Verification Procedure for Resolution. The verification procedure $V^{\mathsf{Res}}(H, \Pi) := V^{\mathsf{Res}}_{n_H, m_H, c}(H, \Pi)$, encoding a proof of size $s = 2^c$ and width $w = c$, is defined by the following variables and constraints.

Variables: V^{Res} describes the proof Π with the following variables. Each line $\ell \in [s]$ is given by a length-w vector $V^{(\ell)}$ where $V^{(\ell)}_i \in [2n_H + 1]$ indicates the i^{th} literal in line ℓ. The values $1, \ldots n_H$ enumerate the positive variables, while $n_H + 1, \ldots 2n_H$ enumerate the negative literals. Finally, the value $2n_H + 1$ denotes the absence of this literal from ℓ.

The line 1 will correspond to the root of the proof ($\perp$) and the lines $s - m_H + 1, \ldots, s + m_H$ will correspond to the clauses of H. For each $\ell \in [s - m_H]$, we have two predecessor pointers $p^{(\ell)}_1, p^{(\ell)}_2 \in [s]$, naming the two lines from which ℓ was derived by the resolution rule.

Constraints: The constraints of V^{Res} are the following:

– Final Line. There are constraints saying that the final line 1 contains no variables: $V^{(1)}_i = 2n_H + 1$ for all $i \in [w]$.

– Axioms. For each $i \in [m_H]$ line $\ell = s - i + 1$, we have constraints stating that the ℓ^{th} line is exactly the i^{th} clause of H. In particular, for each $j \in [w]$, the variable $C_{i,j}$ of the formula Sat equals $V^{(i)}_j$.

– Inferences. For each line $\ell \in [s - m_H]$ we have constraints saying that $p^{(\ell)}_1, p^{(\ell)}_2 > \ell$, and a set of constraints saying that ℓ must be correctly derived from $p^{(\ell)}_1, p^{(\ell)}_2$. In particular, we have constraints stating that $V^{(p^{(\ell)}_1)}$ and $V^{(p^{(\ell)}_2)}$ contain complimentary literals and that $V^{(\ell)}$ contains exactly the literals of $V^{(p^{(\ell)}_1)}$ and $V^{(p^{(\ell)}_2)}$ after cancelling these complimentary literals. To see that this can be expressed as a not-too-large CNF formula, observe that for each value of $p^{(\ell)}_1, p^{(\ell)}_2$, these constraints can be defined by a decision tree querying $p^{(\ell)}_1, p^{(\ell)}_2, V^{(p^{(\ell)}_1)}, V^{(p^{(\ell)}_2)}$, which involve $t = O(\log(s + m_H) + w \log n_H)$-many variables. Hence can be described by a CNF formula of width t and $s^2 2^t$-many clauses, which is quasi-polynomial in the number of variables.

In order to prove Theorem 29 we will use the following characterization of resolution due to [9].

Fact 30. $\textsc{Search}_F \in \mathsf{PLS}^{dt}$ *iff* $\mathsf{Res}(F) = \mathsf{polylog}(n)$.

A complete problem for the class PLS is $\textsc{Iter}$, which states that every dag has a sink.

Iteration. An instance of $\textsc{Iter}_n$ is given by a *successor* function $s : [n] \to [n]$; we think of S as defining a directed graph on the vertices $[n]$, where there is an edge (u, v) if $S(u) = v$. A vertex $v \in [n]$ is a solution if:

- *Source Violation.* If $v = 1$ and $S(1) = 1$.
- *Sink Violation.* If $S(v) \neq v$ and $S(S(v)) = S(v)$.
- *Direction Violation.* If $S(v) < v$.

Hence, it suffices to give a reduction from $\mathsf{Ref}^{\mathsf{Res}}$ to $\textsc{Iter}$.

Proof of Theorem 29. We give a reduction from $\mathsf{Ref}^{\mathsf{Res}} := V^{\mathsf{Res}} \wedge \mathsf{Sat}$ to an instance of $\textsc{Iter}$ on s-many nodes. Let n be the total number of variables of $\mathsf{Ref}^{\mathsf{Res}}$ and let (H, Π, α) be an assignment to $\mathsf{Ref}^{\mathsf{Res}}$.

- Non-leaf Nodes. For each node $\ell \in [s - m_H]$, we define the successor $S(\ell)$ as follows: first, we check whether the clause encoded by $V^{(\ell)}$ is satisfied by the assignment α, querying the at-most-w variables mentioned in $V^{(\ell)}$. If it is, then we set $S(\ell) = \ell$. Otherwise, we check whether the inference constraints of ℓ are violated. This is done querying $p_1^{(\ell)}, p_2^{(\ell)}$ and $V^{(p_1^{(\ell)})}, V^{(p_2^{(\ell)})}$. If an inference constraint is violated then we set $S(\ell) = 1$ to force a direction violation. Otherwise, by soundness of the resolution rule, one of $p_1^{(\ell)}, p_2^{(\ell)}$ must be falsified. To determine which one, we query the bits of α corresponding to the literals specified by $V^{(p_1^{(\ell)})}, V^{(p_2^{(\ell)})}$. If $p_1^{(\ell)}$ is falsified by α, then we set $S(\ell) = p_1^{(\ell)}$, and otherwise we set $S(\ell) = p_2^{(\ell)}$.

  Observe that $s(\ell)$ can be defined by a $\mathsf{polylog}(n)$-depth decision tree. Indeed, the number of variables queried in this process is $O(\log s + w \log n_H)$.

- Root. To define $S(1)$, we also check that for all $i \in [w]$, $V_i^{(1)} = 2n_H + 1$. If this is not the case, then we set $S(1) = 1$ to force a source violation. Otherwise, we define $S(1)$ as in the non-leaf case.

- Axioms (Leaf Nodes). For $i \in [m_h]$, we define $S(s - i)$ by querying the variables of $V^{(s-i)}$ and the variables $C_{i,j}$ for all $j \in [w]$ encoding clause i of H in Sat. If these name different literals, then we set $S(s - i) = 1$ to force a direction violation. Otherwise, $S(s - i) = s - i$. As both $V_j^{(s-i)}$ and $C_{i,j}$ are encoded using $O(\log n_H)$-many variables, this can be done with a $\mathsf{polylog}(n)$-depth decision tree.

Observe that sink violations can only at the nodes v such that $S(v) = s - i$ for some $i \in [m_H]$.

It remains to define the output decision trees T_v^o for each solution $v \in [s]$. These decision trees will first query the decision tree for $S(v)$ and $S(S(v))$. If we see a direction violation or a source violation, then by definition of S, we must have observed that a Inference, Axiom, or Final Line constraint of $\mathsf{Ref}^{\mathsf{Res}}$ was violated and T_v^o outputs the index of that constraint. Otherwise, if we see a sink violation then $S(v) = s - i$ for some $i \in [m_H]$. At this point, we have queried the variables $C_i := \{C_{i,j} : j \in [w]\}$ and the bits of α corresponding to the variables in C_i. We claim that $C_i(\alpha) = 0$. Indeed, as $S(v) = s - i$, the clause encoded by $V^{(s-i)}$ must have be falsified by α, and as we did not violate any Final Line constraints, we know that $V^{(s-i)} = C_i$, and hence C_i is falsified by α. Thus T_v^o can output the index of the constraint of Sat stating that clause i must be falsified by α. □

B Characterization of the Unary Polynomial Calculus

In this appendix we prove Theorem 20, 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 31. *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* IndEnd-of-Line *on $O(sL)$ many variables.*

Proof. Fix a uPC proof Π of F and fix a topological ordering on the lines of Π with the first line being the 1 polynomial. For each monomial m that appears in Π, let c_m be the maximum a absolute value of any coefficient of m in Π. Let the number of nodes of our IndEOL instance be $N := \sum_m c_m$ and the number of pools be L. That is, each pool will have one node for each possible occurrence of m in the corresponding line.

The proof is identical to the proof of Lemma 15, except that we need to specify the direction of each matching. Fix any assignment $x \in \{0,1\}^n$ to F. The edges $E : [L] \to [L] \times [L]$ will be defined exactly as they were in the characterization of $\mathbb{F}_2$-PC:

- If v is an *axiom*, then $E(v)(x) = (v, v)$ always.
- If v was derived by *addition* from lines u, w in Π then $E(v)(x) = (u, w)$ always.
- If v was derived from u by *multiplication* by a variable x_i, then the decision tree $E(v)(x)$ queries x_i and outputs v if $x_i = 0$ and otherwise it outputs u. That is, v becomes a leaf if $x_i = 0$.

For every line $v \in [L]$ of Π, there will be a one-to-one correspondence between the monomials in v (thought of as a multi-set) that are non-zero under x and the nodes that appear in P_v, and hence we abuse notation and refer to nodes as "monomials". To ensure that this is the case, and to avoid consistency violations, for every pool $u \in [L]$ and every monomial $m \in [N]$ that does not occur in u, we fix $M^{(v)}(u, m) = (u, m, +)$. Note that the $+$ is arbitrary, it would have been equally valid to set it to$-$. If m occurs in line v, then query the at most d-many variables of x that define m. If $m(x) = 0$ then we set $M^{(v)}(u, m) = (u, m, +)$ (where again $+$ is arbitrary). Otherwise, if $m(x) \neq 0$, then we set $M^{(v)}(u, m)$ as follows:

- *Addition:* If v was derived by addition from lines u and w then if m is a b-monomial in v, for $b \in \{-, +\}$, then m came from either u or w. Suppose that it was u, then we set $M^{(v)}(v, m) = (u, m, \overline{b})$ where $\overline{b} = \{+, -\} \setminus b$, and $M^{(v)}(u, m) = (v, m, b)$.

  The remaining monomials m in u which have not yet had their matching defined must have cancelled with a monomial in w of the opposite polarity during the addition. If m is a b-monomial for $b \in \{-, +\}$, then we set $M^{(v)}(u, m) = (u, m, \overline{b}) = (w, m, b)$.

- *Multiplication:* If v is derived from u by multiplication by variable x_i, then the decision tree for each $M^{(v)}(z, m)$ for $z \in \{v, u\}$ and monomial m begins by querying x_i. If $x_i = 0$ then $M^{(v)}(z, m) = (z, m)$. Otherwise if $x_i = 1$, then for every monomial $x_i m$ in v, there is some m' equal to m or $x_i m$ in u which $x_i m$ comes from and we match $M^{(v)}(v, m) = (u, m', -)$ and $M^{(v)}(u, m') = (v, m x_i, +)$. Finally, we must be able to pair-off the monomials in u that have yet to be matched into pairs $(m, m x_i)$ into pairs which cancel after multiplication by x_i. That is, one monomial in the pair must be positive and the other negative. Suppose that $b \in \{-, +\}$ is the polarity of m. We match $M^{(v)}(u, m) = (u, m x_i, b)$ and $M^{(v)}(u, m x_i) = (u, m, \overline{b})$.

- *Axioms:* If v is an axiom in Π, then $v = \overline{C}$ for some $C \in F$. For each monomial $m \in \overline{C}$, $M^{(v)}$ queries the at-most-d bits of x corresponding to the variables in $\overline{C}$ to get an assignment α. If $\overline{C}(x) = \overline{C}(\alpha) = 1$ then $C(x) = 0$ and we have found a solution to $\textsc{Search}_F$, so we match $M^{(v)}(v, m)$ in some way that creates a violation; for example $M^{(v)}(v, m) = (1, 1, +)$.

Otherwise, if $\overline{C}(x) = 0$ then either $m(\alpha) = 0$, and we have set $M^{(v)}(v, m) = (v, m, +)$, or we $m(\alpha) \neq 0$. If the latter is the case then as $\overline{C}(x) = 0$, the monomials of $\overline{C}$ must cancel to 0. Hence, there must be another monomial m' in v such that $m \restriction \alpha = m' \restriction \alpha$, and such that if m is a b-monomial, for $b \in \{-, +\}$, then m' is a $\overline{b}$-monomial. Hence, we set $M^{(v)}(v, m) = (v, m', \overline{b})$ and $M^{(v)}(v, m) = (v, m, b)$.

Finally, observe that the only violations occur at the pools corresponding to the axioms. Therefore, the output decision trees are defined identically to the proof of Lemma 15. □

Lemma 32. *Let F be an unsatisfiable CNF formula. If* $\textsc{Search}_F$ *reduces to an instance of* INDEOL *on n variables using depth-d decision trees, then there is an degree-$O(d)$ and size $n^2 2^{O(d)}$* uPC *proof of F.*

Proof. Let $F = C_1 \wedge \ldots \wedge C_m$ be an unsatisfiable CNF formula such that $\textsc{Search}_F$ reduces to $\textsc{IndLeaf}_n$ using decision trees $\{T_i\}, \{T^o_j\}$ of depth at most d. Let L be the number of pools and N the number of nodes of the $\textsc{IndLeaf}_n$ instance. For each pool $v \in [L]$ consider the polynomial

$$P_v := \sum_{m \in [N]} \sum_{(u,m') \neq (v,m)} [\![M^{(v)}(v, m) = (u, m', -)]\!] - [\![M^{(v)}(v, m) = (u, m, +)]\!]$$

which records the difference between the number of positive monomials and negative monomials in line v. Using this reduction, we will prove F in uPC by deriving that $P_v = 0$ by induction from $v = L, \ldots 1$.

Claim 33. *For each $v \in [L]$, there is a degree $O(d)$, size $NL2^{O(d)}$* uPC *proof of the polynomial P_v from F.*

This suffices to complete the proof of the lemma, as once we have derived P_1, we can derive $P_1 - 1$ from the axioms of INDEOL in order to complete the proof.

Claim 34. *There is a degree $O(d)$, size $N2^{O(d)}$* uPC *derivation of the polynomial $P_1 - 1$ from F.*

Having derive P_1 and $P_1 - 1$, we add them together to obtain 1, completing the proof. □

We will now prove the outstanding claims, which rely on the following Key Observation. Let β be a solution to INDEOL (either a dag, root, matching violation, or consistency), the *indicator* of the solution β is the minimal conjunct I_β such that $I_\beta(x) = 0$ iff $x \restriction \mathsf{Vars}(\beta) = \beta$.

Key Observation: If Q is any conjunct such that there is a solution β with $I_\beta \subseteq Q$ then Q has a uPC proof of degree $O(d + \deg(Q))$ and size $O(|Q|2^d)$ from the axioms of F.

Proof. Let T^o_β be the output decision tree for the solution β. As $I_\beta \subseteq Q$, any assignment x which falsifies Q must witness the solution β. Let k the index of the clause of F output by $T^o_\beta(x)$. By the soundness of the reduction, $\overline{C}_k(x) = 0$. Hence, whenever $Q \cdot [\![T^o_\beta = k]\!]$ is falsified, $\overline{C}_k$ is falsified as well. Putting this together,

$$\begin{aligned} Q &= \sum_{k \in [m]} Q \cdot [\![T^o_\beta = k]\!] && \text{(Summing all the root-to-leaf paths in } T^o_\beta \text{ gives 1)} \\ &= \sum_{k \in [m]} \overline{C}_k = 0. && (\overline{C}_k \text{ is an axiom of } F) \end{aligned}$$

As T^o_β has depth at most d, it has at most 2^d-many leaves, and the bound on the degree and size follows. □

Proof of Claim 34. It suffices to express

$$P_1 - 1 := \sum_{m\in[N]} \sum_{(u,m')\neq(1,m)} [\![M^{(1)}(1,m) = (u,m',-)]\!] - [\![M^{(1)}(1,m) = (u,m',+)]\!] - 1$$

as a low-degree polynomial in which each term involves one of the axioms $\overline{C}_i$ of F. Any assignment to the variables of F which satisfies one of these terms is a root violation, and hence a solution to INDEOL. Since summing over all root-to-leaf paths in the decision tree $M^{(1)}(1,1)$ gives 1, we have

$$\begin{aligned}
0 &= \sum_{u\in[L]} \sum_{m\in[N]} \sum_{b\in\{+,-\}} [\![M^{(1)}(1,1) = (u,m,b)]\!] - 1 \\
&= \sum_{(u,m)\neq(1,1)} \sum_{b\in\{+,-\}} [\![M^{(1)}(1,1) = (u,m,b)]\!] + \sum_{b\in\{+,-\}} [\![M^{(1)}(1,1) = (1,1,b)]\!] - 1 \\
&= \sum_{(u,m)\neq(1,1)} \sum_{b\in\{+,-\}} [\![M^{(1)}(1,1) = (u,m,b)]\!] + 0 - 1 && \text{(Key Observation)} \\
&= \sum_{(u,m)\neq(1,1)} [\![M^{(1)}(1,1) = (u,m,-)]\!] + \sum_{(u,m)\neq(1,1)} [\![M^{(1)}(1,1) = (u,m,+)]\!] - 1 \\
&= \sum_{(u,m)\neq(1,1)} [\![M^{(1)}(1,1) = (u,m,-)]\!] - 1 && \text{(Key Observation)} \\
&= \sum_{(u,m)\neq(1,1)} [\![M^{(1)}(1,1) = (u,m,-)]\!] - \sum_{m'\in[N]} \sum_{(u,m)\neq(1,m')} [\![M^{(1)}(m',u) = (u,m,+)]\!] - 1 \\
& && \text{(Key Observation)} \\
&= \sum_{m\in[N]} \sum_{(u,m')\neq(1,m)} [\![M^{(1)}(1,m) = (u,m',-)]\!] - [\![M^{(1)}(1,m) = (u,m',+)]\!] - 1 && \text{(Key Observation)} \\
&= P_1 - 1.
\end{aligned}$$

This polynomial is the sum over $O(N)$-many pairs of decision trees (from the Key Observation), each of depth d and therefore at most 2^d-many leaves. Hence, this is a proof of degree $O(d)$ and size at most $N2^{O(d)}$. □

Proof of Claim 33. Extend the definition of P_u to the matching for v, as

$$P_u^{(v)} := \sum_{m\in[N]} \sum_{(w,m')\neq(u,m)} [\![M^{(v)}(u,m) = (w,m',-)]\!] - [\![M^{(v)}(u,m) = (w,m,+)]\!]$$

To facilitate a proof by induction, we will show that for every $v \in [L]$, the following two polynomials have a size $N^2 2^{O(d)}$ and degree $4d$ proof from the axioms of F:

$$P_v - \sum_{u,w>v} [\![E(v) = (u,w)]\!] \left(P_u^{(v)} + P_w^{(v)}\right) \tag{4}$$

$$\sum_{u,w>v} [\![E(v) = (u,w)]\!] \left(P_u^{(v)} - P_u\right) \tag{5}$$

(4) expresses the correctness of the matching, while the (5) expresses that consistency is maintained — the signs of monomials in a pool u do not switch between matchings.

Supposing that we can derive these polynomials efficiently from F, we complete the proof of the claim by proving that P_v induction on $v = L, \dots, 1$. For the base case, when $v = L$, (4) is simply P_L as there does not exist any $u, w > L$.

Now, suppose that we have derived P_u for all $u > v$. Multiplying these by $E(v)$ and summing gives

$$\begin{aligned}
&\sum_{u,w>v} [\![E(v) = (u,w)]\!]\Big(P_u + P_w\Big) \\
=&\sum_{u,w>v} [\![E(v) = (u,w)]\!]\left(P_u^{(v)} + P_w^{(v)}\right) && \text{(Adding (5))} \\
=&P_v. && \text{(Adding (4))}
\end{aligned}$$

As for every $u, w \in [L]$, (4) and (5) have degree $4d$ and $N2^{O(d)}$-size proofs from F, the total degree of the proof is $O(d)$ and the size is at most $NL2^{O(d)}$. It remains to show that (4) and (5) have efficient proofs from F.

Equation 4. This polynomial states that the monomials in u and w cancel to give the monomials in v. To derive (4), we will use the matching $M^{(v)}$. The intuition is that if a monomial m in this matching is correctly matched, then it is matched to $-m$ and so they are identically 0, and we can derive them as such. If m is incorrectly matched, then this is a matching violation, which the output decision tree maps to a clause of F, and hence we can derive it from this clause.

$$\begin{aligned}
(4) &= P_v\Big(\sum_{u,w\in[L]} [\![E(v) = (u,w)]\!]\Big) - \sum_{u,w>v} [\![E(v) = (u,w)]\!]\Big(P_u^{(v)} + P_w^{(v)}\Big) && \text{(As } \textstyle\sum_{p\in E(v)} p = 1\text{)} \\
&= P_v\Big(\sum_{u<v\vee w<v} [\![E(v) = (u,w)]\!] + \sum_{u,w>v} [\![E(v) = (u,w)]\!]\Big) - \sum_{u,w>v} [\![E(v) = (u,w)]\!]\Big(P_u^{(v)} + P_w^{(v)}\Big)] \\
&= P_v\Big(\sum_{u,w>v} [\![E(v) = (u,w)]\!]\Big) - \sum_{u,w>v} [\![E(v) = (u,w)]\!]\Big(P_u^{(v)} + P_w^{(v)}\Big) && \text{(Key Observation)} \\
&= \sum_{u,w>v} [\![E(v) = (u,w)]\!]\Big(P_v - P_u^{(v)} - P_w^{(v)}\Big) \\
&= \sum_{u,w>v} [\![E(v) = (u,w)]\!] \sum_{z\in\{u,v,w\}} b_z P_z^{(v)} && (6)
\end{aligned}$$

where $b_z = 1$ if $z = v$ and $b_z = -1$ otherwise. It remains to deduce (6) from the axioms of F. To do so, we use the matching $M^{(v)}$. Define the polynomial

$$\mathsf{match}^v := \sum_{u,w>v} [\![E(v) = (u,w)]\!] \sum_{z\in\{u,v,w\}} b_z \sum_{m\in[N]} [\![M^{(v)}(z,m) \neq (z,m,*)]\!] \cdot \Big(\mathsf{match}^v_{z,m,-} - \mathsf{match}^v_{z,m,+}\Big).$$

where, for $\gamma \in \{-,+\}$,

$$\mathsf{match}^v_{z,m,\gamma} := \sum_{a\in[L]} \sum_{\alpha\in[N]} [\![M^{(v)}(z,m) = (a,\alpha,b)]\!] \underbrace{\sum_{z^*\in[L]} \sum_{m^*\in[N]} \sum_{b^*\in\{-,+\}} [\![M^{(v)}(a,\alpha) = (z^*,m^*,b^*)]\!]}_{\equiv 1},$$

where the final part is equal to 1 as it is the sum over all paths in the decision tree for $M^{(v)}(a,\alpha)$, and so (6) $=$ match^v. To show that match^v has a proof from F, we break it into two multisets of conjuncts, V

which contains the conjuncts which witness *violations*, and C which correspond to the *correct* matchings — those without violations. The conjuncts in V can be removed by the Key Observation — they can be derived from the clauses of F — while the conjuncts in C are correctly paired, meaning that each positive occurence is paired with a negative occurence, and hence they sum to 0.

$$\begin{aligned}
\mathsf{match}^v &= \sum_{t\in C} t + \sum_{t\in V} t \\
&= \sum_{t\in C} t + 0 && \text{(Key Observation)} \\
&= \sum_{u,w>v} [\![E(v)=(u,w)]\!]\Bigg(\sum_{z\in\{u,v,w\}} \sum_{m\in[N]} [\![M^{(v)}(z,m)\neq(z,m,*)]\!]\cdot \\
&\quad \sum_{a\in[L]} \sum_{\alpha\in[N]} \Big([\![M^{(v)}(z,m)=(a,\alpha,-)]\!][\![M^{(v)}(a,\alpha)=(z,m,+)]\!] \\
&\quad - [\![M^{(v)}(z,m)=(a,\alpha,+)]\!][\![M^{(v)}(a,\alpha)=(z,m,-)]\!]\Big)\Bigg) \\
&= 0.
\end{aligned}$$

Finally, observe that each $\mathsf{match}^v_{z,m,\gamma}$ is the sum over two decision trees, each of depth at most d and hence size at most 2^d. match^v is formed from $\mathsf{match}^v_{z,m,\gamma}$ by querying $E(v)$ and taking a sum over all monomials $m\in[N]$, and hence has degree at most $3d$ and size $N2^{O(d)}$. Finally, the application of the Key Observation introduces one additional decision tree, and so the proof of (4) has size $N^2 2^{O(d)}$ and degree at most $4d$.

Equation 5. This polynomial states that if u is a child of v then the monomials in P_u occur with the same polarity in the matching $M^{(v)}$. For $z\in\{u,v\}$ and $\gamma\in\{-,+\}$, let $\overline{z}=\{u,v\}\setminus z$, and define

$$\mathsf{pol}^{(z)}_\gamma := \sum_{m\in[N]} \sum_{a\in[L],\alpha\in[N]} [\![M^{(z)}(u,m)=(a,\alpha,\gamma)]\!] \underbrace{\sum_{\substack{w\in[L],\beta\in[N]\\ b\in\{-,+\}}} [\![M^{(\overline{z})}(u,m)=(w,\beta,b)]\!]}_{\equiv 1},$$

where the final sum is equal to 1 as it is the sum over all paths in a decision tree. Therefore, we can write

$$(5) = \sum_{u,w>v} [\![E(v)=(u,w)]\!]\Big(P^{(v)}_u - P_u\Big) = \sum_{u,w>v} [\![E(v)=(u,w)]\!]\Big(\mathsf{pol}^{(v)}_- - \mathsf{pol}^{(v)}_+ - \mathsf{pol}^{(u)}_- + \mathsf{pol}^{(u)}_+\Big) \tag{7}$$

We break each $\mathsf{pol}_\gamma^{(z)}$ into a multi-set of conjuncts $V_\gamma^{(z)}$ containing the conjuncts which witness violations — these will all be consistency violations — and $C_\gamma^{(z)}$ which do not witness violations. Then (7) equals

$$\sum_{u,w>v} [\![E(v)=(u,w)]\!]\Big(\sum_{t\in C_-^{(v)}} t-\sum_{t\in C_+^{(v)}} t-\sum_{t\in C_-^{(u)}} t+\sum_{t\in C_+^{(u)}} t+\sum_{t\in V_-^{(v)}} t-\sum_{t\in V_+^{(v)}} t-\sum_{t\in V_-^{(u)}} t+\sum_{t\in V_+^{(u)}} t\Big)$$

$$=\sum_{u,w>v} [\![E(v)=(u,w)]\!]\Big(\sum_{t\in C_-^{(v)}} t-\sum_{t\in C_+^{(v)}} t-\sum_{t\in C_-^{(u)}} t+\sum_{t\in C_+^{(u)}} t\Big)+0 \qquad \text{(Key Observation)}$$

$$=\sum_{u,w>v} [\![E(v)=(u,w)]\!]\Big(\Big(\sum_{t\in C_-^{(v)}} t-\sum_{t\in C_+^{(u)}} t\Big)+\Big(\sum_{t\in C_-^{(u)}} t-\sum_{t\in C_+^{(v)}} t\Big)\Big)$$

$=0$.

We justify the final equality: as we have removed all of the consistency violations, for any monomial $m\in P_u$, if it occurs at the head of an arrow in $M^{(u)}$ then it must occur at the tail of an arrow in $M^{(v)}$, and vice-versa. Hence for every term $t=[\![M^{(z)}(u,m)=(a,\alpha,-)]\!][\![M^{(\overline{z})}(u,m)=(w,\beta,+)]\!]$ in $C_-^{(v)}$, there is a term $t'=[\![M^{(\overline{z})}(u,m)=(w,\beta,+)]\!][\![M^{(z)}(u,m)=(a,\alpha,-)]\!]$ in $C_+^{(u)}$, and vice-versa. Thus, these terms sum to 0.

Finally, observe that each pol is the sum over all monomials $m\in[N]$, and for each m we have two decision trees, each of depth at most d. Thus, pol has size at most $N2^{O(d)}$ and degree at most $2d$. Then, (5) is obtained by querying the decision tree for $E(v)$. Finally, the Key Observation introduces one additional decision tree, and hence the proof of (5) has size $N2^{O(d)}$ and degree at most $4d$. □