Electronic Colloquium on Computational Complexity, Report No. 129 (2023)

Understanding Nullstellensatz for QBFs

Sravanthi Chede
Indian Institute of Technology Ropar, Rupnagar, India

Leroy Chew
TU Wien, Austria

Balesh Kumar
Indian Institute of Technology Ropar, Rupnagar, India

Anil Shukla
Indian Institute of Technology Ropar, Rupnagar, India

Abstract

QBF proof systems are routinely adapted from propositional logic along with adjustments for the new quantifications. Two main successful frameworks currently exist, the reduction [23] and expansion [7] frameworks, inspired by QCDCL [32] and CEGAR solving [22] respectively. However, the reduction framework, while immensely useful in line-based proof systems, is not refutationally complete for static proof systems.

Nullstellensatz (NS,[6]) is a well known static propositional proof system, inspired by Hilbert's theorem [21] of the same name. It falls into the category of algebraic proof systems such as the Polynomial Calculus (PC,[14]) or the Ideal Proof System (IPS,[19]). In this paper, we initiate the study of the NS proof system for QBFs, using the existing expansion (∀Exp) framework and a new "∀Strat" framework. We introduce four new static, QBF refutation systems: ∀Exp+NS, ∀Exp+NS′, ∀Strat+NS and ∀Strat+NS′, which use NS and a more powerful version of NS (NS′) with twin variable encoding that allows it to simulate tree-like Resolution without an exponential blowup.

We explore relationships among the proposed systems and analyse their proof sizes for a few well-known hard QBF-families. We find that ∀Exp+NS′ and ∀Strat+NS′ are incomparable. This result is in line with the incomparability result between ∀Exp+Res and Q-Res [10].

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Proof systems, Nullstellensatz, Resolution, QBF, simulation, lower bound, strategy

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Cook and Reckhow [1] defined a proof system to be a polynomial-time function on strings whose range is exactly the language of the theorems in question. When this language is the set of all unsatisfiable Boolean formulas, the corresponding proof systems are called propositional proof systems. The inputs to the proof systems are the proofs themselves. Depending on the proof system and language, the proof can be in a conventional format like truth tables and DAGs (Directed acyclic graphs) or be unconventional and take the form of say, a directed graph with cycles [5] or a system of polynomial coefficients [6]. Outside of pathological cases, there are two types of proof systems: dynamic and static. The 'dynamic (refutational) proof systems consist of a set of sound rules and every valid proof in this system derives a contradiction using these rules. Many dynamic systems exist in the literature like Resolution [27] and Frege [16]. On the other hand in static proof systems, the proofs can be seen as one large inference like truth tables. Many static systems exist in the literature like the aforementioned Truth Table, Sum-of-squares (SOS) [18] and Nullstellensatz (NS) [6].

The NS proof system works with polynomial equations. In order to use it for refuting unsatisfiable Boolean formulas, one has to encode clauses as polynomial equations. In the

ISSN 1433-8092

standard encoding positive literals ℓ are encoded as $(1-\ell)$. With the standard encoding (Definition 1), NS has been shown to be incomparable with (tree-like) Resolution [11]. A dynamic version of NS has been introduced in the literature known as Polynomial Calculus (PC) [14]. Owing to the standard encoding, PC cannot efficiently simulate Resolution [25, Lemma 6.2.1]. This was overcome by a variant of Polynomial Calculus with Resolution (PCR) [2] which uses twin variables encoding (Definition 4). In this paper, we observe that adding twin variables encoding to NS (denoted by NS$'$) makes it powerful enough to simulate tree-like Resolution (Res_T, Lemma 6). Unable to find such results in the literature, we are discussing the same explicitly as observations in Section 3.

Quantified Boolean Formulas (QBFs) are an extension to Boolean formulas by quantifying every variable with a $\exists$ or $\forall$. Many dynamic propositional proof systems have been lifted for QBFs like eFrege+$\forall$red [8], Q-Res [24], QU-Res [30], and $\forall$Exp+Res [23]. In this paper, we initiate the study of the basic propositional static system 'Nullstellensatz' for the QBF domain. Static proof systems provide a new challenge for QBF proof theory and proof complexity, and motivates us to find new quantifier techniques. We list out the major contributions of this paper in the following subsection.

1.1 Our Contributions

1. **Introducing a new $\forall$Strat framework for QBFs:** In order to use propositional proof systems for QBF solving, there exists mainly two frameworks: $\forall$Expansion [23] and $\forall$Reduction [7]. The expansion-based framework uses the semantics of the universal quantification and expands the clauses as needed with 0/1-assignments to universal variables. Then propositional proof systems can be applied on the expanded existential clauses. The reduction-based framework adds the $\forall$Red rule (Sec: 2) to the propositional system to make it complete for QBFs [7]. Observe that the former method works for both the dynamic and static propositional proof systems. However, the latter method cannot be applicable for static systems.
   In this paper, we introduce a new '$\forall$Strat framework' which works for both the dynamic and static propositional systems. The idea behind the new framework is simple: For any false QBF, the universal player has a winning strategy in the two player semantic game. By forcing the universal player to play according to this strategy leads to a false (unsatisfiable) propositional formula (Observation 9). Any propositional system P can then refute the resulting formula.
   The real challenge in implementing this idea is to carefully define the format of the added strategy clauses so that it can be easily verifiable. For any such encoding E and a propositional proof system P, we introduce a family of QBF proof systems $\forall\mathsf{Strat}_E$+P (Definition 10).
2. **Introducing NS based proof systems for QBFs**: NS [6] is a simple, static propositional system. We use both the $\forall$Expansion and the new $\forall$Strat framework to study this system for QBFs. In the expansion framework we introduce sound and complete QBF-proof systems $\forall$Exp+NS and $\forall$Exp+NS$'$ (Definition 8). These systems use the axiom download step of $\forall$Exp+Res [23] and encode the downloaded clauses using standard (Definition 1) and twin (Definition 4) encoding respectively. These encoded clauses are then refuted by NS and NS$'$ (Section 3.1) systems respectively.
   In the $\forall$Strat framework, we introduce QBF-proof systems $\forall\mathsf{Strat}_{DL}$+NS and $\forall\mathsf{Strat}_{DL}$+NS$'$ which use the base systems as NS and NS$'$ respectively. Here the encoding DL stands for decision lists [26]. The idea is to represent the universal player's winning strategy as a decision list (Section 2) and encode it using the standard or twin encodings. We

use Tseitin variables to represent each line of the decision list for ease in verification (Definition 14).

3. **Analyzing the strengths and limitations of proposed systems**: Analogous to the in-comparability result of ∀Exp+Res and Q-Res [10], we show that the systems ∀Exp+NS′ and ∀Strat$_{DL}$+NS′ are incomparable (Theorem 22). We show this by providing an easy refutation of QPARITY$_n$ (Definition 23) in ∀Exp+NS′ (Lemma 26) which are hard for ∀Strat$_{DL}$+NS′ by our observation (Theorem 25). For the other direction, we give an easy refutation of JM$_n$ (Definition 27) in the ∀Strat$_{DL}$+NS′ system (Theorem 30) which are hard for ∀Exp+NS′ by our observation (Proposition 29). We also make an observation that eFrege+∀red [8] p-simulates ∀Strat$_{DL}$+NS′ and ∀Strat$_{DL}$+NS (Theorem 31).

Few other relations with existing proof systems are inherited from the propositional world. We summarize our contributions in Figure 1.

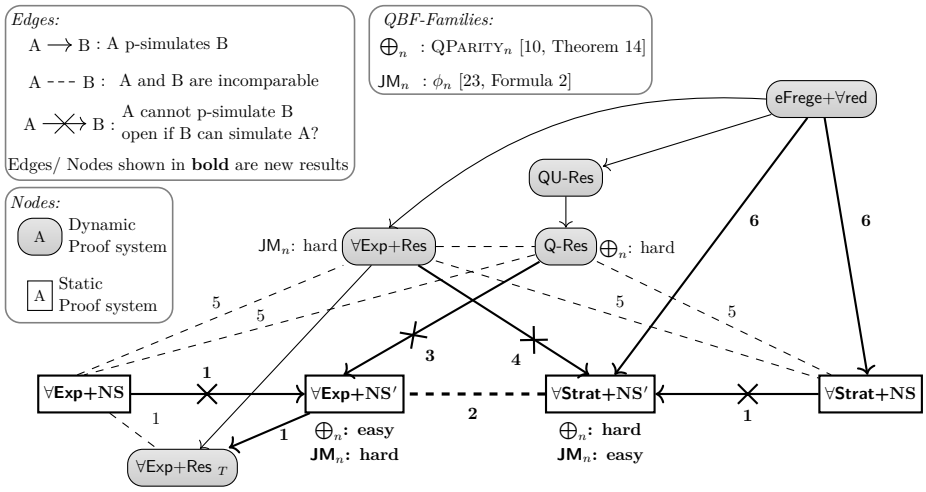

Figure 1 *The proposed systems and their relative complexity with existing QBF-proof systems.* Edges labelled 1 follow from Lemma 6 and incomparability of NS with Res$_T$. Edge 2 is from Theorem 22. Edges 3, 4 appear because of QPARITY$_n$ and JM$_n$ respectively. Edges labelled 5 follow from the incomparability between NS and Resolution. Edges labelled 6 are from Theorem 31.

2 Preliminaries

For a Boolean variable x, its literals can be x (positive literal) and $\overline{x}$ (negative literal). A clause C is a disjunction of literals and a conjunctive normal form (CNF) formula F is a conjunction of clauses. We denote the empty clause by $\bot$. $vars(C)$ is a set of all variables in C and width$(C) = |vars(C)|$. If S is any set of variables, a complete assignment of S will be an assignment which assigns all variables in S to either 1 or 0. Similarly, a partial assignment is an assignment which assigns a subset of variables in S to either 1 or 0. For some clause C and some partial assignment of variables α, $C|_\alpha$ is defined as replacing all occurrences of variables in C with the assignments from α (where defined) and simplifying it. $f(x) \vDash y$ denotes that for every assignment satisfying $f(x)$ should also satisfy y.

Horn formulas are a special case of CNF formulas where each clause is allowed to have at most one positive literal. Given a Horn formula H, there exists a well-known linear time algorithm to test the satisfiabilty of H [15].

Boolean circuit: [31] A Boolean Circuit C_n of n-inputs is a directed acyclic graph. It has n-input nodes and a single output node of out-degree $= 0$. All other nodes (gates) are labelled with $\neg, \wedge, \vee$. Fan-in and Fan-out are the number of incoming edges and outgoing

edges of a gate respectively. Depth of a circuit is the maximum length of a path from its input node to the output node.

AC^0 circuits are constant depth circuits of polynomial number of gates with unbounded fan-in for gates.

Tseitin Transformation: [29] Any Boolean circuit can be converted into a CNF by introducing a new variable for every gate and enforcing it to represent the correct value of the gate. That is, using such variables ensures that the assignments evaluating the circuit to 1 are the same which satisfy the CNF and vice-versa. The size of the CNF in this transformation is polynomial in the size of the initial circuit.

Proof Systems: Given a language $L \subseteq \{0,1\}^*$ and a string $x \in L$, a **proof system** f for L is an inference system, which is capable of showing that x is indeed in L. A proof system f for L is complete if and only if for every $x \in L$ we have a corresponding f-proof for x. A proof system f for L is sound if and only if the existence of an f-proof for x implies that $x \in L$. By definition, a proof system must be sound and complete for the language L. In addition, it must be polynomial time computable (verifiable). That is, given any claimed proof π in f, it must be checkable whether it is a correct proof in the system in time polynomial w.r.t the size of π, in which case, it is said to be a valid f-proof.

A proof system f efficiently simulates (p-simulates) another proof system g (i.e., $f \leq_p g$) if every g-proof of input $x \in L$ can be translated into an f-proof for the same input in time that is polynomial w.r.t size of the g-proof. Otherwise, we say that f does not simulate g ($f \not\leq_p g$). Proof systems f and g are said to be incomparable, if none of them can simulate the other. They are said to be equivalent ($f \equiv_p g$), if both can simulate the other.

Proof systems can be broadly sorted into two basic types. In a **static proof system** we require to present a refutation as a formula containing all the information about the refutation, and the complexity is the size of this formula or its degree. A **dynamic proof system**, defines a set of derivation rules and a refutation needs to be built from the given formula using these rules, and the complexity can be size of all derived formulas or the number of steps used.

Resolution: Resolution [27] is the most studied redundancy rule of the SAT world, we define the same as: $\frac{C \vee x \quad D \vee \overline{x}}{C \vee D}$, where C, D are clauses and x is the pivot variable. We denote this step as '$res(C \vee x, D \vee \overline{x})$' throughout the paper. It is a sound and complete, dynamic propositional proof system. For a Resolution proof π, G_π is the derivation graph with edges directed from the hypothesis to the resolvent. If G_π is a tree, we say π is a Res_T proof.

Nullstellensatz Proof System: The Nullstellensatz proof system (NS) [6] is a static propositional proof system which uses reasoning about polynomial equations to refute Boolean formulas. It is based on the Hilbert's Nullstellensatz [21] and as a proof system was first introduced by Beame.et.al in [6].

▶ **Definition 1** (Standard Encoding). *To convert a CNF to a list of polynomials, we need to convert every clause into a separate polynomial. For a clause* $C = (l_1 \vee \cdots \vee l_k \vee \overline{l_{k+1}} \vee \cdots \vee \overline{l_n})$, *its polynomial equation is denoted by* $\boldsymbol{poly_{std}}(\mathbf{C}) := (1 - l_1) \dots (1 - l_k) l_{k+1} \dots l_n = 0$. *Notice that the set of assignments that satisfy* C *and* $poly_{std}(C)$ *in both cases are exactly the same.*

Given a CNF formula $\mathcal{P} = C_1 \wedge \cdots \wedge C_m$ over variables $x_1, \dots, x_n$. The corresponding ordered-list of equations will be the $poly_{std}(\mathcal{P}) := (f_1 = 0, \dots, f_m = 0)$ where $f_i = poly_{std}(C_i)$ for $i \leq m$ as explained above. To maintain the Boolean nature of solutions, equations $x^2 - x = 0$ are introduced for all variables of $\mathcal{P}$. The final list of polynomial equations in the standard form is $\mathcal{F} := (f_1 = 0, \dots, f_m = 0,\ f_{m+1} := x_1^2 - x_1 = 0, \dots, f_{m+n} := x_n^2 - x_n = 0)$,

where the coefficients of the polynomials come from a fixed finite field $\mathbf{F}$ [1]. The proof system is defined as follows: A Nullstellensatz refutation of $\mathcal{F}$ is a ordered-list of polynomials $(g_1, \ldots, g_{m+n})$ from the polynomial ring $\mathbf{F}[\bar{x}]$ such that the following holds:

$$\sum_{i \in [m+n]} f_i(\bar{x}) g_i(\bar{x}) \ = \ 1$$

Complexity measures: The complexity measures of an NS refutation are degree and size. Degree is defined as $\max_{i \in [m+n]} (deg(f_i) + deg(g_i))$. A monomial is a product of variables and generally polynomials are represented as a sum of monomials. Hence, the size of an NS refutation is the sum of number of monomials in all g_is combined.

Polynomial Calculus: PC is a propositional proof system which is a dynamic version of the NS system. This was first considered by Clegg.et.al in [14]. Given a CNF formula $\mathcal{P} = C_1 \wedge ... \wedge C_m$ over variables $x_1, ..., x_n$. The corresponding standard encoded system of equations will be the $\mathcal{F} := \{f_1 = 0, ..., f_m = 0\}$ as in Definition 1.

([28]) A PC refutation of a system of equations $\mathcal{F}$ over a fixed field $\mathbf{F}$ is a sequence of polynomials $g_1, ..., g_r$ where $g_r = 1$ (as we would derive the contradiction $1 = 0$) and the rest of the g_is are derived using one of the following rules:

$$\frac{}{x_j^2 - x_j}$$

$$\frac{p \quad q}{\alpha p + \beta q} \ (\alpha, \beta \in \mathbf{F})$$

$$\frac{p}{rp} \ (r \text{ is a monomial})$$

Polynomial calculus with Resolution: PCR [2] is an extension of PC with just a change in the encoding of polynomials. That is, given a CNF formula $\mathcal{P} = C_1 \wedge \cdots \wedge C_m$ over variables $x_1, ..., x_n$. The corresponding standard encoded system of equations will be the $\mathcal{F} := \{f_1 = 0, \ldots, f_m = 0, 1 - x_1 - x_1' = 0, \ldots, 1 - x_n - x_n' = 0\}$ as in Definition 4. A PCR refutation of $\mathcal{F}$ over a fixed field $\mathbf{F}$ is a sequence of polynomials $g_1, ..., g_r$ where $g_r = 1$ and the rest of the g_is are derived using the rules of the PC system.

Quantified Boolean Formulas (QBFs): QBFs are an extension of the propositional Boolean formulas where each variable is quantified with one of $\{\exists, \forall\}$, with their general semantic meaning of existential and universal quantifier respectively. In this paper, we assume that QBFs are in closed prenex form with a CNF boolean formula called the matrix i.e., we consider the form $Q_1 X_1 \ldots Q_k X_k.\ \phi(X_1 \cup \cdots \cup X_k)$, where X_i are pairwise disjoint sets of variables; $Q_i \in \{\exists, \forall\}$ and $Q_i \neq Q_{i+1}$, and the matrix ϕ is in CNF form. We denote QBFs as $\mathcal{F} := \Pi.\phi$ in this paper, where Π is the quantifier prefix. For a variable x if $\Pi(x) = \exists$ (resp. $\Pi(x) = \forall$), we call x an existential (resp. universal) variable. If a variable x is in the set X_i, any $y \in X_j$ where $j < i$ ($j > i$), we say that y occurs to the left (right) of x in the quantifier prefix and write $y \leq_\Pi x$ ($y \geq_\Pi x$). The set of variables to the left of a universal variable u will be denoted by $L_\Pi(u)$ in this paper.

QBFs as a game: QBFs are often seen as a game between the universal and the existential player i.e. in the i^{th} step the player Q_i assigns Boolean values to the variables X_i. At the end, the existential (resp. universal) player wins if substituting this total assignment of variables in ϕ evaluates to 1 (resp. 0). For a QBF $\Pi.\phi$, a **strategy** of universal player is a decision function that returns the assignment to all universal variables of Π, where the

[1] In proof complexity, the following problem is of interest: Consider a set of polynomials $\mathcal{F}$ over some finite field. Determine if $\mathcal{F}$ have a common root. This is an important NP-complete problem [6].

decision for each u depends only on the variables in $L_\Pi(u)$. The **winning strategy** for the universal player is a strategy which for every possible assignment of existential variables, gives an assignment to all universal variables such that it falsifies the QBF. A QBF is false if and only if there exists a winning strategy for the universal player [4].

Decision Lists (DL): DLs [26] are a particular representation of functions (or strategies) computing the value of a single variable (say y), given the values of input variables (say $x_1, \ldots, x_n$). It is a sequential list of clauses of the form $(\bigwedge_{m\subset[n],j\in m} x_j/\overline{x_j} \to y/\overline{y})$ and the last clause being $(\mathsf{TRUE} \to y/\overline{y})$. This sequence is evaluated as an 'if, else-if, else' sequence of any programming language. That is, beginning from the first clause, the first time LHS of any clause is satisfied by the given input values, y gets assigned the value of the corresponding RHS. The length of a DL is defined as the number of such clauses in it.

Next, we define a few QBF proof systems that we require in this paper.

Q-Res: Q-Res [24] is an extension of the Resolution proof system for QBFs. It allows the resolution rule defined in Section 2 with the pivot variable being existential. For dealing with the universal variables, it defines a 'universal reduction' ($\forall$Red) rule which allows dropping of a universal variable u from a clause C, provided no existential variable $x \in C$ appears to the right of u.

QU-Res: QU-Res [30] is the Q-Res system which allows resolution step to be defined on universal pivot variables as well.

$\forall$Exp+Res: $\forall$Exp+Res [23] is an expansion based QBF proof system. It works by downloading axiom clauses while retaining only existential literals from input clauses. Each existential literal in a clause is annotated with an assignment to all universal variables in the left of it in the quantifier prefix. It also allows a resolution step (defined in Section 2) on these variables but only when the annotations of the pivot variable are the same in both clauses.

eFrege+$\forall$red: Frege systems are fundamental proof systems of propositional logic. Lines in a Frege proof are formulas inferred from the previous lines via few sound rules and a set of axioms, closed under substitution. The rules are not important as all Frege systems are p-equivalent, so w.l.o.g, we can assume that 'modus ponens' is the only rule in a Frege system. The modus ponens is defined as: if $A \to B$ and A are present in the hypothesis then B can be logically implied by the hypothesis. For a detailed definition and explanation refer [25].

Extended Frege (eFrege) [1] is an extension of Frege systems which allows introduction of new variables. This rule allows lines of the form $v \leftrightarrow f(S)$ where v is a new variable and f can be any formula on the set of variables S, where $v \notin S$. It is equivalent to Circuit Frege which operates as Frege does but with circuits in place of formulas.

For QBFs, eFrege is modified to be eFrege+$\forall$red (Extended Frege + $\forall$ reduction) [8] which is circuit Frege combined with a reduction rule for universal variables. The reduction rule allows a universal variable to take a line, which represents a circuit, where the rightmost variable appearing in the circuit is a universal variable and assign that universal variable to either 0 or to 1.

3 Understanding propositional NS with the twin variables encoding

In Section 3.1, we define the Nullstellensatz proof system with twin variables (NS$'$) and analyse it. Before this, as an antecedent we give an observation (Proposition 3) in propositional world regarding the NS proof system. NS is known to be incomparable to Resolution [11]. One direction of this is shown by propositional modulo 2 principle formulas which needs exponential-sized Resolution proofs [20] but has degree-1 NS proofs [11]. In the other

direction, a family of Horn-formulas known as the 'Induction principle' (Definition 7) needs $\Omega(log(n))$ degree NS-refutations but have linear-size Resolution proofs [11] (which are also tree-like). Therefore, NS and Res_T systems are also incomparable.

In [25], the author describes an inefficient but novel method for Polynomial Calculus(PC) to simulate the Resolution proof system. We use the same idea below to show a quadratic-size NS-simulation of $O(log(n))$-depth Res_T proofs of the Horn Formulas (Proposition 3).

▶ **Lemma 2.** *Given any Horn-clauses P and Q, derivation of the resolvent $res(P,Q)$ can be simulated as an NS proof using at most 2 monomials in each corresponding g_i for P and Q.*

Proof. Let $C_1 := (l \vee \overline{x_1} \vee \cdots \vee \overline{x_k})$ and $C_2 := (y \vee \overline{l} \vee \overline{z_1} \vee \cdots \vee \overline{z_{k'}})$ be two Horn clauses. Clearly, we have $res(C_1, C_2) := (y \vee \overline{x_1} \vee \cdots \vee \overline{x_k} \vee \overline{z_1} \vee \cdots \vee \overline{z_{k'}})$. Assume that $poly_{std}(C_1) := (1-l)x_1 \cdots x_k = 0$ and $poly_{std}(C_2) := (1-y)lz_1 \cdots z_{k'} = 0$ have already been derived in NS. We simulate this resolution step and derive $poly_{std}(res(C_1, C_2))$ as follows:

Multiply each of these equations with the literals of the other which are not present in the former clause and adding the products. That is, $poly_{std}(C_1) * (\mathbf{1-y})\mathbf{z_1} \cdots \mathbf{z_{k'}} + poly_{std}(C_2) * \mathbf{x_1} \cdots \mathbf{x_k} := \Big((1-y)x_1 \cdots x_k z_1 \cdots z_{k'} = 0\Big) := poly_{std}(res(C_1, C_2))$. The corresponding g_is are shown in bold. Note that the promise of at most 2 monomials in the g_is here, comes from the restriction of Horn-clauses to have at most one positive literal (in this case, $y \in C_2$). ◀

The method in Lemma 2 extends to further resolutions. Consider the following sequence.

$$\dfrac{\dfrac{C_1 \qquad C_2}{C_4 = res(C_1, C_2)} \qquad C_3}{C_5 = res(C_4, C_3)}$$

Let $P = 0, Q = 0$ and $R = 0$ be polynomial equations corresponding to input clauses C_1, C_2, C_3 and $S = 0, T = 0$ be polynomial equations corresponding to the derived clauses C_4, C_5. Hence, $P * g_1 + Q * g_2 = S$ and $S * g_3 + R * g_4 = T$ from Lemma 2. Now, rearranging in-terms of input polynomials implies: $P * \mathbf{g_1 g_3} + Q * \mathbf{g_2 g_3} + R * \mathbf{g_4} = T = poly_{std}(C_5)$. This observation leads to the following proposition.

▶ **Proposition 3.** *Any Horn-formula of $O(n)$ clauses, with a tree-like Resolution proof of depth d has an NS proof of size $O(n2^d)$.*

▶ **Proof Sketch.** *Let F be a Horn-formula with $O(n)$ clauses and $\pi := F \vdash_{Res_T} \perp$ with depth d. That is, when drawn as an inference tree G_π, every input clause can have at most d subsequent resolutions till the $\perp$ node. From Lemma 2, every resolution step can be simulated by at most 2 monomials as g_is. In-case of any such g_is for the subsequent resolutions, they need to be multiplied to the corresponding ancestral input clauses of the hypothesis. In this way, in the worst case we will have $O(2^d)$ monomials for some input clause. Therefore the size of the NS-proof will be at most $O(n2^d)$. The proof idea is depicted in the G_π graph below.*

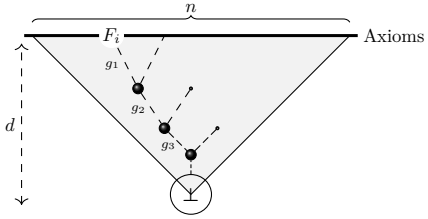

Figure 2 *Proposition 3 proof illustration.* Given a Horn formula F and its Res_T proof π, this figure shows the graph G_π. Here, g_1, g_2, g_3 are the g_is for the intermediate clauses according to Lemma 2. The accumulated g_i for the input clause F_i is $g_1 g_2 g_3$. In the worst case, any F_i can have g_i equal to $g_1 g_2 \cdots g_d$ and each of these have a maximum of 2 monomials, leading to the $O(2^d)$.

The above method can work in the general case of simulating any Resolution proof in NS (without the guarantee of linear sized monomials) as follows: wherever any internal node is used in 2 or more resolutions, the corresponding g_i for the node will be the addition of g_is from all its resolutions. This sum of g_is will then propagate to get multiplied to all the ancestral F_is of the node. This blowup in the number of monomials will be because of the loss of the tree-like structure.

Another reason for the blowup in number of monomials is the unbounded number of positive literals in each clause, which can lead to exponential-size blowup in g_is. Due to the same problem, PC non-efficiently simulates Resolution. To overcome this issue, a simple change in encoding called adding 'twin variables' lead to the proof system PCR (polynomial calculus with resolution). This system used a different notation to represent positive literals i.e. (x) was written as $x' = 0$ instead of $(1 - x) = 0$ and a new input clause $x + x' = 1$ was added for all variables to the system of equations.

3.1 NS′: Nullstellensatz with a twin variables encoding

▶ **Definition 4** (Twin variables Encoding [2]). *To convert a CNF $\mathcal{P}$ into a list of polynomial equations, we need to convert every clause into a separate polynomial equation. For every clause $C = (l_1 \vee \cdots \vee l_k \vee \overline{l_{k+1}} \vee \cdots \vee \overline{l_n})$, its polynomial equation is denoted by $\boldsymbol{poly}_{twin}(\mathbf{C}) := l'_1 \dots l'_k l_{k+1} \dots l_n = 0$. Along with introduction of $1 - x - x' = 0$ for all variables $x \in \mathcal{P}$. Notice that the set of assignments that satisfy C and $poly_{twin}(C)$ in both cases are exactly the same as by the nature of encoding we are writing $(1 - x)$ as x', so the additional clauses are true clauses.*

To maintain the Boolean nature of solutions, equations $x^2 - x = 0$ are introduced for all variables of $\mathcal{P}$. Given a CNF formula $\mathcal{P} = C_1 \wedge \cdots \wedge C_m$ over variables $x_1, \dots, x_n$. The corresponding ordered-list of equations is $\mathcal{F} := \{f_1 = 0, \dots, f_m = 0,\ f_{m+1} := 1 - x_1 - x'_1 = 0, \dots,\ f_{m+n} := 1 - x_n - x'_n = 0,\ f_{m+n+1} := x_1^2 - x_1 = 0, \dots, f_{m+2n} := x_n^2 - x_n = 0\}$ where the coefficients of the polynomials come from a fixed finite field $\mathbf{F}$. The proof system NS′ is defined below:

An NS′ refutation of $\mathcal{F}$ is a ordered-list of polynomials $(g_1, \dots, g_{m+2n})$ from the polynomial ring $\mathbf{F}[\bar{x}]$ such that the following holds: $\sum_{i \in [m+2n]} f_i(\bar{x}) g_i(\bar{x}) = 1$

Complexity measures: The complexity measures of an NS′ refutation are degree and size (same as in NS). Degree is defined as $\max_{i \in [m+2n]} (deg(f_i) + deg(g_i))$. The size of an NS′ refutation is the sum of number of monomials in all g_i and h_j.
The soundness of NS′ can be seen from the following lemma.

▶ **Lemma 5.** *PCR simulates NS′.*

Proof. PCR is a version of the Polynomial Calculus with the twin variable encoding. The simulation of NS′ by PCR is as straightforward as the simulation of NS by PC [25, p.145]. PCR allows the multiplication of a polynomial equation with a variable or with scalars in the field [2], and the addition of two polynomial equations. Hence the polynomial coefficients are built up using the rules of PCR and the total sum gives a contradiction.

Precisely, the simulation happens as follows:

- For every f_i, g_i, we use the monomial multiplication rule on f_i with monomials from g_i. The number of such steps = number of g_i monomials.
- Add all the above resultants with the addition rule. The number of such steps = number of monomials in g_is -1.

Therefore, the length of the PCR proof obtained by simulating an NS′ proof = 2(number of monomials in g_is) of the NS′ proof. ◀

We next show the completeness of NS′ by simulating Res_T in Lemma 6.

▶ **Lemma 6.** *NS′ p-simulates tree-like Resolution (Res_T).*

Proof. Consider the following resolution step:

$$\frac{(l \vee w \vee \overline{x}) \qquad (\overline{l} \vee y \vee \overline{z})}{(w \vee \overline{x} \vee y \vee \overline{z})}$$

NS′ can simulate this as: $\left(l'w'x\right) * \mathbf{y'z} + \left(ly'z\right) * \mathbf{w'x} + \left(1 - l - l'\right) * \mathbf{w'xy'z} := \left(w'xy'z\right)$.

Let $\mathcal{F}$ be any unsatisfiable propositional formula of n variables and m clauses, with a Res_T proof π where every variable is a pivot in at most k resolutions.

As NS′ can simulate a resolution step with 3 polynomials, π can now be viewed as a DAG-like Resolution proof where every step has 3 hypotheses, the third one being the newly added polynomial $(1 - l - l')$ of the pivot variable l. The resulting G_π is no longer a tree-like proof, but only the newly added polynomials $(1 - l - l')$ are used in more than 1 resolutions. Therefore when running the algorithm in Proposition 3, g_i for the rest of the input polynomials are a single monomial each. To extend the algorithm in Proposition 3 to the general case, if a node is used in 2 or more resolutions, the corresponding g_i for the node will be the addition of g_i from all its resolutions. Hence as the polynomials $(1 - l - l')$ are used in at most k resolutions, the corresponding g_is would have at most k monomials. Therefore the size of the resulting NS′ proof is $O(m + n * k)$. ◀

We end this section, by showing an easy NS′ refutation of the Induction principle.

▶ **Definition 7** (Induction principle formula (n)). $\phi = (x_1) \wedge \bigwedge_{i \in [n-1]} (\overline{x_i} \vee x_{i+1}) \wedge (\overline{x_n})$

NS′ refutation of the Induction principle formulas: Below is the NS′ refutation (f_is are shown inside parenthesis and g_is are shown in bold):

$(x'_1) * \mathbf{x'_2x'_3 \cdots x'_n} + (1 - x_1 - x'_1) * \mathbf{x'_2x'_3 \cdots x'_n} + (x_1x'_2) * \mathbf{x'_3x'_4 \cdots x'_n} + (1 - x_2 - x'_2) * \mathbf{x'_3x'_4 \cdots x'_n} + \cdots + (x_{n-1}x'_n) * \mathbf{1} + (1 - x_n - x'_n) * \mathbf{1} + (x_n) * \mathbf{1} \ = \ 1.$

Here every g_i is a single monomial, hence the size of the above NS′ proof is $2n + 1$ and the degree is n.

Observe that Proposition 3 gives an $O(n2^{n-1})$ sized NS refutation for the Induction principle formulas, as the depth of the corresponding Res_T proof is $n - 1$.

4 Expansion based NS systems for QBFs

The ∀Exp+Res [23] system works by downloading a subset of clauses from the complete Boolean expansion of the universal variables of the QBF and proving the subset to be propositionally false by resolution. In a similar approach, we introduce ∀Exp+NS (∀Exp+NS′) proof system which uses NS (NS′) instead of Resolution. The formal definition of these systems is given in the Definition 8.

▶ **Definition 8** (∀Exp+NS, ∀Exp+NS′). *Let $\Phi = \Pi.\phi$ be a false QBF with existential variables X and universal variables U. A ∀Exp+NS (∀Exp+NS′) refutation of Φ consists of the following steps:*

a. ***Axiom Downloads.*** *For a clause $C \in \phi$ and an assignment τ to variables in U, the i^{th}axiom clause A can be downloaded as follows:*

$$\frac{C \in \phi}{A_i := \ \{l^{[\tau]} \mid l \in C \ , \ var(l) \in X\} \cup \{\tau(u) \mid u \in C \ , \ var(u) \in U\}}$$

Sufficient axiom clauses are downloaded (say n) such that the resulting formula $\phi' := A_1 \wedge \cdots \wedge A_n$ is an UnSAT Boolean formula.

b. ***Encoding.*** *Consider the list $\mathcal{F} := (f_1, \ldots, f_n, f_{n+1} := r_1, \ldots, f_{n+m} := r_m)$ to be the standard encoding from Definition 1 (twin encoding from Definition 4) of CNF ϕ' in case of $\forall$Exp+NS ($\forall$Exp+NS$'$) system where the $r_1, \ldots, r_m$ are the Boolean constraints (Boolean constraints and twin constraints) of variables in ϕ'.*

c. ***Applying Nullstellensatz.*** *Find g_is for the set $\mathcal{F}$ which satisfy the condition that $\Sigma_{i \in [n+m]} f_i g_i = 1$.*

The final refutation of Φ will be the ordered list $(f_1, \ldots, f_{n+m}, g_1, \ldots, g_{n+m})$, written in standard encoding in case of $\forall$Exp+NS and twin encoding in case of $\forall$Exp+NS$'$. Note that these are static proof systems.

The $\forall$Exp+NS and $\forall$Exp+NS$'$ systems are sound and complete QBF-proof systems.

Completeness: Given any false QBF, a total Boolean expansion (exponential axiom downloads) will definitely be a false propositional formula. This in turn will have an NS-proof (NS$'$-proof) owing to the completeness of NS (NS$'$) system.

Soundness: Any QBF is equivalent to $\bigwedge$(exponential axiom downloads). If any subset of them have an NS-proof (or NS$'$-proof), it means that $\bigwedge$(subset of axioms) is a false propositional formula. In turn making the QBF definitely false.

Complexity measures: Degree of an $\forall$Exp+NS or $\forall$Exp+NS$'$ proof is defined as $\max_{i \in [m+n]} (deg(f_i) + deg(g_i))$. The size of an $\forall$Exp+NS or $\forall$Exp+NS$'$ proof is the sum of number of monomials in all f_i and g_i combined.

For a simple example of $\forall$Exp+NS and $\forall$Exp+NS$'$ proofs, let us consider the QBF:

$\Phi := \exists e_1, \forall u_1, u_2 \exists e_2 \ . \ (e_1 \vee u_1 \vee u_2 \vee e_2) \wedge (\overline{e_1} \vee \overline{u_1} \vee \overline{u_2} \vee e_2) \wedge (\overline{e_2})$.

This is a false formula as the universal player has the winning strategy to always set $u_1 = u_2 = e_1$. The $\forall$Exp+NS and $\forall$Exp+NS$'$ refutations of this formula are given below. The following axioms are downloaded in both cases: $A_1 := (e_1 \vee e_2^{u_1 \to 0, u_2 \to 0})$, $A_2 := (\overline{e_1} \vee e_2^{u_1 \to 1, u_2 \to 1})$, $A_3 := (\overline{e_2}^{u_1 \to 0, u_2 \to 0})$, $A_4 := (\overline{e_2}^{u_1 \to 1, u_2 \to 1})$. For simplicity let us rename the variable $e_2^{u_1 \to p, u_2 \to q}$ to be denoted as $e_{2,p,q}$.

$\forall$Exp+NS example proof: The standard encoding of these clauses gives the following f_i: $f_1 := (1 - e_1)(1 - e_{2,0,0})$, $f_2 := e_1(1 - e_{2,1,1})$, $f_3 := e_{2,0,0}$, $f_4 := e_{2,1,1}$. A possible list of g_i for these can be $(1, 1, (1 - e_1), e_1)$.

The following is easily checkable:

$$\Big((1 - e_1)(1 - e_{2,0,0})\Big) * \mathbf{1} \ + \ \Big(e_1(1 - e_{2,1,1})\Big) * \mathbf{1} \ + \ \Big(e_{2,0,0}\Big) * (\mathbf{1} - \mathbf{e_1}) \ + \ \Big(e_{2,1,1}\Big) * \mathbf{e_1} \ = \ 1.$$

Therefore the complete $\forall$Exp+NS proof presented in this case would be $(f_1, \ldots, f_4, g_1, \ldots, g_4)$ i.e. $((1 - e_1)(1 - e_{2,0,0}),\ e_1(1 - e_{2,1,1}),\ (e_{2,0,0}),\ (e_{2,1,1}),\ \ 1, 1, (1 - e_1), e_1)$.

Degree of this refutation is 2 and size is 13.

$\forall$Exp+NS$'$ example proof: The twin encoding of the above clauses gives the following f_is: $f_1 := e_1' e_{2,0,0}'$, $f_2 := e_1 e_{2,1,1}'$, $f_3 := e_{2,0,0}$, $f_4 := e_{2,1,1}$, $f_5 := r_1 := 1 - e_1 - e_1'$, $f_6 := r_2 := 1 - e_{2,0,0} - e_{2,0,0}'$, $f_7 := r_3 := 1 - e_{2,1,1} - e_{2,1,1}'$. A possible list of g_is for these can be $(e_{2,1,1}',\ e_{2,0,0}',\ e_{2,1,1}',\ 1,\ e_{2,0,0}' e_{2,1,1}',\ e_{2,1,1}',\ 1)$.

The following is easily checkable:
$\left(e'_1e'_{2,0,0}\right) * \mathbf{e'_{2,1,1}} + \left(e_1e'_{2,1,1}\right) * \mathbf{e'_{2,0,0}} + \left(e_{2,0,0}\right) * \mathbf{e'_{2,1,1}} + \left(e_{2,1,1}\right) * \mathbf{1} + \left(1-e_1-e'_1\right) * \mathbf{e'_{2,0,0}e'_{2,1,1}} + \left(1-e_{2,0,0}-e'_{2,0,0}\right) * \mathbf{e'_{2,1,1}} + \left(1-e_{2,1,1}-e'_{2,1,1}\right) * \mathbf{1} = 1.$

Therefore the complete ∀Exp+NS′ proof presented in this case would be $(f_1, \dots g_1, \dots)$ i.e. $(e'_1e'_{2,0,0},\ e_1e'_{2,1,1},\ e_{2,0,0},\ e_{2,1,1},\ 1-e_1-e'_1, 1-e_{2,0,0}-e'_{2,0,0}, 1-e_{2,1,1}-e'_{2,1,1},\ e'_{2,1,1},\ e'_{2,0,0},\ e'_{2,1,1},\ 1,\ e'_{2,0,0}e'_{2,1,1},\ e'_{2,1,1},\ 1)$.

The degree of this refutation is 3 and size is 20.

5 Strategy based QBF-proof systems

Instead of expansion to handle the universal variables, one can force the universal player to play according to its winning strategy. This can be done by adding strategy clauses to the QBF. We should be careful that the addition of such clauses should not change the satisfiability of the QBF. To be precise, adding strategy-clauses to the matrix of a true QBF, should not give a false propositional formula. However adding arbitrary clauses to a CNF can potentially falsify the CNF. This is formally stated in the following observation.

▶ **Observation 9.** *Consider a QBF $\Pi \cdot \phi$ that we aim to add strategy clauses to. Take the CNF S that consists of all strategy clauses and all clauses for relevant extension variables. We consider α and β as assignments on the existential variables (excluding extension variables).*

- *For every α, $S|_\alpha$ is propositionally satisfiable.*
- *For every α, β, if there is an existential variable x such that α and β differ on x then $S|_\alpha \vDash u$ implies $S|_\beta \vDash u$, when x is right of u in the prefix.*

If S satisfies the above conditions, then the following always holds: If $\Pi \cdot \phi$ is true, then $\phi \wedge S$ is satisfiable.

Proof. Starting with a true QBF $\Pi \cdot \phi$, the existential player has a winning strategy (say $S_\exists$). S from the above definition is a partial strategy for the universal player, this can be extended to a full strategy S' (not a winning strategy). Note that $\phi \wedge S'$ is indeed satisfiable: construct the satisfying assignment according to $S_\exists$ for the existential variables and S' for the universal variables, this should satisfy ϕ by definition of the winning strategy. In the above definition any existential assignment α extended with the appropriate universal assignments should satisfy $S'|_\alpha$. Therefore this assignment will satisfy $\phi \wedge S'$ as well.

◀

Using Observation 9, we now define a ∀Strat framework which would be useful for lifting both static and dynamic propositional proof systems to QBFs.

▶ **Definition 10 (∀Strat Framework).** *Given a false QBF $\Pi.\phi$ and a winning ∀ strategy s_j for each universal variable u_j, we can confirm the strategy is indeed winning with a propositional refutation system P and an encoding of $\phi \wedge (u_j \leftrightarrow s_j)$. Therefore we can define a family of QBF refutation systems ∀Strat$_E$+P, for a specified strategy encoding E (where the properties in Observation 9 are polynomial time checkable) and a propositional refutation system P, that confirms the winning strategy with a proof P.*

- *∀Strat$_{Circ}$: For each universal variable u we have an extension variable s_u (possibly defined using more extension variables, all of which must only depend on variables left of u) such that $(\bar{u}, s_u)$ and $(\bar{s}_u, u)$ are added as clauses.*
- *∀Strat$_{DL}$: We introduce extension variables $t_i^j = C$ for clauses only. The strategy of $u_j \leftrightarrow s_j$ is represented by a series of clauses of the form $t_i^j \vee \bar{t}_{i-1}^j \vee \dots \vee \bar{t}_1^j \vee$ lit$_i(u_j)$.*

Where $\mathit{lit}_i(u_j)$ is either u_j or $\bar{u}_j$ and all t_i^j are defined only using variables left of u_j. $t^j_{i_{max}} = \bot$. A verifier algorithm for this is given as Appendix: Algorithm 3.

▶ **Observation 11.** *$\forall Strat_{Circ}$+eFrege $\equiv_p$ eFrege+$\forall$red.*

Proof. ($\leq_p$) Given a $\forall$Strat$_{Circ}$+eFrege proof of $\Pi.\phi$ we extract an eFrege refutation of $\phi \wedge \bigwedge_{i=1}^{n}(u_i \leftrightarrow s_i) \wedge \bigwedge_{i=1}^{n} Def_{s_u}$, where Def_{s_u} contains extension clauses that define s_u (and other intermediate extension variables). eFrege is simulated by Circuit Frege, so we can replace the extension variables with circuits for a proof of $\neg(\phi \wedge \bigwedge_{i=1}^{n}(u_i \leftrightarrow \sigma_i))$.

We can continue this as an eFrege+$\forall$red refutation. Downloading all ϕ axioms we get $\bigvee_{i=1}^{n}(u_i \nleftrightarrow \sigma_i)$. The rightmost universal variable u_n is now no longer blocked by any existential variables. Using the reduction rule (see [8]) we can reduce u_n in both 0 and 1, and since σ_n is a Boolean function that must be true or false we get $\bigvee_{i=1}^{n-1}(u_i \nleftrightarrow \sigma_i)$. This can be repeated until arriving at the empty disjunction.

($\geq_p$) Given an eFrege+$\forall$red proof, we put it into normal form (see [8]) with the first half being a proof of $\neg(\phi \wedge \bigwedge_{i=1}^{n}(u_i \leftrightarrow \sigma_i))$ in circuit Frege, we then use the simulation of circuit Frege by eFrege to get an eFrege refutation of $\phi \wedge \bigwedge_{i=1}^{n}(u_i \leftrightarrow s_i) \wedge Def_{s_u}$. ◀

▶ **Observation 12.** *$\forall Strat_{DL}$+Resolution $\geq_p$ QU-Resolution.*

Proof. We derive the decision list by the strategy extraction method (see [8])

We will show by induction that we can with a short $\forall$Strat$_{DL}$+Resolution obtain.

a For each universal reduction line $\frac{C \vee u}{C}$ prove $C \vee \bar{u}$ is derived before C can be derived.

b For each line D in the QU-Res proof we derive D.

Base Case(s): a) the first DL clause of the decision list for u is $t_1^u \vee lit(u)$ and t_1^u can be resolved with its long clause. b) all axioms are available in $\forall$Strat$_{DL}$+Resolution.

Inductive Step: a) We start with the DL clause $t_j^u \vee \bar{t}_{j-1}^u \ldots \bar{t}_1^u \vee lit(u)$ and remove each negative t literal. Singleton t_i^u, for $i < j$ can be derived from taking the clause it is equivalent to, which is derived as part of a previous reduction step, and resolving it with all short clauses for t_i^u, eventually deriving the unit clause t_i^u. Then we can derive $t_j^u \vee lit(u)$ b) We mimic resolution steps exactly, for universal reduction steps we resolve $C \vee u$ with $C \vee lit(u)$, here $lit(u)$ will be $\neg u$ by construction.

◀

In the main part of the paper, because we want to work as close as possible to QCDCL (Quantified conflict-driven clause learning) systems where strategy extraction can be output as a decision list we will use $\forall$Strat to denote $\forall$Strat$_{DL}$ and drop the subscript.

5.1 Strategy based NS system for QBFs: $\forall$Strat+NS

In this section, we define NS based system for QBFs using the $\forall$Strat$_{DL}$ framework defined in Definition 10. Since, NS works on polynomial equations, we need to encode QBFs as a list of Q-polynomial equations. We use standard encoding (Definition 1) for the same.

Converting a QBF into standard Q-polynomial equations Given a QBF $\Pi.\phi$, for every clause $C_i \in \phi$, we have an Q-polynomial as
$F_i :=$ Π. $poly_{std}(C_i) = 0$. For instance, see the following example.

▶ **Example 13.** False QBF $\Pi.\phi := \exists e_1\ \forall u\ \exists e_2\ .\ (\overline{e_1} \vee \overline{u}) \wedge (e_1 \vee e_2) \wedge (u \vee \overline{e_2}) \wedge (\overline{e_1} \vee \overline{e_2}) \wedge (\overline{u} \vee e_2)$
Then the standard Q-polynomial $\mathcal{F}$ consists of the following:

$F_1 := \exists e_1\ \forall u\ \exists e_2\ .\ e_1.u = 0$; $\quad F_2 := \exists e_1\ \forall u\ \exists e_2\ .\ (1 - e_1).(1 - e_2) = 0$;

$F_3 := \exists e_1\ \forall u\ \exists e_2\ .\ (1 - u).e_2 = 0$; $\quad F_4 := \exists e_1\ \forall u\ \exists e_2\ .\ e_1.e_2 = 0$;

$F_5 := \exists e_1\ \forall u\ \exists e_2\ .\ u.(1-e_2) = 0$;

In addition, there are the following Boolean axioms:

$F_6 := \exists e_1\ \forall u\ \exists e_2\ .\ e_1^2 - e_1 = 0$; $F_7 := \exists e_1\ \forall u\ \exists e_2\ .\ u^2 - u = 0$;

$F_8 := \exists e_1\ \forall u\ \exists e_2\ .\ e_2^2 - e_2 = 0$;

Given a set of Q-polynomials $\{F_i\text{s}\}$ (all with the same quantifier prefix), their truth value is evaluated as a 2-player game similar to QBFs. That is, the set of Q-polynomials are false if the universal player (when playing in the quantification sequence) can always falsify at least one F_i. Observe that when converting a QBF $\Pi.\phi$ to $\{F_i\text{s}\}$, the satisfiability is preserved as "$\Pi.\phi$ is false if and only if the universal player always falsifies at least one equation from F_is."

In the above Example 13, the universal player setting $u = e_1$ will win the game.

Now, we are ready to define the proposed static ∀Strat+NS proof system.

▶ **Definition 14** (∀Strat+NS proof system)**.** *Let $\mathcal{P} := \Pi.(F_1 = 0, F_2 = 0, \ldots, F_n = 0)$ be a list of standard Q-polynomials. A ∀Strat+NS refutation of $\mathcal{P}$ is a combination of 3 ordered-lists of polynomials: $(g_1, \ldots, g_n)$, $(poly_{std}(S_1), \ldots, poly_{std}(S_m))$, $(h_1, \ldots h_m)$ such that the following is satisfied:*

$$\sum_{i\in[n]} F_i g_i + \sum_{i\in[m]} poly_{std}(S_i)h_i = 1 \tag{1}$$

where $poly_{std}(S_i)$ are the standard representations (Definition 1) of CNF versions of the following types of clauses:

- *a Decision list of length ℓ of strategy-clauses for all universal variables $u_j \in \Pi$, where the i^{th} clause has the format as $(\overline{t_i^{u_j}} \wedge t_{i-1}^{u_j} \wedge \cdots \wedge t_1^{u_j}) \rightarrow u_j/\overline{u_j}$ and the last clause has the format as $(t_{\ell-1}^{u_j} \wedge t_{\ell-2}^{u_j} \wedge \cdots \wedge t_1^{u_j}) \rightarrow u_j/\overline{u_j}$ Here the t_i^u variables are new Tseitin variables (defined below) used to represent each decision line for verifiability of proof. Note that no other new variables are allowed in the proof.*
- *b Definition clauses of the new Tseitin variables in-terms of existential variables from $L_\Pi(u_j)$. That is every $t_i^{u_j}$ is introduced by the clause $(\overline{t_i^{u_j}} \leftrightarrow (l_1 \wedge l_2 \wedge \ldots))$ where $var(l_1), var(l_2), \ldots \in L_\Pi(u_j)$.*

Note that each $poly_{std}(S_i)$ also contain Boolean axioms: $x^2 - x = 0$ for the new Tseitin variables.

Complexity measures:

- Degree is defined as $max((\underset{i\in[n]}{max}(deg(F_i) + deg(g_i))), (\underset{i\in[m]}{max}(deg(poly_{std}(S_i)) + deg(h_i))))$.
- The size being the number of all monomials in g_is, $poly_{std}(S_i)$ and h_is combined.

For instance, below we give the ∀Strat+NS proof of Q-polynomials from Example 13.

▶ **Example 15.** Consider the Q-polynomials $\mathcal{F}$ from Example 13 to be the input list $\mathcal{F}$.

Below is the winning strategy for u as a decision list of length 2:

$if(e_1) \rightarrow u$

$else \rightarrow \overline{u}$

The new Tseitin variable t_1^u is defined as: $(\overline{t_1} \leftrightarrow e_1)$. That is, $S_1 = (t_1 \vee e_1), S_2 = (\overline{t_1} \vee \overline{e_1})$. Note that since there is only one universal variable u, we drop the superscript u from t_1^u for simplicity.

The decision list clauses are accordingly: $S_3 := (\overline{t_1} \rightarrow u), S_4 := (t_1 \rightarrow \overline{u})$

The standard encoding of these polynomials is as follows:

$poly_{std}(S_1) := (1-t_1).(1-e_1) = 0$; $poly_{std}(S_2) := t_1.e_1 = 0$;

$poly_{std}(S_3) := (1-t_1).(1-u) = 0$; $poly_{std}(S_4) := t_1.u = 0$;

A possible list of g_is for these can be $(2-2t_1, 1, 1, -1, -1, 0, 0, 0,\ \ u, 1, e_1, 1)$. The following is easily checkable:

$$\Big(e_1u\Big)*(\mathbf{2-2t_1}) + \Big((1-e_1)(1-e_2)\Big)*\mathbf{1} + \Big((1-u)e_2\Big)*\mathbf{1} + \Big(e_1e_2\Big)*\mathbf{-1} + \Big(u(1-e_2)\Big)*\mathbf{-1} + \Big(e_1^2-e_1\Big)*\mathbf{0} + \Big(u^2-u\Big)*\mathbf{0} + \Big(e_2^2-e_2\Big)*\mathbf{0} + \Big((1-t_1)(1-e_1)\Big)*\mathbf{u} + \Big(t_1e_1\Big)*\mathbf{1} + \Big((1-t_1)(1-u)\Big)*\mathbf{e_1} + \Big(t_1u\Big)*\mathbf{1} = 1.$$

Therefore the complete ∀Strat+NS proof presented in this case would be (g_is), ($poly_{std}(S_i)$s), (h_is) i.e. $\Big(2-2t_1, 1, 1, -1, -1, 0, 0, 0\Big)$, $\Big((1-t_1)(1-e_1), t_1e_1, (1-t_1)(1-u), t_1u\Big)$, $\Big(u, 1, e_1, 1\Big)$.

Degree of this refutation is 3 and size is 20.

The ∀Strat+NS system is a sound and complete QBF proof system.

▶ **Theorem 16** (Soundness). *Let $\pi = (g_1, \ldots, g_n)$, $(poly_{std}(S_1), \ldots, poly_{std}(S_m))$, $(h_1, \ldots, h_m)$ be a ∀Strat+NS refutation of a list of Q-polynomials $\Pi.\{F_i = 0\}_{i=1}^n$. Then, the set $\Pi.\{F_i = 0\}_{i=1}^n$ is false.*

Proof. Given a valid ∀Strat+NS refutation $\pi = (g_1, \ldots, g_n), (poly_{std}(S_1), \ldots, poly_{std}(S_m))$, $(h_1, \ldots, h_m)$. It implies that $S_1, \ldots, S_m$ are derived from decision lists computing assignments for universal variables in Π. These additional clauses are either definition clauses for new Tseitin variables or decision list clauses using these variables.

The definition clauses are CNF versions of the clauses $(\overline{t_i} \leftrightarrow \mathsf{DNF}(\bar{x}))$, these extra clauses are all satisfiable clauses (i.e. evaluate to true). Also, the decision clauses are careful not to introduce any contradictions. That is, there is no assignment α to $L_\Pi(u)$ which can satisfy more than one decision clause. This is because every new clause has the negation of LHS of all previous clauses in it. Hence, if there exists a π which is satisfying Equation 1 for these F_is ∪ $poly(S_j)$s, it implies that $poly(S_j)$s are the winning strategy of the universal player and therefore implies that the input Q-polynomials $\Pi.\{F_i = 0\}^n$ are indeed false owing to the soundness of the Nullstellensatz system. ◀

▶ **Theorem 17** (Completeness). *For every false set of Q-polynomials $\Pi.\{F_i\}$s, there exists at least one ∀Strat+NS refutation.*

Proof. The evaluation of a set of Q-polynomials is done as a 2-player game. Implying that a false set of Q-polynomials should definitely have a winning strategy for the universal player. Every Boolean function can be represented as a decision function [3]. Therefore adding this winning strategy as decision and definition clauses as defined in Definition 14 (i.e. F_is ∪ $poly(S_j)$s) will make a set of false polynomial equations owing to Observation 9. Now drawing on the completeness of the Nullstellensatz proof system for a set of polynomial equations, there exists a set of g_i and h_i that satisfies Equation 1. ◀

A proof system by definition should be sound, complete and polynomial time verifiable [1]. In order to show that ∀Strat+NS is easily verifiable, one must show that the winning strategy encoded as decision list is easily verifiable. We introduced the Tseitin variables in the encoding, solely for this purpose. For the detailed verification algorithm, see Appendix: Algorithm 1. As an example, next let us see how the proof from Example 15 can be verified as a correct ∀Strat+NS proof.

▶ **Example 18.** Consider the Q-polynomials $\mathcal{F}$ and the final ∀Strat+NS proof presented in Example 15 as input. We follow the following steps in-order to validate the same and reject it if any of the following checks fail.

1. First, we ***check*** that Equation 1 is satisfied.

2. We then convert the $poly_{std}(S_i)$s into clauses (ignoring any Boolean axioms present) as follows:
   - if any literal is present in all monomials, it is a negative literal in the clause.
   - Remove the common literals found above from all monomials in the polynomial.
   - all single degree monomials with coefficient of '-1' are positive literals of the clause.
   - lastly, ***check*** if the standard encoding of accumulated positive and negative literals gives back the original $poly_{std}(S_i)$.

   For illustration of the above Step 2, see Example 32 in the Appendix.
3. We perform the following Steps 3-7 for all universal variables in the prefix Π. Recall that u is the only universal variable in this example. We find the shortest width clause containing $\mathsf{var}(u)$. If this is a singleton clause, that would be some trivial (0/1) strategy for u. To handle such a case see [Algorithm 3, lines 9-11]. This is not the case for this example. ***Check*** if the shortest width clause containing u contains a positive literal t. In this example, it is found as $(t_1 \vee u)$, mark it as checked. Intuitively, every such clause is a new line in the potential decision list.
4. Next, we find the longest width clause containing $\overline{t_1}$ (the Tseitin variable found in Step 3) and not containing $var(u)$. If not found reject. In this example, it is $(\overline{t_1} \vee \overline{e_1})$, ***check*** if e_1 is in $L_\Pi(u)$ and mark the clause as checked.
5. For every extra literal l in the above clause ($\overline{e_1}$ in this example) we ***check*** that a clause with $\bar{l}$ and positive t_1 is present. In this example, it is $(t_1 \vee e_1)$ mark it as checked. Intuitively, clauses from Step 4,5 are the definition clauses corresponding to every Tseitin variable.
6. We then find the next smallest width clause with $var(u)$ and check if there is an unhandled positive t_i literal. If yes, we repeat Steps 4,5,6 again.
7. We do not have any more t_i variables in this example, so the next smallest clause containing $var(u)$ is $(\overline{t_1} \vee \overline{u})$. ***Check*** whether all t_i variables in this clause are handled and then mark it as checked. Intuitively, this is the last line of the potential decision list.
8. Finally, if all clauses in $poly_{std}(S_i)$ are marked as checked, then accept otherwise reject. In this example, all 4 clauses are marked. Therefore we have a valid $\forall$Strat+NS proof.

5.2 Strategy based NS′ system for QBFs: $\forall$Strat+NS′

In this section, we define an NS′ based system for QBFs using the $\forall\text{Strat}_{DL}$ framework defined in Definition 10. Since, NS′ works on polynomial equations, we need to encode QBFs as a list of Q-polynomial equations. We use the twin encoding (Definition 4) for the same.

Converting a QBF into twin Q-polynomial equations Given a QBF $\Pi.\phi$, for every clause $C_i \in \phi$, we have an Q-polynomial as $F_i := \Pi'.\ poly_{\text{twin}}(C_i) = 0$. Here the quantified prefix Π is changed to Π' to include the twin variables. Twin variables are always quantified existentially to the right of the original variables. Recall that in the twin encoding, we also include polynomials $(1 - x - x')$ for every variable x in the input polynomials. For instance, the same QBF from Example 13 can be encoded as twin Q-polynomials as shown below.

▶ **Example 19.** False QBF $\Pi.\phi := \exists e_1\ \forall u\ \exists e_2\ .\ (\overline{e_1} \vee \overline{u}) \wedge (e_1 \vee e_2) \wedge (u \vee \overline{e_2}) \wedge (\overline{e_1} \vee \overline{e_2}) \wedge (\overline{u} \vee e_2)$
Then the twin Q-polynomial F_is are the following:

$F_1 := \exists e_1\ \forall u\ \exists e_2\ .\ e_1.u = 0$; $F_2 := \exists e_1\ \forall u\ \exists e_2\ .\ e'_1.e'_2 = 0$;
$F_3 := \exists e_1\ \forall u\ \exists e_2\ .\ u'.e_2 = 0$; $F_4 := \exists e_1\ \forall u\ \exists e_2\ .\ e_1.e_2 = 0$;
$F_5 := \exists e_1\ \forall u\ \exists e_2\ .\ u.e'_2 = 0$;

Twin axioms are as follows:

$F_6 := \exists e_1\ \forall u\ \exists e_2\ .\ 1 - e_1 - e'_1 = 0$; $F_7 := \exists e_1\ \forall u\ \exists e_2\ .\ 1 - u - u' = 0$;

$F_8 := \exists e_1\ \forall u\ \exists e_2\ .\ 1 - e_2 - e'_2 = 0$;
In addition, there are the following Boolean axioms:
$F_9 := \exists e_1\ \forall u\ \exists e_2\ .\ e_1^2 - e_1 = 0$; $F_{10} := \exists e_1\ \forall u\ \exists e_2\ .\ u^2 - u = 0$;
$F_{11} := \exists e_1\ \forall u\ \exists e_2\ .\ e_2^2 - e_2 = 0$;

The truth value of a set of twin Q-polynomials are evaluated exactly as for standard polynomials described in Section 5.1. However, here the satisfiability is kept intact by making the twin variable of the universal player always be existential and to play after the universal player does. This way, the newly added polynomials $F_6, \ldots, F_8$ always evaluate to true and do not interfere in the satisfiability of the input QBF.
Now, we are ready to define the proposed static ∀Strat+NS′ proof system.

▶ **Definition 20** (∀Strat+NS′ proof system). *Let $\mathcal{P} := \Pi.(F_1 = 0, F_2 = 0, \ldots, F_n = 0)$ be a list of twin Q-polynomials. A ∀Strat+NS′ refutation of $\mathcal{P}$ is a combination of 3 ordered-lists of polynomials: $(g_1, \ldots, g_n)$, $(poly_{twin}(S_1), \ldots, poly_{twin}(S_m))$, $(h_1, \ldots h_m)$ such that the following is satisfied:*

$$\sum_{i\in[n]} F_i g_i + \sum_{i\in[m]} poly_{twin}(S_i) h_i = 1 \tag{2}$$

where $poly_{twin}(S_i)$s are the twin representations (Definition 4) of CNF versions of the strategy-clauses as defined in Definition 14 and the twin axioms of the new Tseitin variables.

Complexity measures are the same as defined for ∀Strat+NS in Section 5.1.
Soundness and Completeness: Since, the changes in the twin encoding i.e. addition of new clauses and variables does not change the satisfiability of the input QBF. The soundness and completeness of the ∀Strat+NS′ system follows analogously from Theorem 16 and Theorem 17 of ∀Strat+NS.
For instance, below we give the ∀Strat+NS′ proof of Q-polynomials from Example 19.

▶ **Example 21.** Consider the Q-polynomials from Example 19 to be the input list $\mathcal{F}$.
Below is the winning strategy for u as a decision list of length 2:
if$(e_1) \to u$
else $\to \overline{u}$
The new Tseitin variable is defined as: $(\overline{t_1}^u \leftrightarrow e_1)$. That is, $S_1 = (t_1 \vee e_1), S_2 = (\overline{t_1} \vee \overline{e_1}), S_3 = (1 - t_1 - t'_1)$.
Hence the decision list clauses are accordingly: $S_4 := (\overline{t}_1 \to u), S_5 := (t_1 \to \overline{u})$
The twin encoding of these polynomials is as follows:
$poly_{twin}(S'_1) := t'_1.e'_1 = 0$; $poly_{twin}(S'_2) := t_1.e_1 = 0$;
$poly_{twin}(S'_3) := 1 - t_1 - t'_1 = 0$; $poly_{twin}(S'_4) := t'_1.u' = 0$;
$poly_{twin}(S'_5) := t_1.u = 0$;
A possible list of g_i and h_i polynomials for these can be $(0, t_1, t_1, t'_1, t'_1,\ \ (e'_2 t_1 + e_2 t'_1), (e'_2 t'_1 + e_2 t_1), (t'_1 + t_1), 0, 0, 0,\ \ e_2, e'_2, 1, e'_2, e_2)$. The following is easily checkable:
$\Big(e_1 u\Big)*\mathbf{0} + \Big(e'_1 e'_2\Big)*\mathbf{t_1} + \Big(u' e_2\Big)*\mathbf{t_1} + \Big(e_1 e_2\Big)*\mathbf{t'_1} + \Big(u e'_2\Big)*\mathbf{t'_1} + \Big(1 - e_1 - e'_1\Big)*(\mathbf{e'_2 t_1} + \mathbf{e_2 t'_1})$
$+ \Big(1 - u - u'\Big) * (\mathbf{e'_2 t'_1} + \mathbf{e_2 t_1}) + \Big(1 - e_2 - e'_2\Big) * (\mathbf{t'_1} + \mathbf{t_1}) + \Big(e_1^2 - e_1\Big) * \mathbf{0} + \Big(u^2 - u\Big) * \mathbf{0}$
$+ \Big(e_2^2 - e_2\Big)*\mathbf{0} + \Big(t'_1 e'_1\Big)*\mathbf{e_2} + \Big(t_1 e_1\Big)*\mathbf{e'_2} + \Big(1 - t_1 - t'_1\Big)*\mathbf{1} + \Big(t'_1 u'\Big)*\mathbf{e'_2} + \Big(t_1 u\Big)*\mathbf{e_2} = 1.$
Therefore the complete ∀Strat+NS′ proof presented in this case would be $(g_i$s$), (poly_{twin}(S'_i)$s$), (h_i$s$)$
i.e. $\Big(0, t_1, t_1, t'_1, t'_1,\ \ (e'_2 t_1 + e_2 t'_1), (e'_2 t'_1 + e_2 t_1), (t'_1 + t_1), 0, 0, 0\Big),\ \ \Big(t'_1 e'_1, t_1 e_1, 1 - t_1 - t'_1, t'_1 u', t_1 u\Big),\ \ \Big(e_2, e'_2, 1, e'_2, e_2\Big)$.
The degree of this refutation is 3 and size is 22.

∀Strat+NS′ is a polynomial time verifiable static proof system. For the detailed verification algorithm, see Appendix: Algorithm 2.

6 Strengths and limitations of proposed systems

In this section, we analyze the strength and limitations of the proposed proof systems among themselves and against some important QBF proof systems. We first show that ∀Exp+NS′ and ∀Strat+NS′ are incomparable (Theorem 22). The proof follows from Lemmas 24 and 28.

▶ **Theorem 22.** *∀Exp+NS′ and ∀Strat+NS′ are incomparable.*

Consider the family of QBFs QPARITY$_n$ from [10, Theorem 14] restated below.

▶ **Definition 23** (QPARITY$_n$ formulas ($\bigoplus_n$) [10]). $\exists x_1, \ldots, x_n\ \forall z\ \exists t_2, \ldots, t_n$
$xor(x_1, x_2, t_2) \wedge \Big(\bigwedge_{i \in [3,n]} xor(t_{i-1}, x_i, t_i) \Big) \wedge (z \vee t_n) \wedge (\overline{z} \vee \overline{t_n})$
where, $xor(o_1, o_2, o)$ *sets* $o = o_1 \oplus o_2$ *by the clauses:*
$(\overline{o_1} \vee \overline{o_2} \vee \overline{o}) \wedge (\overline{o_1} \vee o_2 \vee o) \wedge (o_1 \vee \overline{o_2} \vee o) \wedge (o_1 \vee o_2 \vee \overline{o})$

▶ **Lemma 24.** *∀Strat+NS′ cannot p-simulate ∀Exp+NS′.*

Proof. QPARITY$_n$ (Definition 23) is hard for ∀Strat+NS′ from Theorem 25. On the other hand, Lemma 26 below proves that QPARITY$_n$ has easy proofs in ∀Exp+NS′. ◀

▶ **Theorem 25.** *QPARITY$_n$ formulas ($\bigoplus_n$) have exponential sized ∀Strat+NS and ∀Strat+NS′ refutations.*

▶ **Proof Sketch.** *Observe that the size of ∀Strat+NS (∀Strat+NS′) refutations also counts the strategy clauses. Therefore an obvious lower-bound in these systems can be from those QBFs where the winning strategy needs exponential number of clauses to be represented as a decision list. The winning strategy for $\bigoplus_n$ is to set $z = t_n$ i.e. $z = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ (in terms of variables left of z). It is well known that if one can represent a Boolean function f as a polynomial-size decision list then $f \in AC^0$ [10, Lemma 13]. On the other hand, we know that the Boolean function 'parity' of n-variables does not belong to AC^0 [17]. Therefore, the decision list representing the winning strategy of $z = PARITY(x_1, \ldots, x_n)$ must be of exponential size in n.*

▶ **Lemma 26.** *QPARITY$_n$ has linear size ∀Exp+NS′ proofs.*

Proof. In [9, Theorem 2], the authors gave a polynomial size ∀Exp+Res$_T$ proof π of QPARITY$_n$. In π, the clauses are expanded with $0, 1$ values of z, doubling the total number of clauses. We can easily convert π into an ∀Exp+NS′ proof π': derive the same expansion clauses as axioms in π' and Res$_T$ proof can be converted into NS′ proof by Lemma 6. In π, every variable is resolved at most constant (i.e. 8) times. Therefore size(π') is linear in n. ◀

Consider the family of QBFs defined by Janota and Marques-Silva from [23, Formula 2] restated below. (The original paper uses notation ϕ_n, but here we refer them as JM$_n$).

▶ **Definition 27** (Janota and Marques-Silva (JM$_n$) formulas [23]).
$\exists e_1 \forall u_1 \exists c_1, c_2 \ldots \exists e_n \forall u_n \exists c_{2n-1}, c_{2n}$
$\bigwedge_{i \in [n]} \Big\{ (\overline{e_i} \vee c_{2i-1}) \wedge (\overline{u_i} \vee c_{2i-1}) \wedge (e_i \vee c_{2i}) \wedge (u_i \vee c_{2i}) \Big\} \wedge \big(\bigvee_{i \in [2n]} \overline{c_i} \big)$

▶ **Lemma 28.** *∀Exp+NS′ cannot p-simulate ∀Strat+NS′.*

Proof. JM_n is hard for $\forall$Exp+NS$'$ from Proposition 29. On the other hand, Theorem 30 below proves that JM_n has easy proofs in $\forall$Strat+NS$'$. ◀

▶ **Proposition 29.** *JM_n needs exponential size proofs in the $\forall$Exp+NS and $\forall$Exp+NS$'$ systems.*

▶ **Proof Sketch.** *The size of an $\forall$Exp+NS and $\forall$Exp+NS$'$ proof also includes f_i polynomials . Hence, one type of obvious lower bounds in these systems are all formulas where exponential axiom downloads are required to maintain the falsity of the formula. The authors in [23, Proposition 3] prove that JM_n formulas need exponential axiom downloads as every sub-formula in any expansion is satisfiable.*

▶ **Theorem 30.** *JM_n formulas have linear sized $\forall$Strat+NS$'$ proofs.*

Proof. The winning strategy for universal variables in JM_n formulas (Definition 27) is to set all $u_i = \overline{e_i}$. As a decision list:

$if(e_i) \to \overline{u_i}$

$else \to u_i$

The Tseitin definition clauses would be $\bigwedge_{i\in[n]} (\overline{t_i} \leftrightarrow e_i)$ and the decision list clauses would be $\bigwedge_{i\in[n]} (t_i \vee \overline{u_i}) \wedge (\overline{t_i} \vee u_i)$. Now, let JM'_n be the resulting formulas after adding these clauses to the definition clauses of JM_n from Definition 27. JM'_n has easy Res_T proof as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{(t_i \vee e_i)\quad(\overline{t_i} \vee u_i)}{(\overline{e_i} \vee c_{2i-1})\qquad(e_i \vee u_i)}}{(\overline{u_i} \vee c_{2i-1})\qquad(u_i \vee c_{2i-1})}}{(c_{2i-1})} \qquad \dfrac{\dfrac{\dfrac{\dfrac{(\overline{t_i} \vee \overline{e_i})\quad(t_i \vee \overline{u_i})}{(e_i \vee c_{2i})\qquad(\overline{e_i} \vee \overline{u_i})}}{(\overline{u_i} \vee c_{2i})\qquad\qquad(u_i \vee c_{2i})}}{(c_{2i})}}{\dfrac{(\overline{c_1} \vee \cdots \vee \overline{c_{2n}})}{\bot}}$$

Lemma 6 implies that JM'_n has a linear-sized NS$'$ proof. That is, (JM_n + strategy clauses) has easy NS$'$ refutations. Implying that JM_n has easy proofs in $\forall$Strat+NS$'$. ◀

We end this paper by showing that eFrege+$\forall$red ([8]) p-simulates the strategy based static proof systems $\forall$Strat+NS and $\forall$Strat+NS$'$ (Theorem 31). It is known that eFrege+$\forall$red p-simulates almost all important existing dynamic QBF proof systems [13, Figure 1]. Our result shows that eFrege+$\forall$red is capable of simulating even static QBF-proof systems.

▶ **Theorem 31.** *eFrege+$\forall$red p-simulates $\forall$Strat+NS$'$ and $\forall$Strat+NS.*

Proof. From either a $\forall$Strat+NS$'$ or $\forall$Strat+NS refutation of $\Pi.\phi$ we immediately have a winning strategy S in a series of decision lists $D_1, \ldots D_n$ for each of the universal variables $u_1, \ldots u_n$ as a part of the input. The NS and NS$'$ proofs refute $(\bigwedge_{i=1}^{n} D_i) \wedge \phi$. By propositional simulation (Lemma 5) we can get a PCR refutation of $(\bigwedge_{i=1}^{n} D_i) \wedge \phi$ and thus by propositional simulation [25, Lemma 7.1.1] again an eFrege refutation of the same. The decision list D_i explicitly assigns the universal variable u_i, but instead of D_i which outputs a $\{u_i, \bar{u}_i\}$-assignment, a circuit σ_i in the same input variables can be constructed by re-balancing the decision list into a circuit that outputs a $\{0, 1\}$-assignment (in fact a bounded-depth circuit as shown in [8, Prop. 4.2]). And using eFrege to prove that D_i can be replaced by $u_i \leftrightarrow \sigma_i$ due to logical equivalence. We therefore have an eFrege refutation of $\bigwedge_{i=1}^{n}(u_i \leftrightarrow \sigma_i) \wedge \phi$. Now that we have a formalised proof of the soundness of the strategy, in fact we get a $\forall\mathsf{Strat}_{Circ}$+eFrege refutation which is equivalent to eFrege+$\forall$red (Observation 11).

◀

7 Conclusion and Future work

The paper commences the study of Nullstellensatz (NS) for QBFs as well as a new general ∀Strat framework for static (and dynamic) propositional proof systems along with the reintroduction of ∀Exp. To the best of our knowledge, these are the first static QBF proof systems with non-trivial results in the literature. The remaining open problems in the scope of this paper are pointed below.

In this paper, we show an incomparability result between the ∀Exp+NS′ and the ∀Strat+NS′ systems. However, the relationship between ∀Exp+NS and ∀Strat+NS is still open. As is the case for ∀Exp+NS′ and ∀Strat+NS′ when quantifier alternations are bounded. Bounded alternations are particularly interesting because Q-Res and ∀Exp+Res are not incomparable in bounded alternations [9], but the simulation of Q-Res by ∀Exp+Res clearly uses the dynamic structure of the Resolution proof [9] and it is not clear how this can be replicated in a static system.

Further is the open relationship with Tree-like Q-Resolution. Tree-like ∀Exp+Res can be simulated by ∀Exp+NS′, but it is unclear if Tree-like Q-Resolution can be simulated by ∀Strat+NS′. The issue is whether one can extract a strategy from a Tree-like Q-Resolution proof that when encoded will never cause a change in propositional hardness for NS′.

From our observation in the propositional domain (Lemma 3), we derive that ∀Exp+NS cannot simulate ∀Exp+NS′ and also ∀Strat+NS cannot simulate ∀Strat+NS′. The other direction in both of these is still open.

Finally we can speculate, but require more work to prove that just as eFrege+∀red p-simulates ∀Strat+NS and ∀Strat+NS′, eFrege+∀red also p-simulates ∀Exp+NS and ∀Exp+NS. First the strategy extraction property would have to be proven for these systems and then a formalised strategy extraction argument can be made for a simulation (see [13, 12] for examples of how this can be done for other systems).

References

1 Cook S. A. and Reckhow A. R. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1977.

2 Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM J. Comput.*, 31(4):1184–1211, 2002. doi:10.1137/S0097539700366735.

3 Martin Anthony. *Decision Lists and Related Classes of Boolean Functions*, page 577–596. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2010. doi:10.1017/CBO9780511780448.017.

4 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

5 Albert Atserias and Massimo Lauria. Circular (yet sound) proofs in propositional logic. *ACM Trans. Comput. Log.*, 24(3):20:1–20:26, 2023. doi:10.1145/3579997.

6 Paul Beame, Russell Impagliazzo, Jan Krajícek, Toniann Pitassi, and Pavel Pudlák. Lower bound on Hilbert's Nullstellensatz and propositional proofs. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 794–806. IEEE Computer Society, 1994. doi:10.1109/SFCS.1994.365714.

7 Olaf Beyersdorff, Ilario Bonacina, and Leroy Chew. Lower bounds: From circuits to QBF proof systems. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 249–260. ACM, 2016.

8 Olaf Beyersdorff, Ilario Bonacina, Leroy Chew, and Ján Pich. Frege systems for quantified Boolean logic. *J. ACM*, 67(2):9:1–9:36, 2020. doi:10.1145/3381881.

9 Olaf Beyersdorff, Leroy Chew, Judith Clymo, and Meena Mahajan. Short proofs in QBF expansion. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2019. doi:10.1007/978-3-030-24258-9_2.

10 Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. Proof complexity of resolution-based QBF calculi. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 76–89. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.76.

11 Samuel R. Buss and Toniann Pitassi. Good degree bounds on Nullstellensatz refutations of the induction principle. *J. Comput. Syst. Sci.*, 57(2):162–171, 1998. doi:10.1006/jcss.1998.1585.

12 Leroy Chew. Proof simulation via round-based strategy extraction for QBF. *Electron. Colloquium Comput. Complex.*, TR23-053, 2023. URL: https://eccc.weizmann.ac.il/report/2023/053, arXiv:TR23-053.

13 Leroy Chew and Friedrich Slivovsky. Towards uniform certification in QBF. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 22:1–22:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.22.

14 Matthew Clegg, Jeff Edmonds, and Russell Impagliazzo. Using the groebner basis algorithm to find proofs of unsatisfiability. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 174–183. ACM, 1996. doi:10.1145/237814.237860.

15 William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Log. Program.*, 1(3):267–284, 1984. doi:10.1016/0743-1066(84)90014-1.

16 Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache der reinen Denkens, Halle 1879*. Harvard University Press, Cambridge, 1879. English translation in: from Frege to Gödel, a source book in mathematical logic (J. van Heijenoord editor), 1967.

17 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.

18 Dima Grigoriev and Nicolai N. Vorobjov Jr. Complexity of null-and positivstellensatz proofs. *Ann. Pure Appl. Log.*, 113(1-3):153–160, 2001. doi:10.1016/S0168-0072(01)00055-0.

19 Joshua A. Grochow and Toniann Pitassi. Circuit complexity, proof complexity, and polynomial identity testing: The ideal proof system. *J. ACM*, 65(6):37:1–37:59, 2018. doi:10.1145/3230742.

20 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

21 David Hilbert. Ueber die vollen invariantensysteme. *Mathematische Annalen*, 42:313–373, 1893. URL: http://eudml.org/doc/157652.

22 Mikolás Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. *Artif. Intell.*, 234:1–25, 2016. doi:10.1016/j.artint.2016.01.004.

23 Mikolás Janota and João Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015. doi:10.1016/j.tcs.2015.01.048.

24 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995.

25 Jan Krajícek. *Proof Complexity*. Cambridge university press, 2019.

26 Ronald L. Rivest. Learning decision lists. *Mach. Learn.*, 2(3):229–246, 1987. doi:10.1007/BF00058680.

27 John Alan Robinson. Theorem-proving on the computer. *J. ACM*, 10(2):163–174, 1963.

28 Jakob Nordström Sam Buss. *Proof Complexity and SAT Solving chapter from Handbook of Satisfiability.* 2021. URL: https://ebooks.iospress.nl/volume/handbook-of-satisfiability-second-edition.

29 G. S. Tseitin. On the complexity of derivations in propositional calculus. In A. O. Slisenko, editor, *Studies in Mathematics and Mathematical Logic, Part II*, pages 115–125. springer, 1970.

30 Allen Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In *Principles and Practice of Constraint Programming*, pages 647–663. Springer, 2012.

31 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. doi:10.1007/978-3-662-03927-4.

32 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified boolean satisfiability solver. In Lawrence T. Pileggi and Andreas Kuehlmann, editors, *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, pages 442–449. ACM / IEEE Computer Society, 2002. doi:10.1145/774572.774637.

Appendix *(Missing Algorithms and Examples from Section 5)*

Algorithm 1 is the verifier for a ∀Strat+NS proof and Algorithm 2 is the verifier for a ∀Strat+NS′ proof. These both inherently use the Algorithm 3 to decide if or not the CNF clauses form a decision list.

Algorithm 1 Verifier for ∀Strat+NS

Require: Quantifier prefix Π, list of standard input polynomials $F_1, \ldots, F_n$, Strategy encodings as $poly_{std}(S_1), \ldots, poly_{std}(S_m)$, standard multipliers $g_1, \ldots, g_n, h_1, \ldots, h_m$

function VERIFY($\Pi, \{F_1, \ldots, F_n\}, \{poly_{std}(S_1), \ldots, poly_{std}(S_m)\}, \{g_1, \ldots, g_n\}, \{h_1, \ldots, h_n\}$)

  $sum = \sum_{i \in [n]} F_i g_i + \sum_{j \in [m]} poly_{std}(S_j) h_j$
  if $sum \neq 1$ **then return** FALSE
  end if

  $Strat_C = \{\}$ /*empty list to hold strategy clauses computed from encodings*/
  for $S \in \{poly_{std}(S_1), \ldots, poly_{std}(S_m)\}$ **do**
    if S is a Boolean axiom **then**
      continue /*ignores current clause and moves to the next iteration*/
    end if

    neg = literals appearing in all monomials of S. /*negative literals in the clause*/
    $pos = \{\}$ /*empty list to hold positive literals in the clause*/
    for monomial $M \in S$ **do**
      $M := M \setminus neg$ /*drop the common literals from all monomials*/
      if degree(M)= 1 and coeff(M)= -1 **then**
        pos.append($|M|$)
      end if
    end for
    $S_{new} = neg$
    for literal $l \in pos$ **do**
      $S_{new} := S_{new} \cdot (1 - l)$
    end for
    if $S \neq S_{new}$ **then return** FALSE
    else
      $lit = \{\}$ /*empty list to hold literals of a clause*/
      for literal $l \in neg$ **do**
        lit.append($\bar{l}$)
      end for
      for literal $l \in pos$ **do**
        lit.append(l)
      end for
      $Strat_C$.append(lit)
    end if
  end for
return $VERIFY\text{-}STRATEGY(\Pi, Strat_C)$
end function

Description of Algorithm 1: This ∀Strat+NS-verifier takes as input the Quantifier prefix, input polynomials, polynomials of strategy clauses and multiplier-polynomials of the ∀Strat+NS-refutation π needing verification. Lines 3-5 check if Equation 1 is satisfied for π and reject otherwise. Strategy clauses (ignoring axiom clauses) are handled from lines 13 onward. Lines 13-20 identify positive and negative literals from the polynomial of the strategy clause. Lines 21-25 rebuild the polynomial with the above found positive and negative literals and cross-check if it matches the original and reject otherwise. Lines 27-35 make a set of positive literals as x and negative as $\overline{x}$ to represent a CNF clause and add it to the $Strat_C$ list. That is, at last $Strat_C$ is a list of CNF clauses with each CNF clause in-turn encoded as a list. Finally this algorithm calls the VERIFY-STRATEGY function on these CNF clauses to check if they form a decision list or not.

Step 2 of Example 18 explains lines 13-25 of Algorithm 1. These lines handle the process of mapping the refutation-polynomials to their respective CNF clause-forms. To understand the process of this, let's see two cases in the following example.

▶ **Example 32.** Let polynomial $\mathcal{P}_1 := p_1 - p_1n_1 - p_1n_2 + p_1n_1n_2$
First, positive literals of the clause = literals common in all monomials = $\{p_1\}$ in this case.
Dropping them implies, $\mathcal{P}_1' := 1 - n_1 - n_2 + n_1n_2$
Next, negative literals of the clause = single degree monomials with a coeficient of -1 = $\{n_1, n_2\}$ in this case.
To cross-check, re-building the polynomial implies $\mathcal{P}_1'' := p_1(1-n_1)(1-n_2)$. Opening up the monomials, $\mathcal{P}_1 = \mathcal{P}_1''$. Therefore this polynomial $\mathcal{P}_1$ is valid.
On the other hand, suppose polynomial $\mathcal{P}_2 := p_1 - p_1n_1 - p_1n_2 - p_1n_3 + p_1n_1n_2 - p_1n_1n_3$
positive literals = $\{p_1\}$.
negative literals = $\{n_1, n_2, n_3\}$
$\mathcal{P}_2'' := p_1(1-n_1)(1-n_2)(1-n_3) := p_1 - p_1n_1 - p_1n_2 - p_1n_3 + p_1n_1n_2 + p_1n_1n_3 + p_1n_2n_3 - p_1n_1n_2n_3$. Since $\mathcal{P}_2 \neq \mathcal{P}_2''$, this polynomial $\mathcal{P}_2$ is in-valid.

Description of Algorithm 2: This ∀Strat+NS′-verifier works similar to the ∀Strat+NS-verifier above. That is, it takes as input the Quantifier prefix, input polynomials, polynomials of strategy clauses and multiplier-polynomials of the ∀Strat+NS′-refutation π needing verification. Lines 3-5 check if Equation 2 is satisfied for π and reject otherwise. Strategy clauses (ignoring boolean and twin axioms) are handled from lines 16 onward. Lines 16-23 make a set of positive literals as x and negative as $\overline{x}$ to represent a CNF clause and add it to the $Strat_C$ list. That is, at last $Strat_C$ is a list of CNF clauses with each CNF clause in-turn encoded as a list. Finally this algorithm calls the VERIFY-STRATEGY function on these CNF clauses to check if they form a decision list or not.

Next, we see how the proof from Example 21 can be verified as a correct ∀Strat+NS′ proof.

▶ **Example 33.** Given the Q-polynomials $\mathcal{F}$ and the final proof in Example 21 as input, we will see how to verify if it is a valid ∀Strat+NS′ proof or not.

1. First, we cross-check that Equation 2 is satisfied.
2. Next, we convert the $poly_{twin}(S_i)$s into clauses and ignore any Boolean/ twin axioms present.
   - In a monomial, l are negative literals and l' are positive literals of the clause.
3. The next steps are same as in ∀Strat+NS Example 18. Therefore at the last, the $1 - t_1 - t_1'$ will be ignored and the rest 4 clauses will be marked in the same way as the former example. Therefore it is a valid ∀Strat+NS′ proof.

Algorithm 2 Verifier for ∀Strat+NS′

Require: Quantifier prefix Π, list of twin input polynomials $F_1, \ldots, F_n$, Strategy encodings as $poly_{twin}(S_1), \ldots, poly_{twin}(S_m)$, twin multipliers $g_1, \ldots, g_n, h_1, \ldots, h_m$

1: **function** VERIFY(Π, $\{F_1, \ldots, F_n\}$, $\{poly_{twin}(S_1), \ldots, poly_{twin}(S_m)\}$, $\{g_1, \ldots, g_n\}$, $\{h_1, \ldots, h_n\}$)
2:
3: $sum = \sum_{i \in [n]} F_i g_i + \sum_{j \in [m]} poly_{twin}(S_j) h_j$
4: **if** $sum \neq 1$ **then return** FALSE
5: **end if**
6:
7: $Strat_C = \{\}$ /*empty list to hold strategy clauses computed from encodings*/
8: **for** $S \in \{poly_{twin}(S_1), \ldots, poly_{twin}(S_m)\}$ **do**
9: **if** size(S)$\neq 1$ **then**
10: **if** S is a Boolean/ Twin axiom **then**
11: continue /*ignores current clause and moves to the next iteration*/
12: **else**
13: **return** FALSE
14: **end if**
15: **end if**
16: $lit = \{\}$ /*empty list to hold literals of a clause*/
17: **for** literal $l' \in S$ **do**
18: lit.append(l)
19: **end for**
20: **for** literal $l \in S$ **do**
21: lit.append($\bar{l}$)
22: **end for**
23: $Strat_C$.append(lit)
24: **end for**
25: **return** VERIFY-STRATEGY(Π, $Strat_C$)
26: **end function**

Description of Algorithm 3: This is a general verifier for proof systems using ∀Strat$_{DL}$ framework. It takes as input the quantifier prefix Π and the strategy clauses as CNFs. For a universal variable u, in Lines 5-6, it separates out the decision list clauses of u and sorts them by ascending order of width size (if two clauses are of same width, the one with a positive t variable will be first). Lines 9-11 check if u has some trivial strategy as 0/1 i.e without any decision list. Lines 13-15 check that all negative literals (except u) in the decision clause are already seen t_is, otherwise rejects instantly. Line 16-21 deal with the positive literals (excluding u) in the decision clause, note that if no positive literal in the clause it is the 'else' part of the decision list and if more than 1 positive literal is present, then it is against the definition so rejects immediately. Now in the case that only 1 positive t literal is present, Lines 22-25 separate out the definition clauses of this t variable and find the one clause C_T with $\bar{t}$ in it. The Lines 26-33 are checking if the remaining literals (say x) in C_T are to the left of u in Π and if there exist definition clauses $t \vee \overline{x}$ in the strategy clauses. Once all such clauses exist, we mark them and remember that this particular t has been handled. This process repeats as you keep finding new ts in the decision clauses of every universal variable $u \in \Pi$. Finally we accept the input if all clauses have been marked and none of the clauses we search for are missing.

Algorithm 3 Algorithm for checking if strategy clauses form a decision list

Require: Quantifier prefix Π and list of CNF clauses $Strat_C$.

function VERIFY-STRATEGY($\Pi, Strat_C$)
  $n = length(Strat_C)$
  $checked[n] = \{0, \ldots, 0\}$
  for $\forall u \in \Pi$ **do**
    $clauses_u := \{C \in Strat_C \mid u/\overline{u} \in C\}$ /*decision list clauses*/
    sort-by-width-ascending($clauses_u$) /*if equal positive t-clause will be first*/
    $vars_T = \{\}$ /*empty list to store Tseitin variables used for a u*/
    for $C \in clauses_u$ **do**
      if length(C)= 1 **then** /*trivial 0/1 strategy*/
        $checked$=MARK-CHECKED($C, checked, Strat_C$)
        break /*breaks loop to fetch another universal u*/
      end if
      $neg := \{l \mid \bar{l} \in C \ \& \ var(l) \neq u\}$
      if $neg \neq vars_T$ **then return** FALSE
      end if
      $pos := \{l \mid l \in C \ \& \ var(l) \neq u\}$
      if length(pos)> 1 **then return** FALSE
      end if
      $checked$=MARK-CHECKED($C, checked, Strat_C$)
      if length(pos)= 0 **then** break /*last line in decision list*/
      end if
      $clauses_T := \{C' \in Strat_C \mid pos/\overline{pos} \in C' \ \& \ u \notin C'\}$ /*definition clauses*/
      $C_T := \{C' \in clauses_T \mid \overline{pos} \in C'\}$ /*long $\bar{t}$ clause*/
      if length(C_T)> 1 **then return** FALSE
      end if
      for literal $l \in C_T$ & $l \neq \overline{pos}$ **do**
        if $l \notin L_\Pi(u)$ **then return** FALSE
        end if
        if $\{\bar{l}, pos\} \notin clauses_T$ **then return** FALSE
        else
          $checked$=MARK-CHECKED($\{\bar{l}, pos\}, checked, Strat_C$)
        end if
      end for
      $checked$=MARK-CHECKED($C_T, checked, Strat_C$)
      $vars_T$.append(pos)
    end for
  end for
  if $checked[n] = \{1, \ldots, 1\}$ **then return** TRUE
  else return FALSE
  end if
end function
function MARK-CHECKED($C, checked, Strat_C$)
  $indx = Strat_C.index(C)$
  $checked[indx] = 1$
  return $checked$
end function

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il