Abstract

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

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

- Every well-behaved proof system which can prove its own soundness (a reflection principle) is characterized by a TFNP problem. Conversely, every TFNP problem gives rise to a well-behaved proof system which proves its own soundness.
- Every well-behaved monotone circuit model which admits a universal family of functions is characterized by a TFNP problem. Conversely, every TFNP problem gives rise to a well-behaved monotone circuit model with a universal problem.

As an example, we provide a TFNP characterization of the Polynomial Calculus, answering a question from [25], and show that it can prove its own soundness.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Proof Complexity, Circuit Complexity, TFNP

Digital Object Identifier 10.4230/LIPIcs.ITCS.2023.65

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

Introduction

In recent years, connections between proof systems and monotone circuit models have revolutionized the areas of proof and circuit complexity, allowing for the tools from one area to be applied to problems from the other. These connections take the form of
– Interpolation Theorems, which translate small proofs into efficient computations in an associated model of monotone circuit [6, 16, 17, 19, 30, 34–36, 41, 43, 45].
– Query-to-Communication Lifting Theorems, which translate efficient monotone computations into small proofs in an associated proof system [10, 14, 15, 21, 27–29, 33, 37, 39, 40, 44, 47].

Recently, the landscape of total functional NP (TFNP) has emerged as an organizing principle for connections between proof systems and models of monotone circuits [12, 26]. For many of the proof systems which admit an interpolation theorem or lifting theorem there is a TFNP problem which characterizes it in the following sense: the set of TFNP problems which are reducible to this problem, via query-efficient reductions, is equivalent to the set of tautologies that can be efficiently proven in the system. This has resulted in proof complexity becoming a major tool for proving separations in black-box TFNP. Conversely, the novel perspective offered by TFNP has provided a number unique results for proof complexity, such as complete tautologies for certain proof systems, as well as striking intersection theorems [25].

An analogous phenomenon has emerged for monotone circuit complexity. For many monotone circuit models, the set of functions which can be computed efficiently is equivalent to the set of problems that can be reduced to a certain TFNP problem using communication-efficient reductions. When these TFNP problems collide — that is, when there is both a proof and circuit characterization of a particular TFNP problem — then we immediately obtain an interpolation theorem between this proof system and circuit model [46]! Moreover, many of the query-to-communication lifting theorems can be viewed as constructing a query-efficient reduction to a particular TFNP problem out of a communication-efficient reduction to that problem. This is exciting as it suggests understanding when TFNP problems admit such characterizations as a pathway for developing further connections between proof complexity and circuit complexity.

In this paper we give exact conditions under which a proof system or monotone circuit model admits a characterization by a TFNP problem. For proof complexity, we show that every well-behaved\(^1\) proof system which can prove its own soundness (a reflection principle) is characterized by a TFNP problem — simply the search problem associated with its reflection principle. This gives a recipe for constructing a TFNP problem which characterizes a given proof system, simply write down the search problem for a reflection principle corresponding to that proof system! Conversely, every TFNP problem gives rise to a well-behaved proof system which proves its own soundness and which is closed under decision tree reductions. Furthermore, this result is constructive: for every TFNP problem we give a proof system which it characterizes. As an example, we provide a TFNP characterization of the Polynomial Calculus, answering a question from [25], and show that it can prove its own soundness. For circuit complexity, we show that every well-behaved model of monotone circuit which admits a universal family of functions is characterized by a natural TFNP problem. Conversely, every TFNP problem gives rise to a well-behaved monotone circuit model with a universal problem.

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

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

\(^1\) We will say that a proof system of monotone circuit model is well-behaved if it satisfies some minor technical conditions discussed in Subsection 1.2.
Black-Box TFNP and Proof Complexity.

Beginning with [3], proof complexity has become a major tool for proving black-box TFNP separations. In fact, black-box TFNP — denoted TFNP$^{dt}$ — can be viewed as the study of the false clause search problem. Every TFNP$^{dt}$ problem is equivalent to $S_F$ for some unsatisfiable CNF formula $F$. Using this connection, Göös et al. [26] observed that many prominent TFNP$^{dt}$ problems are characterized by associated proof systems in the sense that the CNF formulas $F$ that are efficiently provable in that proof system are exactly the problems $S_F$ that are reducible to the TFNP$^{dt}$ problem.

This has led to the characterization of many well-studied TFNP$^{dt}$ subclasses:

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

That is, these proof systems are characterized by complete problems for these classes, and therefore an unsatisfiable formula $F$ can be efficiently proven in one of these proof systems iff $S_F$ lies in the corresponding class. Thus, separations between these proof systems translate into separations between their corresponding TFNP$^{dt}$ subclasses. This has resulted in a complete picture of how the most prominent TFNP$^{dt}$ subclasses relate [2,7,25,26].

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

- **Complete Problems:** Any proof system which is characterized by a TFNP$^{dt}$ problem $S_F$ has $F$ as its complete problem, in the sense that it has short proofs of exactly the formulas $F'$ for which $S_{F'}$ can be efficiently reduced to $S_F$. [26]
Intersection Theorems: Proof systems which can efficiently prove a formula iff that formula has short proofs in several other proof systems [25].

Coefficient Separations: Separations between the complexity of certain algebraic proof systems when their coefficients are represented in unary versus binary [25].

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

Communication TFNP and Monotone Circuit Complexity.

Karchmer and Wigderson [32] showed that the monotone formula complexity of any monotone function $f$ is equal to the communication complexity of $mKW_f$. Building on this, Razborov [45] considered reductions between black-box TFNP classes where one measures the amount of communication needed to perform the reduction (for some suitable partition of the input), denoted TFNP\textsuperscript{cc}, and showed that PLS\textsuperscript{cc}-complete problems characterize monotone circuit complexity. There is good reason for this; analogous to how TFNP\textsuperscript{dt} is the study of the false clause search problem, TFNP\textsuperscript{cc} can be viewed as the study of the monotone Karchmer-Wigderson game. Indeed, every $R \in$ TFNP\textsuperscript{cc} is equivalent to $mKW_f$ (over the same partition of the variables) for some associated monotone function $f$ [20, 26].

Following these results, a number of TFNP\textsuperscript{cc} problems have been characterized by models of monotone circuits [17, 26]. However, there remain many important circuit models for which no TFNP\textsuperscript{cc}-characterization is known.

A Theory of Interpolation and Lifting Theorems.

As we have just discussed, certain proof systems are characterized by TFNP\textsuperscript{dt} problems, while certain models of monotone circuits are characterized by problems in TFNP\textsuperscript{cc}. Göös et al. [26] observed that in all-known examples of TFNP problems which admit both a characterization by a proof system and a monotone circuit, there exists both an interpolation theorems and query-to-communication lifting theorem between that proof system and monotone circuit. This is to be expected, as a key component of both interpolation and query-to-communication lifting theorems proceeds by relating $S_F$ to $mKW_f$ for associated pairs $(F, f)$. In fact, it is not difficult to see that whenever a TFNP class admits a characterization by both a proof system and a monotone circuit model then there is an interpolation theorem between this proof system and circuit model — this follows by the simple observation that communication protocols can simulate decision trees [46]! Thus, the landscape of TFNP, together with characterizations of TFNP problems by proofs and circuits, appears to provide a roadmap for potential interpolation and query-to-communication lifting theorems.

1.2 Our Results

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

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

ii) Proves its own soundness: $P$ can prove that its proofs are sound. That is, $P$ has small proofs of a reflection principle about itself, encoded in an efficiently-verifiable manner.
Conversely, we show that every $\text{TFNP}^{dt}$ problem has a proof system which characterizes it. Furthermore, this proof system satisfies both conditions (i) and (ii). Our first main results can be informally stated as follows.

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

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

By writing down an efficiently verifiable reflection principle for a proof system, this provides a somewhat systematic way of generating a $\text{TFNP}^{dt}$ problem which characterizes that proof system.

As an example, we define a new $\text{TFNP}$ subclass called IND-PPA, which contains problems which can be solved by inductive *inductive* parity arguments. We show that the IND-PPA-complete problem characterizes the $F_2$-Polynomial Calculus proof system, and furthermore that the $F_2$-Polynomial Calculus can prove its own soundness.

**Theorem 2 (Informal).** IND-PPA$^{dt} = F_2$-PC. As well, $F_2$-PC has small proofs of an efficiently verifiable reflection principle about itself.

As a bonus, we show that the technique that we use to generate the $\text{TFNP}^{dt}$ problem which characterizes the $F_2$-Polynomial Calculus can readily be applied in order to generate $\text{TFNP}^{dt}$ problems which characterize all of the dynamic variants of static proof systems for which $\text{TFNP}^{dt}$ are known.

In Subsection 2.4, we provide $\text{TFNP}^{dt}$ problems for $F_q$-Polynomial Calculus, unary Polynomial Calculus, and unary dag-like Sherali-Adams.

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

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

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

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

Finally, we investigate whether this characterization can be extended from partial function complexity measures to *total function* measures. Since complexity measures on total functions induce measures on partial functions, this allows us to give a general condition under which a complexity measure on total functions has a $\text{TFNP}^{cc}$ characterization (Theorem 17) by applying Theorem 3.

**A Note on the Provability of Reflection Principles.**

Theorem 1 establishes that the property of $P$ having short proofs of a reflection principle about itself is closely related to having a $\text{TFNP}^{dt}$ 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$^d$ classes characterizable in terms of second-order theories of bounded arithmetic, see [5].

Analogous results were obtained for even stronger propositional proof systems by [23]. On the other
hand, Garlik [22] showed that resolution requires exponential length for refutations of (a particular
“leveled” version of) its reflection principle, and Atserias-Müller [1] gave exponential lower bounds
on resolution refutations of a relativized reflection principle.

Theorem 1 requires that the proof system $P$ has short proofs of a variant of a reflection principle
about itself. There are two main differences between our encodings and previous ones in the literature.
The first is that the reflection principle is parameterized by a complexity parameter $c$ (see Section 2)
rather than the typical size parameter. The second is that the reflection principle must be efficiently
verifiable, meaning that an error in the purported $P$-proof in the reflection principle can always be
verified by examining in a small number of bits. Thus, for example, the bound of Garlik [22] does not
contradict our results.

2 Proof Complexity and Black-Box TFNP

We begin by defining black-box TFNP. A total search problem is a sequence of relations $R_{n} \subseteq
\{0, 1\}^n \times \mathcal{O}_n$, one for each $n \in \mathbb{N}$ which is total — for each $x \in \{0, 1\}^n$ there is $i \in \mathcal{O}$ such that
$(x, i) \in R_{n}$. A total search problem is in TFNP$^d$ its solutions are verifiable: for each $i \in \mathcal{O}$ there
there is a decision tree $T^o_i$ of polylog$(n)$ depth such that
\[
T^o_i(x) = 1 \iff (x, i) \in R_n.
\]

Decision Tree Reductions. A decision tree reduction from $Q \subseteq \{0, 1\}^s \times \mathcal{O}'$ to $R \subseteq \{0, 1\}^n \times \mathcal{O}$ is a
set of decision trees $T_i : \{0, 1\}^s \to \{0, 1\}$ for $i \in [n]$ and $T^o_j : \{0, 1\}^s \to \mathcal{O}'$ for $j \in \mathcal{O}$ such that for
any $x \in \{0, 1\}^s$,
\[
(T_1(x), \ldots, T_n(x), j) \in R \iff (x, T^o_j(x)) \in Q.
\]

That is, the $T_i$’s map inputs to from $Q$ to $R$, and the $T^o_j$’s map solutions to $R$ back to solutions to $Q$.
The depth of the reduction is $d$, the maximum depth of any of the decision trees involved, and the size
is $n$. The complexity of the reduction is $\log n + d$ and the complexity of reducing $Q$ to $R$, denoted
$R^d(Q)$, is the minimum complexity of any decision tree reduction from $Q$ to $R$. The TFNP$^d$ subclass
associated with $R$, denoted $R^d$, is the set of all $Q \in$ TFNP$^d$ such that $R^d(Q) = \text{polylog}(n)$.

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

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

Proof. As $R \in$ TFNP$^d$ there are polylog$(n)$-depth decision trees $\{T_i\}_{i \in \mathcal{O}}$ which verify $R$. Define
a canonical CNF formula associated with $R$ to be
\[
F := \bigwedge_{i \in \mathcal{O}} \neg T^o_i,
\]
where we have abused notation and associated $T^o_i$ with the DNF obtained by taking a disjunction over
the (conjunction of the literals along) the accepting paths in $T^o_i$. This makes a $\neg T^o_i$ a CNF formula
expressing that $T^o_i$ outputs 0. It is not difficult to check that a solution to $S_F$ is equivalent to a solution
to $R$. 

The upshot is that black-box TFNP is exactly the study of the false clause search problem! Thus, it suffices to study the search problems for the canonical CNF formulas $S_F$ associated with $\mathcal{R} \in \text{TFNP}^{dt}$ instead of $\mathcal{R}$ itself. Furthermore, note that this is robust as for any pair of decision trees $\{T_o\}$ and $\{T'_o\}$ that verify the same $\mathcal{R} \in \text{TFNP}^{dt}$, the resulting false clause search problems $S_F$ and $S'_F$ are polylog$(n)$-reducible.

Using this connection, Göös et al. [26] observed that many important proof systems are characterized by associated TFNP$^{dt}$ problems in the sense that the CNF formulas $F$ that are efficiently provable in that proof system are exactly the problems $S_F$ that are efficiently reducible to that TFNP$^{dt}$ problem.

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

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

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

We say that a TFNP$^{dt}$ problem $\mathcal{R}$ characterizes a proof system $P$ if $\mathcal{R}^{dt} = \{S_F : P(F) = \text{polylog}(n)\}$; this is reflexive and so we also say that $P$ characterizes $\mathcal{R}$. In fact, many of these characterizations hold in the following stronger sense: let $P$ be any of the proof systems listed above, and $\mathcal{R}$ be the canonical complete problem for its corresponding TFNP$^{dt}$ class, then for any unsatisfiable CNF formula $F$,

$$P(F) = \Theta(R^{dt}(S_F)).$$

In this section we give necessary and sufficient conditions for such a characterization to occur. The first condition is that the proof system proves an efficiently verifiable variant of a reflection principle.

### What is a Reflection Principle?

The second condition of Theorem 1 is that the proof system must be able to prove its own soundness. A reflection principle $\text{Ref}_P$ for a proof system $P$ states that $P$-proofs are sound; it says that if $\Pi$ 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 $\Pi$, and a truth assignment $\alpha$ to $H$. The formula (falsely) asserts that $\Pi$ is a $P$-proof of $H$ and $\alpha$ satisfies $H$,

$$\text{Proof}_P(H, \Pi) \land \text{Sat}(H, \alpha).$$

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

For a proof system $P$, there are many ways to encode its proofs, with the choice of the encoding potentially affecting the complexity of proving the associated reflection principle. Rather than worrying about the particular encoding, we will instead define one reflection principle for each
efficiently verifiable way of encoding P-proofs, which we call a verification procedure. Recall that the complexity c of a proof is always an upper bound on the width of the CNF being proven. For this reason, and to simplify notation, we will bound the width of the CNF H by c.

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

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

As the complexity parameter c bounds the logarithm of the size of the proof, and by the third point, the number of variables is exponential in \(\Theta(c)\), the second condition ensures that \(V_{n, m, c}\) has width \(\text{poly}(c)\) and can be verified by looking at polynomial-in-c many variables. The third condition can be relaxed, and larger numbers of variables can be tolerated at the cost of worse bounds in Theorem 6.

We give a concrete example of a verification procedure for the Polynomial Calculus proof system in Section 2.3.

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

We can now define a reflection principle for any proof system based on a verification procedure.

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

\[
\text{Proof}_{n_H, m_H, c}(H, \Pi) \land \text{Sat}_{n_H, m_H, c}(H, \alpha),
\]

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

\[
\begin{aligned}
\forall i \in [m_H], \exists j \in [c] \bigg[ &\left(\left(C_{i,j} = x_k\right) \land \alpha_k\right) \lor \left(\left(C_{i,j} = \neg x_k\right) \land \neg \alpha_k\right)\bigg],
\end{aligned}
\]

where \(p = \ell\) is the indicator function of \(p\) being equal to \(\ell\). This can be encoded as a CNF formula of width \(O(c \log n_H)\) and size \(m_H \exp(O(c \log n_H))\).

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

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

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

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

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

\begin{itemize}
  \item \textbf{Theorem 1.} The following hold:
  \begin{enumerate}
    \item For any TFNP\textsuperscript{dt} problem \(R\) there is a proof system \(P\) such that \(R\) characterizes \(P\). Furthermore, \(P\) is closed under decision tree reductions and there is a reflection principle \(\text{Ref}_P\) for \(P\) such that \(P(\text{Ref}_P) \leq \text{polylog}(n)\).
    \item For any proof system \(P\) which is closed under decision tree reductions and for which there is a reflection principle \(\text{Ref}_P\) of which \(P\) has polylog\((n)\)-complexity proofs, there is a TFNP\textsuperscript{dt} problem \(R\) which characterizes \(P\).
  \end{enumerate}
\end{itemize}

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

2.1 A Proof System for any TFNP Problem

We begin by describing how any TFNP\textsuperscript{dt} problem \(R\) can be transformed into a proof system for refuting unsatisfiable CNF formulas of polylog width. The key observation is that because each TFNP\textsuperscript{dt} problem is equivalent to the search problem for some unsatisfiable CNF formula, we can view decision tree reductions between TFNP\textsuperscript{dt} problems as proofs in a proof system — indeed, these reductions are sound and efficiently verifiable! More formally, a proof \(P\) in this proof system, of the (unsatisfiability) of a CNF formula \(H\), will consist of a low-depth decision reduction from \(S_H\) to an instance of the false clause search problem \(S_F\) for the unsatisfiable formula \(F\) associated with the TFNP problem \(R\). For this, we first define a notion of reduction between CNF formulas.

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

\[C(T) := \bigwedge\{\neg p : p\ is\ a\ rejecting\ path\ of\ T\},\]
where a path $p$ is identified with the conjunction of the literals set true along the path, and $\neg p$ is its negation.

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

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

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

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

Canonical Proof System. Let $S_F \in \text{TFNP}^d$. The canonical proof system $P_F$ for $S_F$ proves an unsatisfiable CNF formula $H$ on $n_H$ variables if $H$ is reducible to an instance of $F$ on some $n$ variables. A $P_F$-proof $\Pi$ consists of the decision trees $T = \{T_i\}_{i \in [m]}$ and $T^o = \{T_i^o\}_{i \in [m]}$ of the reduction. The size of $\Pi$ is the number of variables $n$ of the instance of $F$, and the depth is the maximum depth among the decision trees. The complexity of proving an unsatisfiable CNF formula $H$ is then the minimum over all $P$-proofs of $H$,

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

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

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

Theorem 4. Let $S_F \in \text{TFNP}^d$ and $H$ be an unsatisfiable CNF formula. Then,

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

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

In particular, $S_F^d(S_H) = \Theta(P_F(H))$.

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

\end{document}
To prove (a), let \( n, T_1, \ldots, T_n, \tilde{T}_1, \ldots, \tilde{T}_m \) be a \( P_F \) proof of \( H \) of size \( s \) and depth \( d \). We claim that this is also a reduction from \( S_{\bar{H}} \) and \( S_F \). Indeed, fix any assignment \( x \) such that \( T_1, \ldots, T_n(x) \) falsifies clause \( C_i \) of \( F \) and the decision tree \( T_i^s(x) \) follows some path \( p \). Then, a clause of the formula \( C_i(T) \lor \neg p \) is falsified under \( x \), and furthermore that clause is a weakening of the \( T_i^s(x) \)-th clause of \( H \). Thus, \( (x, T_i^s(x)) \in S_H \). This reduction has depth \( d \) and size \( n = O(s) \).

Canonical Proof Systems Prove their own Soundness

In this section we define a natural formulation of the reflection principle for the proof system \( P_F \) for any TFNP\(^H\) problem \( S_F \) by way of defining a verification procedure for \( P_F \). We show that the canonical proof system can prove this encoding of the reflection principle. To encode proofs \( \Pi \) in the canonical proof system — which are decision tree reductions — we require the notion of a generic of a decision tree, which is a template for decision trees of depth at most \( d \) — any decision tree of depth at most \( d \) (over a set of variables \( \alpha_1, \ldots, \alpha_n \) and output set \( O \)) can be recovered from an assignment to the variables of a generic decision tree.

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

Recall that in the reflection principle \( \text{Proof}(H, \Pi) \) states that the proof \( \Pi \) (which we will encode using generic decision trees) is indeed a proof of \( H \). To state \( \text{Proof}(H, \Pi) \), it will be helpful to have the following definition. The decision tree simulating a generic decision tree \( \tilde{T} \) is obtained from \( \tilde{T} \) as follows: Replace each internal vertex \( v \) of \( \tilde{T} \) by a complete binary tree querying the variables of \( x_v \), and at each leaf where \( x_v = i \), queries \( \alpha_i \). The leaves \( l \) of the generic decision tree are replaced with complete binary trees querying \( x_l \) in which each leaf where \( x_l = j \) is labelled by the output \( j \in O \).

Verification Procedure for \( P_F \). Let \( S_F \in \text{TFNP}^H \). We define a verification procedure \( \text{V}_{n_H, m_H, (d, n_F)}^{P_F} \) for \( P_F \), which encodes a complexity \( c = (d + \log n_F) \)-proof \( \Pi \) of a CNF formula \( H \) on \( n_H \) variables and \( m_H \) clauses as follows. \( \Pi \) is specified by \( n_F \) depth-\( d \) generic decision trees \( \tilde{T}_1, \ldots, \tilde{T}_{n_F} \) with output in \( \{0, 1\} \) and \( m_F \) depth-\( d \) generic decision trees \( \tilde{T}_1^0, \ldots, \tilde{T}_m^0 \) with output in \( \{m_H\} \). The constraints of \( \text{Proof} \) enforce that each clause of the reduced CNF formula \( F_H \) is a weakening of a clause of \( H \). For each \( i \in [n_F] \), let \( S_i \) be the decision tree simulating \( \tilde{T}_i \) but eliminating the queries to the variables \( \alpha_i \).\(^2\) Recall that the assignment of truth values to the vector of variables \( x_v \) at a vertex \( v \) determines the index \( i \in [n_H] \) of the variable being queried at \( v \) in the decision tree. Let \( z_k \in [n_F] \) denote the \( k \)-th variable of \( F \).

We will construct the constraints of \( \text{Proof} \) from the following decision trees \( T_{C_i} \), for each clause \( C_i \) in \( F \): First, it runs the decision trees \( S_k \) for every \( k \in [n_F] \) such that \( C_k \) involves \( z_k \); this determines the literals which occur in one of the clauses of \( F_H \), namely in one of the clauses that is formed by applying the decision trees \( \tilde{T}_i \) to the clause \( C_i \). We temporarily use \( C' \) to denote this clause of \( F_H \). Note that \( C' \) involves variables of \( H \); however, the truth values (the \( \alpha_i \) values) of the

\(^2\) Proof\(_{n_H, m_H, (d, n_F)}^{P_F}(H, \Pi) \) does not involve the variables \( \alpha_i \).
variables in \( C' \) have not been queried and are instead treated in the next phase as being set to the values that falsify \( C' \). Second, it runs the decision tree simulating \( \tilde{T}_i \). A vertex of \( \tilde{T} \) labelled with an \( x_v \) is handled by querying the variables \( x_v \). The results of the queries to \( x_v \) specify a variable \( \alpha_v \). The variable \( \alpha_v \) may appear in \( C' \) and if so is treated as having the value that falsifies \( C' \). If, however, the variable \( \alpha_v \) does not appear in \( C' \), then it is non-deterministically queried; that is, the tree \( T_C \) branches to try both 0 and 1 as truth values for \( \alpha_v \). The result of running the decision tree simulating \( \tilde{T}_i \) is a value \( \ell \) specifying a clause of \( H \). Third, it queries the vector of variables \( C_{i,j} \) for \( j \in [c] \); this determines the literals of the \( \ell \)-th clause of \( H \). If a path in this decision tree determines that the clause \( C' \) of \( F_H \) is not a weakening of the \( \ell \)-th clause of \( H \), then the path is called “bad”.

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

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

**Canonical Reflection Principle.** Let \( S_F \in \text{TFNP}^{d_1} \). We define its canonical reflection principle \( \text{Ref} \) to be the conjunction

\[
\text{Proof}_{n_H,m_H,(d,n_F)}(H,\Pi) \land \text{Sat}_{n_H,m_H,(d,n_F)}(H,\alpha),
\]

where \( \text{Sat} \) is defined as in the definition of the reflection principle and \( \text{Proof} := V^P_{n_H,m_H,(d,n_F)} \). 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 \( \text{Ref} \) can be evaluated by looking at the values of \( \text{polylog}(n) \) many variables, where \( n \) is number of variables of \( \text{Ref} \). Thus, \( S_{\text{Ref}} \in \text{TFNP}^{d_1} \).

**Theorem 5.** For any \( S_F \in \text{TFNP}^{d_1} \), \( P_F(\text{Ref} \lor \text{Ref}) \leq \text{polylog}(n) \).

**Proof.** Fix an instance of \( S_{\text{Ref}} \). By Theorem 4, it suffices to show that \( S_{\text{Ref}} \) is reducible to an instance of \( S_F \). Let the instance of \( S_F \) be specified with parameters \( (n_H,m_H,(d,n_F)) \), letting \( c = d + \log n_F \). For each generic decision tree \( \tilde{T}_i \) of \( S_F \), let \( S_i \) be the decision tree that simulates it. As well, let \( S_o \) be the decision tree that simulates \( T_o \).

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

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

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

In this section we identify the necessary conditions for a proof system to be characterized by a TFNP^{dt} problem. The first necessary condition is that the proof system must be closed under decision-tree reductions, as TFNP^{dt} is closed under these reductions. That is, it must admit short proofs of a reflection principle about itself. As we will show, any verification procedure for its proofs will do.

Theorem 6. Let $P$ be a proof system that is closed under decision tree reductions, let $V$ be a verification procedure for $P$, and denote $Ref_{P,V}$ by $Ref$. For any unsatisfiable CNF formula $H$, the following hold.

1. $S_{Ref}^{dt}(S_H) \in O(P(H))$.
2. $P(H) \in O(S_{Ref}^{dt}(S_H)P(Ref))$.

In particular, if $P$ has polylog$(n)$-complexity proofs of Ref then $P$ is characterized by $S_{Ref}$.

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

Before proving this theorem we will give a high-level sketch of the proof for the case of polylog$(n)$-complexity reductions. Let $P$ be any proof system that is closed under decision tree reductions. Observe that $S_{Ref} \in$ TFNP^{dt} as Ref is efficiently verifiable. Consider any $S_H \in$ TFNP^{dt} such that $S_{Ref}^{dt}(S_H) = \text{polylog}(n)$ ($S_H$ reduces to $S_{Ref}$ with polylog-depth decision trees). Then, as $P$ is closed under decision tree reductions and there is an $O(\text{polylog}(n))$-complexity $P$-proof of Ref$_P$, there must also be an efficient $P$-proof of $H$. Conversely, suppose that $II$ is a $\text{polylog}(n)$-complexity $P$-proof of an unsatisfiable CNF formula $H$. We can construct a reduction from $S_H$ to $S_{Ref}$ by hard-wiring $H$ and $II$ into $S_{Ref}$, leaving the only truth assignment variables free. On any input $\alpha$ to the variables of $H$, the hard-wired instance of $S_{Ref}$ must output a falsified clause of $H$ as $II$ is a valid $P$-proof of $H$.

Proof of Theorem 6. We will begin by proving (ii). Let $H$ be any unsatisfiable CNF formula and recall that $S_{Ref}^{dt}(S_H)$ denotes the complexity of reducing $S_H$ to $S_{Ref}$. As $P$ is closed under decision tree reductions, there is a $P$-proof of $H$ with complexity $P(H) = O(S_{Ref}^{dt}(S_H)P(Ref))$.

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

2.3 Example: The Polynomial Calculus

As an example, we give a characterization of the Polynomial Calculus by a natural TFNP^{dt} problem and show that it can prove a reflection principle about itself, establishing Theorem 2. This answers an open question from [25], asking for a characterization of the Polynomial Calculus. To define our characterization of the $F_2$-Polynomial Calculus, we will leverage the characterization of its static variant, $F_2$ Nullstellensatz, by PPA-complete problems [26]. PPA is the class of TFNP problems.
The Polynomial Calculus (PC). The \(\mathbb{F}_2\)-Polynomial Calculus proves that an unsatisfiable system of \(\mathbb{F}_2\)-polynomial equations \(\{p_i(x) = 0\}_{i \in [n]}\) has no solutions over \(\{0, 1\}\). An unsatisfiable CNF formula \(F = C_1 \land \ldots \land C_m\) is encoded as such a system of equations by mapping each clause to an equation \(C_i\) such that \(C_i(x) = 1\) iff \(C_i(x) = 0\) (for example, \((x_1 \lor x_2 \lor x_3)\) represented as \((1 + x_1)x_2(1 + x_3) = 0\)). Note that the degree of \(C_i\) is equal to the width of \(C_i\). We will operate exclusively with multilinear arithmetic; that is, \(x_i^2\) and \(x_i\) are represented by the same function. Formally, we operate modulo the ideal \((x_1^2 = x_1)_{i \in [n]}\).

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

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

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

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

Next, we define IND-PPA, the subclass of TFNP problems which are reducible to the IND-PPA-complete problem \(\text{IND-LEAF}\), which will characterize \(\mathbb{F}_2\)-PC. At a high level this is the \(\text{LEAF}\) problem defined over a directed acyclic graph (dag). An instance of this problem is given by a set set of \(N\) nodes (corresponding to monomials) and a set of \(L\) pools (corresponding to lines in the proof).

The pools are arranged in a dag: each pool \(\ell \in [L]\) has a set of immediate predecessors described by variables \(\mathcal{P}_{\ell}^{(i)} \in \{0, 1\}\) for \(\ell' < \ell\). Each pool \(\ell\) is associated with a set of nodes \(A^{(\ell)} \subseteq [N]\) and we hard-code that the root pool \(1\) has \(A^{(1)} = \{1\}\) for some distinguished 1-node. We have an instance of \(\text{LEAF}\) defined over the nodes of this dag as follows: for each pool \(\ell\) we have a matching \(M^{(\ell)}\) between the nodes of \(\ell\) and the nodes of its predecessors; see Figure 1. Since the \(L\)-th pool contains only a single node, there must be some pool with an unmatched node. A solution is an unmatched or mismatched node.

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

**Induction** PPA. The IND-PPA-complete problem \(\text{IND-LEAF}\) is defined as follows

- **Structure.** \([L]\) pools and \([N]\) nodes. We think of each \(\ell \in [L]\) as being associated with its own copy of \([N]\).
- **Variables.** For each \(\ell \in [L]\) and \(\ell' < \ell\) we have \(\mathcal{P}_{\ell}^{(i)} \in \{0, 1\}\) indicating whether \(\ell'\) is an immediate predecessor of pool \(\ell\). For each pool \(\ell \in [L]\) and node \(m \in [N]\), we have a variable \(A_{m}^{(\ell)} \in \{0, 1\}\) indicating whether node \(m\) is active at pool \(\ell\). Finally, we have a matching between the nodes of
\( \ell \in [\ell] \) and the nodes of all of its predecessors: For each \( \ell' < \ell \) and \( m \in [N] \) there is a variable \( M_{\ell',m'} \in [\ell] \times [N] \) indicating where \( \ell' \)'s copy of node \( m' \) is matched in the matching for pool \( \ell \).

The root pool \( L \) has \( A_1^{(L)} = 1 \) and \( A_m^{(L)} = 0 \) for all \( m \neq 1 \).

- **Solutions.** Since the root has an odd number of active nodes, and each matching is even, there must be some pool \( \ell \in [L] \) with an erroneous matching. A solution is any triple \( (\ell, \ell', m) \in [L]^2 \times [N] \) such that \( \ell' \) is a predecessor of \( \ell \) and \( m \) is an active node for \( \ell' \), and \( m \) is matched to some node of some pool \( \ell'' \) which is not matched to \( m \). That is, \( P_{\ell'}^{(l)} = 1 \), \( A_m^{(l)} = 1 \), \( M_{\ell',m}^{(l)} = (\ell'', m') \), and either \( P_{\ell''}^{(l')} = 0 \), \( A_{m'}^{(l')} = 0 \), or \( M_{\ell'',m'}^{(l')} \neq (\ell', m) \).

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

**Theorem 7.** For any unsatisfiable CNF formula \( F \),

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

In particular, IND-LEAF\(^{dt}(S_F) = \Theta(F_{2-PC}(F)) \).

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

We remark that an analogous statement holds for the \( F_{2-PCR} \) proof system, which builds on \( F_{2-PC} \) to include additional “dual” variables \( \pi_i \) for each \( i \in [n] \) to represent \( \neg x_i \), along with the additional axioms \( x_i + \pi_i = 0 \). Indeed, this is only a change to the encoding of the CNF formula \( F \) as a set of polynomials and does not affect the resulting TFNP\(^{dt} \) problem. Note that this does not
contradict the separation between PC and PCR due to de Rezende et al. [13], as their separation is in terms of size, while this characterization is in terms of the complexity measure.  

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

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

**Proof.** Let $F$ be an unsatisfiable CNF formula and suppose that $S_F$ is reducible to an instance of IND-LEAF on $n$ variables using decision trees of depth at most $d$. That is, each variable $x$ of the IND-LEAF instance is computed by a depth-$d$ decision tree $T_x$ querying variables of $F$; for simplicity, we will abuse notation and associate each variable $x$ with the polynomial formed by taking the sum over the (product of the literals on each of the) accepting paths of $T_x$ (those labelled 1). As well, let $\{T^i_x\}_i$ be the decision trees for each solution $i$ of the IND-LEAF instance.

For $\ell \in L$ let

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

over $\mathbb{F}_2$. We will derive by induction on $\ell = 1, \ldots, L$ that $q_\ell = 0$. Roughly, this polynomial states that there is a perfect matching between the nodes in $\ell$ and the nodes in its predecessors. This will be sufficient to complete the proof as $A^{(L)}_m = 1$ and $A^{(L)}_m = 0$ for all $m \neq 1$, and so the decision trees for these variables are identically 1 and 0 respectively. Thus, we will have derived 0 = $\sum_{m \in [N]} A^{(L)}_m = A^{(L)}_1 = 1$.

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

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

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

from the axioms.

This claim is sufficient to complete the proof. Indeed, we can use it in order to derive $q_\ell = 0$ from $q_{\ell'} = 0$ for $\ell' < \ell$ (which we have derived by induction) without significantly increasing the degree.

To see this, multiply each $q_{\ell'}$ by $P^{(\ell)}_{\ell'}$ and sum them to obtain

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

Adding this polynomial to $\sum_{\ell' \leq \ell} P^{(\ell)}_{\ell'} A^{(\ell')}_{m} = 0$, which has a low-degree proof from $F$ by the previous claim, gives $p_{\ell} = 0$. Note that since every $p_{\ell'}$ is a degree-$d$ polynomial, each of these

---

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

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

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

$$\text{match}^{(m, ℓ')} := \sum_{m' ∈ [N], ℓ' ∈ [ℓ]} [M^{(m, ℓ')}_m = (m^*, ℓ^*)] \sum_{α, β ∈ \{0, 1\}} [P^{(ℓ')}_{ℓ, α} = α] [A^{(m^*)}_{m, ℓ'} = β] \sum_{m^*, ℓ'} [M^{(m, ℓ')}_m = (m^*, ℓ')] \cdot \text{match}^{(m, ℓ)}(m'_m, ℓ)' \cdot N \]$$

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

$$\text{match}^{(ℓ)} := \sum_{ℓ' ∈ [ℓ]} P^{(ℓ')}_{ℓ, α} \sum_{m ∈ [N]} A^{(m^*)}_{m, ℓ'} \cdot \text{match}^{(m, ℓ')}$$

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

We will partition the terms of match$_{(ℓ)}$ into two sets, where $C$ is the set of terms where the copy of $m$ belonging to $ℓ'$ is correctly matched — that is, for all $ℓ' ≤ ℓ$ and $m ∈ [N]$ with $P^{(ℓ')}_{ℓ, α} = 1$ and $A^{(m^*)}_{m, ℓ'} = 1$, $m$ is matched to a node $m^* ∈ [N]$ belonging to a pool $ℓ^*$ to $ℓ'$ and the remaining terms correspond to erroneous matchings. Observe that each term in $C$ occurs an even number of times, as $(m, ℓ')$ is matched to $(m^*, ℓ^*)$ iff $(m^*, ℓ^*)$ is matched to $(m, ℓ')$. Thus, summing over the terms in $C$ gives $\sum_{t ∈ C} t = 0$.

Consider a term $t ∈ E$. This term corresponds to a node $m$ in some pool $ℓ'$ being incorrect matched: let $s$ be this incorrect matching and we will denote by $t_s$ that $t$ witnesses the incorrect matching $s$. Let $T^s_s$ be the decision tree for solution $s$ and abuse notation by identifying it with the polynomial obtained by summing over (the product of the literals on) each of its paths. Recalling that the result of summing over all paths in a decision tree is 1, we have

$$\text{match}^{(ℓ)} = \sum_{ℓ' ∈ C} t^* + \sum_{t_s ∈ E} t_s = 0 + \sum_{t_s ∈ E} t_s \cdot T^s_s$$

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

$$\sum_{ℓ' ≤ ℓ} \sum_{m ∈ [N]} A^{(m^*)}_{m, ℓ'} = \sum_{ℓ' ∈ [ℓ]} \sum_{m ∈ [N]} A^{(m^*)}_{m, ℓ'} \cdot \text{match}^{(m, ℓ')} = \text{match}^{(ℓ)} = \sum_{t_s ∈ E} t_s \cdot T^s_s = 0$$
Lemma 9. Let \( F \) be an unsatisfiable CNF formula on \( n \) variables. If there is a \( \mathbb{F}_2 \)-Polynomial Calculus proof of \( F \) with size \( s \), length-\( L \), and degree-\( d \) then \( S_F \) is reducible by decision trees of depth \( O(d) \) to an instance of IND-LEAF on \( O(sL) \) variables.

A representation of this construction is given in Figure 2.

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

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

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

Fix some \( \ell \in [L] \) and for each monomial \( m \in [N] \) occurring in line \( \ell \) define \( A_m^{(\ell)} \) to be the decision tree which outputs 1 iff \( m(x) = 1 \). For the remaining monomials \( m \), set \( A_m^{(\ell)} = 0 \). Next, we set the predecessor variables as follows. If \( \ell \) was derived by addition from \( \ell' \), \( \ell'' \), then set \( P_{\ell}^{(\ell')} = P_{\ell}^{(\ell'')} = 1 \) and \( P_{\ell}^{(\ell^*)} = 0 \) for all other \( \ell^* \in [L] \). Otherwise, if \( \ell \) was derived by multiplication by a variable \( x_i \) from \( \ell' \), then we set \( P_{\ell}^{(\ell')} = x_i \) and \( P_{\ell}^{(\ell^*)} = 0 \) for all \( \ell^* \neq \ell' \). Finally, if \( \ell \) was an initial clause of \( F \) then we set \( P_{\ell}^{(\ell^*)} = 0 \) for all \( \ell^* \).

Next, we set the matching variables of each \( \ell \) which does not correspond to an initial clause of \( F \) as follows. Observe that if \( \ell \) was derived by addition from \( \ell' \), \( \ell'' \) then every monomial \( m \) in \( \ell \) must occur in exactly one of \( \ell', \ell'' \) as otherwise it would have cancelled over \( \mathbb{F}_2 \). Thus, if \( \ell' \) is the child of \( \ell \) in which \( m \) also occurs, then we set \( M_{\ell,m}^{(\ell)} = (\ell, m) \) and \( M_{\ell,m}^{(\ell')} = (\ell', m) \), matching those two occurrences of the \( m \)-th node. Otherwise, if \( m \) does not appear in \( \ell \), but is in one of the predecessors of \( \ell \), say \( \ell' \), then it must also appear in \( \ell'' \). In this case we set \( M_{\ell,m}^{(\ell')} = (\ell'', m) \) and \( M_{\ell,m}^{(\ell'')} = (\ell', m) \). Finally if \( m \) does not occur in any of these lines, then we set \( M_{\ell,m}^{(\ell)} \) arbitrarily for \( \ell^* \in \{\ell, \ell', \ell''\} \).
Otherwise, if \( \ell \) was derived from \( \ell' \) by multiplication with some variable \( x_i \) then we set the matching in a similar way as above. A monomial \( m \) occurs in \( \ell \) if either \( m \) or \( m \setminus x_i \) occurs in \( \ell' \), but not both. For each \( m \in [N] \), if \( m \) occurs in \( \ell \) then we set \( M^{(\ell)}_{\ell,m} \), match it to the \( m \) or \( m \setminus x_i \) that occurs in \( \ell' \), and set the matching variable for this node to match it back to \( (\ell, m) \). Otherwise, if \( m \) and \( m \setminus x_i \) occur in \( \ell' \) then set \( M^{(\ell)}_{\ell',m} = (\ell', m \setminus x_i) \) and \( M^{(\ell)}_{\ell',m \setminus x_i} = (\ell', m) \). Finally, for match the \( m \) which do not occur in \( \ell \) or \( \ell' \) arbitrarily.

Lastly, we set the matching variables of the \( \ell \in L \) which correspond to an axiom \( A \in \{ \overline{C} : C \in F \} \). Each \( M^{(\ell)}_{\ell,m} \) is defined by querying the variables in \( A \) (of which there are at most \( d \) by definition).

If \( A \) is satisfied, then we fix an arbitrary matching between the monomials of \( A \), and otherwise if \( A \) is falsified then we fix an arbitrary false matching (say, match each of the monomials in \( A \) in a cycle).

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

The Polynomial Calculus Proves its own Soundness

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

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

\[
\frac{l_1 \quad l_2}{l_1 x + l_2 y}
\]

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

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

The constraints are as follows:

- Initial Clauses. We enforce that the first \( m_H \) lines are active, that the monomials of \( \ell \in [m_H] \) are exactly the monomials of the \( \ell \)-th clause of \( H \), and that each active monomial is matched with another active monomial in \( \ell \).
– **Root.** To require that this is indeed a proof of \( H \), we enforce that the root \( L \) of the proof has
\[
a^{(L)}_1 = 1, \text{ mon}^{(L)}_{i,j} = n_H + 1 \text{ (i.e., is the constant 1 polynomial) for all } i \in [d], \text{ and } a^{(L)}_{m} = 0 \text{ for all } m \neq 1.
\]

– **Inference.** To express the inference rule, we will require that if line \( \ell > m_H \) was derived from lines
\[
p_1^{(\ell)}, p_2^{(\ell)} \text{ with variables } v_1^{(\ell)}, v_2^{(\ell)}, \text{ then the monomials of } \ell \text{ are exactly those in } v_1^{(\ell)} p_1^{(\ell)} + v_2^{(\ell)} p_2^{(\ell)}
\]
after cancelling \( \mod 2 \). More concretely, that each active monomial in \( \ell \) is matched to the active monomial in \( p_1^{(\ell)} \) or \( p_2^{(\ell)} \) from which it was derived.

Define \( \text{Ref}^{\text{PC}} := \text{Sat} \land \text{Proof}^{\text{PC}} \) where \( \text{Proof}^{\text{PC}} := V^{\text{PC}} \). We show that \( F_2\text{-PC} \) has efficient proofs of \( \text{Ref}^{\text{PC}} \).

\[\textbf{Theorem 10.} \quad \text{PC(Ref}^{\text{PC}}) \leq \text{polylog}(n).\]

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

First, we define the decision trees for the variables of \( \text{IND-LEAF} \). For each \( \ell \in [L] \) and \( \ell' < \ell \), we define its predecessor variables \( P^{(\ell)}_{\ell'} \) by querying \( p_1^{(\ell)} \) and \( p_2^{(\ell)} \) and outputting 1 if either of these is \( \ell' \), and 0 otherwise.

We define the activity \( A^{(\ell)}_m \) of the \( m \)-th node of \( \ell \) by querying whether \( a^{(\ell)} = 1 \), then querying the \( d \log n_H \) bits of \( \text{mon}^{(\ell)}_m \), and then checking that \( a_\ell = 1 \) for all \( i \in \text{Vars}(\text{mon}^{(\ell)}_m) \) (the variables in monomial \( m \)). \( A^{(\ell)}_m = 1 \) if all of these checks pass, and 0 otherwise.

Finally, the matching variables \( M^{(\ell)}_{\ell',m} \) are defined as follows. If \( \ell' \neq \ell \) we query \( p_1^{(\ell')} \) and \( p_2^{(\ell')} \) to determine if \( \ell' \) is one of the children of \( \ell \). If it is not then the output of \( M^{(\ell)}_{\ell',m} \) can be arbitrary. Otherwise, if \( \ell' = \ell \) then we can continue. We query \( v_1^{(\ell')} \) to determine the variable \( y \) that was used to derive monomial \( m' \), and we query \( \text{match}^{(\ell')}_{v_1^{(\ell')},m} \) to obtain a pair \( j \in \{0, 1, 2\} \times [s] \) and \( m^* \in [s] \) indicating to which child of \( \ell \) and which monomial \( m^* \) the monomial \( m \) is matched. As well, we query \( \text{match}^{(\ell')}_{v_1^{(\ell')},m^*} \) to ensure that this matching is consistent. Finally, query \( \text{mon}^{(\ell)}_m \) and \( \text{mon}^{(\ell')}_{m^*} \),

where \( p_0^{(\ell)} := \ell \). If the variables occurring in \( m \) are not the the same as those in \( v_1^{(\ell')} m^* \), then let \( M^{(\ell')}_{\ell',m} \) be some arbitrary \( (\ell', m) \) such that \( \ell \neq p_1^{(\ell')}, p_2^{(\ell')} \). In particular, this means that \( a^{(\ell')} = 0 \) and this will cause a violation (solution). Otherwise, set \( M^{(\ell')}_{\ell',m} = (p_1^{(\ell')}, m^*) \).

Next, we define the output decision trees for the solutions of this instance of \( \text{IND-LEAF} \). Let \((\ell, \ell', m) \) be a solution, we create a decision tree mapping this solution back to a falsified clause of \( \text{Ref}^{\text{PC}} \) as follows. If \( \ell \) is one of the initial clauses \( C_\ell \) of \( H \), i.e., \( \ell \leq m_H \), then we query whether \( C_\ell(\alpha) = 0 \), and if so we output the index of the falsified constraint of SAT which states that the \( \ell \)-th clause of \( H \) is satisfied under \( \alpha \). Otherwise, this decision tree queries the decision tree for \( M^{(\ell')}_{\ell',m} \).

If we discover that \( m \) is matched to a monomial \( m^* \) with \( m \neq v_1^{(\ell')} m^* \), or if \( m \) is matched to a monomial \( m^* \) but that monomial is not matched to \( m \), then we output the index of the clause of \( \text{Ref}^{\text{PC}} \) which states that this should not happen.

This completes the description of the reduction. Each of the decision trees involved queries at most \( \text{polylog}(n) \) many variables and thus by **Theorem 7** it follows that there is a \( \text{polylog}(n) \)-complexity \( F_2\text{-PC} \) proof of \( \text{Ref}^{\text{PC}} \). ❄️
2.4 Characterizing Dynamic Variants of Static Systems

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

\( \mathbb{F}_q \)-Polynomial Calculus.

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

Unary Polynomial Calculus.

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

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

\[4\] Typically, the Polynomial Calculus is defined with a multiplication rule rather than addition, where one may derive \( \alpha p + \beta q \) from previously derived polynomials \( p, q \) and \( \alpha, \beta \in \mathbb{Z} \). However, as we are measuring coefficients in unary, multiplication by positive coefficients may be simulated by repeated addition. To simulate the use of negative coefficients, we push the negations to the leaves of the proof and include both \( \overline{C}_i = 0 \) and \(-\overline{C}_i = 0 \) as axioms.
$m$ belongs to a pool $\ell$ then if it occurs at the head of an arrow in $M^{(\ell^*)}$ for $\ell^* > \ell$ then it is considered negative and if it as the tail then it is positive. This change in meaning depending on whether this is the matching for the pool $\ell$ to which it belongs versus a parent pool should be thought of as the sign of monomials propagating from the children $\ell$ to the parent $\ell^*$ in the matching $M^{(\ell^*)}$.

### 2.4.0.1 Induction PPAD.

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

- **Structure.** $[L]$ pools and $[N]$ nodes. Each $\ell \in [L]$ will correspond to a line in the polynomial calculus proof and be associated with its own copy of $[N]$.

- **Variables.** For each $\ell \in [L]$ and $\ell' < \ell$ we will have a predecessor variable $P_{\ell'}^{(\ell)} \in \{0, 1\}$ indicating whether $\ell'$ is a predecessor of $\ell$. For each pool $\ell \in [L]$ and each node $m \in [N]$ we have a variable $A_m^{(\ell)} \in \{0, 1\}$ indicating whether node $m$ is active in pool $\ell$. There is a distinguished node $1 \in [N]$ and we hardcode that $A_1^{(\ell)} = 1$ and $A_0^{(\ell)} = 0$ for all $m \neq 1$. Finally, we have a directed matching between the nodes in pools $\ell' \leq \ell$, defined by variables $M_{\ell', m}^{(\ell)} \in \{-, +\} \times [L] \times [M]$ indicating to which node and pool $\ell'$'s copy of $m$ is matched in a directed fashion, and whether it appears at the head (or tail) of the arrow.

- **Solutions.** IND-PPAD will state the following: (i) For each pool $\ell$ with no predecessors, $M^{(\ell)}$ enforces that there is a matching between the nodes of pool $\ell$. (ii) If $\ell' < \ell$ is a predecessor of pool $\ell$ then either every active node of $m$ of $\ell$ occurs at the opposite end of an arrow in the matching $M^{(\ell')}$ for $\ell$ than in matching for $M^{(\ell)}$ (e.g., $m$ occurs at the tail of an edge in $M^{(\ell)}$ and the head of an edge in $M^{(\ell')}$, or every active node $m$ of $\ell$ occurs at the same end of an arrow in $M^{(\ell')}$ as in $M^{(\ell)}$. (iii) The root pool $L$ contains only a distinguished 1-node. Observe that (i) – (iii) cannot hold simultaneously, and thus IND-PPAD is total. Formally, the solution of IND-PPAD are as follows:

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

- **Polarity Solutions.** A tuple $(\ell, \ell', m) \in [L]^2 \times [N]^2$ which violates (ii). That is, $A_m^{(\ell')} = 1, P_{\ell'}^{(\ell)} = 1, M_{\ell', m}^{(\ell')} = (\alpha, *, *)$ and $M_{\ell', m}^{(\ell)} = (\alpha, *, *)$.

A portion of an instance of IND-END-OF-LINE is depicted in Figure 3.
**Figure 3** Part of an IND-END-OF-LINE instance. The yellow circles indicate the active nodes of each pool; for example $A_4^1 = A_4^3 = A_4^4 = 1$ and $A_4^m = 0$ for all other $m$. The pink area indicates the predecessors of pool 4; $P_1^4 = P_2^4 = 1$. The solid arrows indicate the matching $M_4^4$ for pool 4, while the dashed arrows indicate that matchings for pools 1 and 2. For example $M_{3,1}^4 = (+, 2, 1)$ and $M_{2,1}^4 = (−, 4, 1)$. Positive nodes are nodes which correspond to positive monomials in the uPC proof, while negative nodes correspond to negative monomials.

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

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

In particular, IND-END-OF-LINE\textsuperscript{u\textit{d}}$(S_F)$ = $\Theta(uPC(F))$.

A proof of this theorem is given in the Appendix.

**Unary DAG-Like Sherali-Adams.**

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

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

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

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

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

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

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

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

### 3 Communication TFNP and Monotone Circuit Complexity

In addition to proof system characterizations of black-box TFNP problems, the communication versions of TFNP problems have provided characterizations of monotone circuit models [26, 32, 45]. When combined with lifting techniques translating decision tree lower bounds to communication complexity lower bounds, this has resulted in numerous new lower bounds for a variety of monotone circuit models. For example, bounds on the \( \mathbb{F}_2 \)-Nullstellensatz proof system, which is characterized by black-box PPA were lifted to communication-PPA lower bounds, which characterizes \( \mathbb{F}_2 \)-monotone span programs [40]. Conversely, as described in the introduction, a black-box and communication characterization of the same TFNP subclass generically gives rise to a monotone interpolation theorem, translating small proofs in the associated proof system into efficient computations in the associated model of computation.

In this section, we give generic conditions under which a monotone circuit model has a communication-TFNP characterization. We will formalize monotone circuit models as complexity measures on partial monotone functions. As has been pointed out in the past, there is a direct mapping from TFNP problems to partial monotone functions, and we utilize this mapping. This will allow us to give an exact characterization of when a complexity measure on partial functions has a TFNP characterization, proving Theorem 3. Since complexity measures on total functions induce complexity measures

---

5 Note that we could have added this final pool solution to our definition of \( IND-END-OF-LINE \) without changing its complexity. Indeed, this solution just enforced that the final line is \(-1\) in the uPC proof, which can be assumed without loss of generality, and thus \( IND-END-OF-LINE \) with the final pool solution reduces to \( IND-END-OF-LINE \).
on partial functions, this also gives a general condition under which a complexity measure on total
monotone functions has a TFNP characterization. Unfortunately, we don’t have a converse statement
for total functions and it is conceivable that measures that don’t meet our criteria also have TFNP
characterizations.

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

3.1 Communication TFNP

For \( n \) bit strings \( x \) and \( x' \), we say that \( x' \) dominates \( x \), written \( x \leq x' \), if \( x_i \leq x'_i \) for every \( i \in [n] \). A
partial Boolean function \( f \) on \( n \) bit strings is described by two disjoint sets of inputs, \( \text{No}_f \) which is
the set of strings that \( f \) rejects, and \( \text{Yes}_f \), the strings that it accepts. \( f \) is total if \( \text{No}_f \cup \text{Yes}_f = \{0,1\}^n \).
A partial Boolean function \( f \) is monotone if whenever \( x \in \text{No}_f \) and \( x' \leq x \), then \( x' \in \text{No}_f \) and
whenever \( y \in \text{Yes}_f \) and \( y \leq y' \) then \( y' \in \text{Yes}_f \). For partial functions \( f \) and \( g \), we say \( f \) is solved by \( g \)
if \( \text{No}_f \subseteq \text{No}_g \) and \( \text{Yes}_f \subseteq \text{Yes}_g \). 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 \( \text{Yes}_{f\circ h} = \{x|h(x) \in \text{Yes}_f\} \) and \( \text{No}_{f\circ h} = \{x|h(x) \in \text{No}_f\} \). If \( h \) is monotone
in its input, and \( f \) is monotone, then \( f \circ h \) is monotone.

3.1.0.1 Monotone Partial Function Complexity Measures.

A monotone partial function complexity measure \( \text{mpc} \) is a map from partial monotone functions to
non-negative integers that is Monotone Under Solutions: whenever \( g \) solves \( f \), \( \text{mpc}(g) \geq \text{mpc}(f) \).\(^6\)
Typical such measures are the minimum circuit size in a monotone model of a total function that
solves \( f \), but we won’t include a circuit model explicitly.

We are now ready to define what a communication-TFNP characterization of a measure means.
For a partial Boolean function \( f \) on \( n \) inputs, the Karchmer-Wigderson game for \( f \), denoted \( \text{KW}_f \), is
the communication problem where one player has \( x \in \text{No}_f \) the other has \( y \in \text{Yes}_f \) 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 \( m\text{KW}_f \), is a restriction of the Karchmer-Wigderson game
to require that the output is a position \( i \) such that \( x_i < y_i \). Karchmer and Wigderson [32] showed that
communication complexity of \( \text{KW}_f \) (\( m\text{KW}_f \)) is an exact characterization of the (monotone) circuit
depth needed to compute \( f \), or equivalently communication-FP.

3.1.0.2 Communication TFNP.

Consider relational communication problems defined by a predicate \( R \subseteq X \times Y \times [\ell] \). The
corresponding communication problem has one player given \( x \in X \), the other \( y \in Y \), and the goal
being to output an index \( i \) so that \( R(x, y, i) \) holds. We say this problem is in \( \ell \)-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 \( \ell \)-bit communication
protocol \( V(x, y, i) \) to determine whether \( R(x, y, i) \) holds. We say that \( R \in \text{TFNP}^{\text{cc}} \) if \( R \) is in
polylog(\( n \))-bit communication TFNP.

We say that one communication problem \( R \subseteq X \times Y \times [\ell] \) mapping reduces to another \( R' \subseteq
X' \times Y' \times [\ell'] \) with communication \( t \) if there are functions \( M_X : X \to X' \), \( M_Y : Y \to Y' \) and a

\( ^6 \) Recall that a partial function \( g \) solves \( f \) if \( \text{No}_f \subseteq \text{No}_g \) and \( \text{Yes}_f \subseteq \text{Yes}_g \).
We claim that \( mKW \) would allow totally unrelated TFNP subclasses to be used in a characterization, e.g., a class that is paddable.

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

**Proof.** Let \( S(x, y, j) \) be a \( t \)-bit protocol that verifies that \( j \in [\ell] \) is a valid solution on input \( (x, y) \).

We define a partial function \( F \) on \( N = 2^\ell \) 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 \( N_o_F \) as follows: for each \( j \in [\ell] \) and \( t \)-bit communication pattern \( p \in \{0, 1\}^t \) we set

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

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

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

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 \( \alpha_{(j, p)}(x) = \beta_{(j, p)}(y) \) 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 \( \alpha_{(j, p)}(x) > \beta_{(j, p)}(y) \), then we know that \( S(x, y, j) \) accepts, and we can return \( j \).

Thus, we can restrict attention to instances of the monotone Karchmer-Wigderson search problem.

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

\[
R^{cc}(mKW_f) := \min \log n + t,
\]

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

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

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

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

### 3.2 Complete Problems give TFNP Characterizations

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

\begin{enumerate}
  \item *Closed Under Reductions.* Say that an mpc is closed under reductions if for any \( h : \{0, 1\}^n \rightarrow \{0, 1\}^{n'} \) that is computable by monotone Boolean circuits of depth \( d \), and any partial monotone
  function \( f \) on \( n' \) bit inputs, \( \text{mpc}(f \circ h) \leq \text{poly}(n, n', \text{mpc}(f), 2^d) \).
  \item *Admits a Complete Family.* A complete family for an mpc is a family \( F_m \) of partial functions
  on \( N(m) \leq \text{quasi_poly}(m) \) bit inputs such that for every partial monotone function \( f \) with
  \( \text{mpc}(f) \leq m \), there is a \( \text{polylog}(m) \)-depth monotone circuit computing a function \( h \) so that
  \( F_m \circ h \) solves \( f \), and \( \text{mpc}(F_m) \leq \text{quasi_poly}(m) \).

\end{enumerate}

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

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

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

\textbf{Lemma 14.} Let \( f \) and \( g \) be monotone partial Boolean functions. Then mKW\(_f\) has a communication-
\( t \)-mapping reduction to mKW\(_g\) \( \iff \) there is a function \( h \) computable by a depth-\( t \) monotone circuit so
that \( g \circ h \) solves \( f \).

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

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

\footnote{Note that in the definition of admitting a complete family are insisting that \( f \) reduce to \( F_m \) for an \( m \) only dependent
on its complexity, not its input size. Most natural notions of circuit complexity have circuit size be always at least the
number of bits the function actually depends on, and the reduction can ignore the irrelevant bits, so this should not
usually be a problem.}
Let $m_{KW}$ 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 $NO_f \cup Yes_f$, which outputs the $i$-th bit of $h$. Since $h$ is computable by depth-$t$ monotone circuits, so is $h_i$. Thus, by the Karchmer-Wigderson transformation [32], there is a $t$-bit communication protocol $S_i(x, y)$ for $m_{KW}$. 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 $m_{KW}$. 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 $m_{KW}$ to $m_{KW}$. From the protocol $S$, which maps solutions $i$ to $m_{KW}$ on input $M_X(x), M_Y(y)$ back to solutions $S(x, y, i)$ to $m_{KW}$ 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 $x' \in NO_f$ and $M_X(x') = 0$, and whose yes-inputs are those $y$ for which there is $y \leq y'$ with $y' \in Yes_f$ and $M_X(y') = 1$; we call such an input pair a dominating and dominated pair for $H_i$.

By the definition of reduction, whenever $x' \in NO_f, M_X(x') = 0, y' \in Yes_f$ and $M_Y(y') = 1$, the communication protocol $S(x', y', i)$ returns a position $j$ with $x'_j < y'_j$. Given any input pair $(x, y)$ to $m_{KW}$, 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 $m_{KW}$. Game. Therefore, by the Karchmer-Wigderson transformation [32], there is a depth-$t$ monotone circuit computing a function $h_i$ that rejects all $x \in NO_f$ with $M_X(x) = 0$ and accepts all $y \in Y_f$ with $M_Y(y) = 1$; it follows that $h_i(x) \leq M_X(x)$, for all $x \in NO_f$, and if $y \in Yes_f$ then $M_Y(y) \leq h_i(y)$. Letting $h = (h_1, \ldots, h_n)$, where $n$ is the number of input bits to $f$, we have that for each $x \in NO_f$, $h(x) \leq M_X(x) \in NO_g$, so by monotonicity of $g$, $h(x) \in NO_g$. Similarly, if $y \in Yes_f$, $M_X(y) \leq h(y)$ and $h(y) \in Yes_g$. Thus, $g \circ h$ solves $f$ and $g$ is computable by depth-$t$ monotone circuits.

We will now use the lemma to prove the theorem.

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

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

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

Next we prove the converse direction of the theorem. Let $R_n$ be any paddable sequence of communication TFNP problems and define a monotone partial function complexity measure $mpc$ as $mpc(f) := 2^{R^cc(m_{KW})}$ for every monotone partial function $f$. By construction, $mpc$ is monotone under solutions. We will show that $mpc$ has the properties (i) and (ii). First, assume $g \circ h$ solves $f$ and $h$ is computable by
 monotone (total function) complexity measure.

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

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

\section*{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

\[ \text{mpc}(F) := \min \{ \text{mc}(f) : \text{total } f \text{ solving } F \} \]

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 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 mc(F_m) \leq poly(m).

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

\textbf{Lemma 15.} mpc is closed under reductions and has a complete (partial function) family if and only if mc is closed under reductions and has a complete total function family.

An immediate consequence is the following.

\textbf{Corollary 16.} If a monotone complexity measure mc is closed under reductions and has a complete family, then it has a TFNP_{cc} characterization by a sequence of paddable relations. If not, mc has no such characterization.

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

\textbf{Proof of Lemma 15.} To prove the lemma, we will first assume mc is closed under reductions, e.g., mc(f \circ h) \leq poly(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 mpc(F \circ h) \leq mc(f \circ h) \leq poly(mc(f), 2^d) = poly(mpc(F), 2^d). Conversely, since mpc(f) = 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 mpc($f_m$) = mpc($F_m$) = quasipoly($m$). Let $g$ be any total function and let $m = \text{mpc}(g) = \text{mc}(g)$. Then there is a function $h$ computable by 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 $m$, then let $G$ be any partial function, let $g$ be a minimal complexity total function solving $G$, and let $m = \text{mpc}(G) = \text{mc}(g)$. Then $g = f_m \circ h$ for some function $h$ computable by polylog$m$-depth circuits, and so solves $G$. Thus, $f_m$ is also complete for mpc.

### 3.3 Universal Functions vs. Complete Functions

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

ii$^\dagger$) 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 mpc$(g) \leq m$ , there is a fixed string $z_g$ so that $F(x \circ z_g)$ solves $g(x)$. Observe that such an $F_m$ can be viewed as complete under depth 0 reductions.

▶ Theorem 17. Let mpc be a monotone partial function complexity measure satisfying (i) and (ii).

Then mpc admits a universal family if and only if it admits a complete family.

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

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

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

Conversely, say that a monotone partial complexity measure mpc admits a complete family under $d(m)$-depth reductions if there exists a family $G_m$ of functions such that mpc($G_m$) $\leq 2^{d(m)}$ and for every partial monotone function $f$ with 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 $|x| = M \leq \text{poly}(m)$. We want to construct a partial function $F_m$ which can code any composition $g(x) = G_m(h(x))$ for any $g$ with 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 $\land$-$\lor$ 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 $\lor$-gates, and those at odd levels $\land$-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 \( H_j = \sqrt{H_i \land B_{i,j}} \) at even levels, and \( H_j = \bigvee(H_i \lor B_{i,j}) \) 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 \( M2^{d(m)} \land \) and \( \lor \)'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, \( mpc(F_m) \) is quasi-polynomial in \( m \). Also, for any \( g \) with \( mpc(g) \leq m \), \( g \) can be solved by \( F \circ h \) where \( h \) can be computed by monotone depth-\( d \) circuits.

The input \( z_g \) includes the values \( B_{i,j} \) according to the connectivity for \( h \); unused bits in \( z_g \) can be set to 0. By construction, \( F_m(x \circ z_g) = G_m(h(x)) \) which solves \( g \).

### 4 Future Directions

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

In particular,

1. We have a generic relationship between proof systems and decision tree TFNP problems, and a generic relationship between monotone circuit complexity problems and circuit lower bounds. Can we complete the chain by proving a generic lifting theorem, and show that for each TFNP problem, lower bounds for the corresponding proof systems and complexity measures are equivalent?
2. Our characterization of proof systems that correspond to TFNP problems involves proving their own soundness. Can we use this to show a version of Gödel’s second incompleteness theorem, that some proof systems cannot prove their own soundness because they do not have a tight TFNP connection?
3. TFNP has a direct connection to monotone complexity via the monotone KW games. Can we similarly characterize the class of communication problems corresponding to non-monotone KW games?
4. We showed that reductions between the monotone KW games were equivalent to small depth monotone reductions between the corresponding functions. Does this extend to non-monotone games and non-monotone reductions? If not, can we give an example of functions with reductions between the KW games and no reductions between the corresponding functions? (Since this is interesting even for sub-logarithmic bit reductions, this could possibly be shown unconditionally without proving new formula lower bounds.)

### References

TFNP Characterizations of Proof Systems and Monotone Circuits


Appendix: Proof of Theorem 11

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

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

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

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

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

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

Case 1. If $\ell$ was derived by addition from some $\ell', \ell'' \neq \ell$ then every monomial $m$ in line $\ell$ comes from one of $\ell', \ell''$ — suppose that $m$ comes from $\ell'$ — and so we match $m$ to the copy of $m$ in $\ell'$.

If $m$ has a positive coefficient in $\ell$, then we set $M_{\ell,m}^\ell = (+, \ell', m)$ and $M_{\ell,m}^\ell = (-, \ell, m)$, and if it has a negative coefficient we set $M_{\ell,m}^\ell = (-, \ell', m)$ and $M_{\ell,m}^\ell = (+, \ell, m)$.

It remains to define the matchings for all monomials $m$ which occur in $\ell'$ or $\ell''$ but not in $\ell$; suppose that $m$ belongs to $\ell'$. For this to happen, $m$ must have cancelled with a negative
will be in a violated clause of 
otherwise.

\[ \ell \]

derive by induction on 
it appears at the head of an arrow in 
instance let 
accepting 
\[ T \]

Lemma 19.

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

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

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

Lemma 19. Let \( F \) be an unsatisfiable CNF formula. If \( S_F \) reduces to an instance of \( \text{IND-END-OF-LINE} \) on \( n \) variables using depth-\( d \) decision trees, then there is a degree-\( O(d) \) and size \( n^{3^d2^{O(d)}} \) \( u\text{PC} \) proof of \( F \).

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

For \( \ell \in [L] \) define the polynomial

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

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

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

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

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

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

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

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

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

We will represent (ii) by the polynomial

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

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

\[\triangleright \text{Claim 4.}\] For any $\ell \in [L]$, $\text{deriv}^{(\ell)} = 0$ has a degree-$O(d)$ and size-$NL2O(d)$ uPC proof from the axioms.

\[\triangleright \text{Claim 5.}\] For any $\ell \in [L]$ and $\ell' < \ell$, $\text{consist}^{(\ell)}_{\ell'}$ has a degree-$O(d)$ and size-$NL2O(d)$ uPC proof from the axioms.

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

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

Summing these polynomials with $\text{deriv}^{(\ell)} = 0$ gives $q_\ell = 0$. We apply Claim 4 $\ell \leq L$ times and Claim 5 once. Thus, this induction step can be performed in degree $O(d)$ and size $NL^22O(d)$.  \[\square\]
Proof of Claim 4. For \( \ell' \leq \ell, m \in [N] \) and \( \alpha \in \{-, +\} \) define
\[
\operatorname{match}(\ell, m, \ell') := \sum_{m^* \in [N]} \left[ M(\ell, m, \ell') = (\alpha, m^*, \ell^*) \right] \sum_{\gamma, \delta \in \{0, 1\}} \left[ P(\ell, m^*, \ell') = \gamma \right] \left[ A(\ell, m^*, \ell') = \delta \right],
\]
which records whether node \( m \) belonging to \( \ell' \) is at the head or tail of an arrow, and whether it is correctly matched in the matching \( M(\ell) \) for \( \ell \). Note that
\[
\sum_{\gamma, \delta \in \{0, 1\}} \left[ P(\ell, m^*, \ell') = \gamma \right] \left[ A(\ell, m^*, \ell') = \delta \right] = 1, \tag{1}
\]
as it is the polynomial obtained from summing over all paths in the stacked decision tree obtained by running the decision trees for \( P(\ell, m^*, \ell') \) and then \( M(\ell, m, \ell') \).

Now, consider the polynomial \( P(\ell, m^*, \ell') \operatorname{match}(\ell, m, \ell') \) and partition its terms into two sets, a set \( C(\ell, m) \) which corresponds to correct matchings — that is, \( m \) is matched to a node \( m^* \in [N] \) belonging to a pool \( \ell^* \leq \ell \) (\( M(\ell', m^*) = (\alpha, \ell^*, m^*) \)) with \( P(\ell, m^*, \ell') = 1 \) and \( A(\ell, m^*, \ell') = 1 \), which is matched back to \( m \), meaning that \( M(\ell', m^*) = (\gamma, \ell', m) \), where \( \gamma \) is the opposite sign of \( \alpha \) — and \( E(\ell^*, m^*) \) which will contain the remaining terms, corresponding to erroneous matchings. Using these polynomials, define
\[
\operatorname{match}(\ell) := \sum_{\ell' \in [\ell]} \sum_{m \in [N]} A(\ell') \cdot P(\ell, m^*, \ell') \left( \operatorname{match}(\ell, m, \ell') - \operatorname{match}(\ell, m, \ell') \right),
\]
which records the matching for pool \( \ell \). By (1), this polynomial is equivalent to \( \operatorname{deriv}(\ell) \), and therefore it suffices to show that this polynomial has a low-degree derivation from the axioms. To do so, partition the terms of \( \operatorname{match}(\ell) \) into three sets, \( C_+, C_-, E \) as above, where \( C_+ = \bigcup C\ell,m,\alpha \) for \( \alpha \in \{-, +\} \), and \( E = \bigcup E(\ell^*, m^*) \cup E(\ell, m^*) \) where the unions are taken over \( \ell' \leq \ell \) and \( m \in [N] \).

Observe that because the matchings in \( C_+ \) and \( C_- \) are correct, for every node at the head of an arrow, a node occurs at the tail of that arrow. It follows that \( \sum_{t \in C_+} t - \sum_{t' \in C_-} t' = 0 \).

Next, consider a term \( t \in E \). This term corresponds to a node \( m \) in some pool \( \ell' \leq \ell \) that is incorrectly matched; let \( s \) be this incorrect matching. We will denote by \( t_s \) the term \( t \) witnesses \( s \). Let \( T_s \) be the output decision tree for solution \( s \) and abuse notation by letting \( T_s \) also denote the polynomial formed by taking the sum over all of the paths in the decision tree \( T_s \). Recalling that the sum over all paths in a decision tree is 1,
\[
\operatorname{match}(\ell) = \sum_{t \in E} t - \sum_{t' \in E} t' + \sum_{t_s \in E} t_s = 0 + \sum_{t_s \in E} t_s = \sum_{t_s \in E} t_s \cdot T_s.
\]
An incorrect matching is a solution to \textsc{IND-END-OF-LINE}. Therefore, because this instance solves \( S_F \), any truth assignment \( x \) which satisfies \( t_s \) must falsify the \( T_s \)-th clause of \( F \). It follows that each term of \( t_s \cdot T_s \) that is not identically 0 must contain the polynomial \( C = 0 \) for some clause \( C \) of \( F \). Thus, \( t_s \cdot T_s \) can be derived by multiplication from the axioms \( C = 0 \) and \( -C = 0 \). It follows that \( \operatorname{deriv}(\ell) \) has a proof of degree at most the degree and size of \( \operatorname{match}(\ell) \), which are \( 6d \) and \( NL2^{O(d)} \) respectively.
Proof of Claim 4. For $\alpha \in \{-, +\}$, define the \textit{polarity polynomial}
\[
\text{pol}_\alpha^{(\ell')} := P_\ell^{(\ell')} \sum_{m \in [N]} A_{m}^{(\ell')} \sum_{\ell' \leq \ell \cdot m \in [N]} \left[ M_{\ell' \cdot m}^{(\ell')} = (\alpha, \ell', m^*) \right] \sum_{\beta \in \{-, +\}} \left[ M_{\ell' \cdot m}^{(\ell')} = (\beta, \hat{\ell}, \hat{m}) \right].
\]
which records for each node at the $\alpha$-end of an arrow in the matching for $\ell'$, which end of an arrow it occurs at in the matching for pool $\ell'$. We will partition the set of terms of this polynomial into two sets, $C^{(\ell')}_{\alpha}$ and $E^{(\ell')}_{\alpha}$. $C^{(\ell')}_{\alpha}$ will be the terms $t$ which are the indicators of correct assignments of polarities of the nodes in pool $\ell'$ in the matchings $M^{(\ell')}$ and $M^{(\ell')}$ — that is, if $m$ is an active node for $\ell'$ and $m$ occurs at the head of an arrow in the matching for $M^{(\ell')}$ then it is at the tail of an arrow in the matching for $M^{(\ell')}$ if $\ell'$ is a predecessor of $\ell$. $E^{(\ell')}_{\alpha} C^{(\ell')}_{\alpha}$ will be the remaining terms which correspond to erroneous assignments of polarities. As well, observe that
\[
\text{pol}_\alpha^{(\ell')} = P_\ell^{(\ell')} \sum_{m \in [N]} A_{m}^{(\ell')} \sum_{\ell' \leq \ell \cdot m \in [N]} \left[ M_{\ell' \cdot m}^{(\ell')} = (\alpha, \ell', m^*) \right] \sum_{\beta \in \{-, +\}} \left[ M_{\ell' \cdot m}^{(\ell')} = (\beta, \hat{\ell}, \hat{m}) \right] \cdot 1,
\]
as $\sum_{\ell \leq \ell \cdot m \in [N], \beta \in \{-, +\}} [M_{\ell \cdot m}^{(\ell')} = (\beta, \hat{\ell}, \hat{m})]$ is the polynomial obtained by taking a sum over all paths in the decision tree for $M_{\ell' \cdot m}^{(\ell')} = (\beta, \hat{\ell}, \hat{m})$, which sums to 1.
Similarly, let
\[
\text{pol}_\alpha^{(\ell')} := P_\ell^{(\ell')} \sum_{m \in [N]} A_{m}^{(\ell')} \sum_{\ell' \leq \ell \cdot m \in [N]} \left[ M_{\ell' \cdot m}^{(\ell')} = (\alpha, \ell', m^*) \right] \sum_{\beta \in \{-, +\}} \left[ M_{\ell' \cdot m}^{(\ell')} = (\beta, \hat{\ell}, \hat{m}) \right],
\]
be the polynomial which records for each active node of $\ell'$ which occurs at the $\alpha$-end of an arrow in $M^{(\ell)}$, which end of an arrow it occurs at in $M^{(\ell')}$. Define $C^{(\ell')}_{\alpha}$ and $E^{(\ell')}_{\alpha}$ analogously, and note that
\[
\text{pol}_\alpha^{(\ell')} = P_\ell^{(\ell')} \sum_{m \in [N]} A_{m}^{(\ell')} \sum_{\ell' \leq \ell \cdot m \in [N]} \left[ M_{\ell' \cdot m}^{(\ell')} = (\alpha, \ell', m^*) \right] \sum_{\beta \in \{-, +\}} \left[ M_{\ell' \cdot m}^{(\ell')} = (\beta, \hat{\ell}, \hat{m}) \right],
\]
by the same reasoning as above.
Putting these together, we have
\[
\text{consist}_{\ell'}^{(\ell')} = \text{pol}_-^{(\ell')} - \text{pol}_+^{(\ell')} - \text{pol}_+^{(\ell')} + \text{pol}_+^{(\ell')}.
\]
We will derive $\text{pol}_+^{(\ell')} - \text{pol}_-^{(\ell')} = 0$ and $\text{pol}_-^{(\ell')} - \text{pol}_+^{(\ell')} = 0$ separately from the axioms, beginning with $\text{pol}_-^{(\ell')} - \text{pol}_-^{(\ell')} = 0$. Consider any term $t$ in $C^{(\ell')}_{+}$ and observe that since $t$ is \textit{correct}, it records that an active monomial $m$ of $\ell'$ which occurs at the head of an arrow in $M^{(\ell')}$ occurs at the tail of an arrow in $M^{(\ell')}$, thus, $t$ occurs also in $C^{(\ell')}_{-}$. By a symmetric argument, any term $t$ occurring in $C^{(\ell')}_{-}$ occurs in $C^{(\ell')}_{+}$. Thus, $\sum_{t \in C^{(\ell')}_{+}} t = \sum_{t \in C^{(\ell')}_{-}} t = 0$, and also $\sum_{t \in C^{(\ell')}_{+}} t = \sum_{t \in C^{(\ell')}_{-}} t = 0$ by a similar argument. Denoting the union of all of the error sets by $E := E^{(\ell')}_{+} \cup E^{(\ell')}_{-} \cup E^{(\ell')}_{+} \cup E^{(\ell')}_{-}$, we have
\[
\text{consist}_{\ell'}^{(\ell')} = \left( \sum_{t \in C^{(\ell')}_{+}} t - \sum_{t \in C^{(\ell')}_{-}} t \right) + \left( \sum_{t \in C^{(\ell')}_{+}} t - \sum_{t \in C^{(\ell')}_{-}} t \right) + \sum_{t \in E} t = 0 + \sum_{t \in E} t.
\]
It remains to show that each term $t \in E$ can be derived from the axioms with a low-degree uPC proof. As each $t \in E$ witnesses a node which switched polarity between the matching for line $\ell'$ and the matching for line $\ell$, this is a solution $s$ to \textit{IND-END-OF-LINE}; we will denote $t$ by $t_s$ to record the fact that $t$ witnesses solution $s$. Let $T^o_s$ be the output decision tree corresponding to solution $s$, and
abuse notation by identifying it with polynomial formed by taking the sum over all paths in $T_s^C$. As
the sum over all paths in a decision tree gives the 1 polynomial, we have $t_s = t_s \cdot T_s^C$. As $t_s$ witnesses
solution $s$, it follows that any assignment $x$ such that $t_s(x) = 1$ must falsify the $T_s^C(x)$-th clause $C$ of
$F$. Thus, $t_s \cdot T_s^C$ can be derived from the axioms $\overline{C} = 0$ and $-\overline{C} = 0$. It follows that

$$\operatorname{consist}_C^{(d)} = 0 + \sum_{t_s \in E} t_s \cdot T_s^C = 0$$

has a $\mu\mathcal{P}C$ proof from the axioms of degree at most $4d$ and size $NL^{O(d)}$. $\blacktriangleleft$