Colourful TFNP and Propositional Proofs

Ben Davis
School of Computer Science, McGill University, Montreal, Canada

Robert Robere
School of Computer Science, McGill University, Montreal, Canada

Abstract

Recent work has shown that many of the standard TFNP classes – such as PLS, PPADS, PPAD, SOPL, and EOPL – have corresponding proof systems in propositional proof complexity, in the sense that a total search problem is in the class if and only if the totality of the problem can be efficiently proved by the corresponding proof system. We build on this line of work by studying *coloured* variants of these TFNP classes: C-PLS, C-PPADS, C-PPAD, C-SOPL, and C-EOPL. While C-PLS has been studied in the literature before, the coloured variants of the other classes are introduced here for the first time. We give a family of results showing that these coloured TFNP classes are natural objects of study, and that the correspondence between TFNP and natural propositional proof systems is not an exceptional phenomenon isolated to weak TFNP classes. Namely, we show that:

- Each of the classes C-PLS, C-PPADS, and C-SOPL have corresponding proof systems characterizing them. Specifically, the proof systems for these classes are obtained by adding depth to the formulas in the corresponding proof system for the uncoloured class. For instance, while it was previously known that PLS is characterized by bounded-width Resolution (i.e. depth 0.5 Frege), we prove that C-PLS is characterized by depth-1.5 Frege ($\mathrm{Res}(\mathsf{polylog}(n))$).
- The classes C-PPAD and C-EOPL coincide exactly with the uncoloured classes PPADS and SOPL, respectively. Thus, both of these classes also have corresponding proof systems: unary Sherali-Adams and Reversible Resolution, respectively.
- Finally, we prove a *coloured intersection theorem* for the coloured sink classes, showing $\mathsf{C\text{-}PLS} \cap \mathsf{C\text{-}PPADS} = \mathsf{C\text{-}SOPL}$, generalizing the intersection theorem $\mathsf{PLS} \cap \mathsf{PPADS} = \mathsf{SOPL}$. However, while it is known in the uncoloured world that $\mathsf{PLS} \cap \mathsf{PPAD} = \mathsf{EOPL} = \mathsf{CLS}$, we prove that this equality *fails* in the coloured world in the black-box setting. More precisely, we show that there is an oracle O such that $\mathsf{C\text{-}PLS}^O \cap \mathsf{C\text{-}PPAD}^O \supsetneq \mathsf{C\text{-}EOPL}^O$.

To prove our results, we introduce an abstract multivalued proof system – the *Blockwise Calculus* – which may be of independent interest.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases oracle separations, TFNP, proof complexity, Res(k), lower bounds

Digital Object Identifier 10.4230/LIPIcs.CCC.2023.36

Related Version *Full Version*: https://eccc.weizmann.ac.il/report/2023/068/

Funding B.D. and R.R. supported by NSERC.

1 Introduction

1.1 Introduction to TFNP and Proof Complexity

This work continues a recent line of research relating the theory of *total* NP *search problems* [22, 27] to the theory of *propositional proof complexity*.[1] A total NP search problem is a search problem S satisfying:

[1] This paper is an extended abstract. Please see the full version at https://eccc.weizmann.ac.il/report/2023/068/.

- **Totality.** On every input x, some solution y with $|y| \leq |x|^{O(1)}$ is guaranteed to exist.
- **Efficient Certification.** Checking if y is a valid solution for x is polynomial-time computable.

The class TFNP contains all such search problems, and many important computational problems lie inside of this class – such as the problem of computing a Nash equilibrium of a bimatrix game, or the problem of computing a prime factor of a given number.

Since the initial study of TFNP it has been known that no problem in TFNP can be NP-Hard unless NP = coNP [26]. As a result, in order to understand the internal structure of TFNP, researchers have defined subclasses of TFNP based on polynomial-time reducibility to fixed total search problems [27]. For example, some of the most well-studied subclasses of TFNP can be defined by reductions to the following problems:

- PLS. Given a directed acyclic graph, output a sink node.
- PPAD. Given a directed graph with an unbalanced node (in-degree $\neq$ out-degree), output another unbalanced node.
- PPADS. Given a directed graph with a *negatively* unbalanced node (in-degree $<$ out-degree), output a *positively* unbalanced node (in-degree $>$ out-degree).

The theory of TFNP has been an extraordinary success in capturing the complexity of many computational problems that have avoided classification in other settings. For example, the class PPAD captures the complexity of computing a Nash Equilibrium [15] along with other important problems in economics [13, 11, 12].

Black-Box TFNP Classes and Propositional Proof Systems

An important caveat in the definitions of the above classes is in the input representation. It is clear that all of the above problems are computationally easy (i.e. inside of P), if we are given the directed graphs in some standard encoding like an adjacency list or an adjacency matrix. Instead, in the definitions of the TFNP classes we assume that the inputs are given *implicitly*. For instance, we can represent an (exponentially large) $O(1)$-degree directed graph G by a polynomial-size boolean circuit C that, when given a node $u \in V(G)$ as input, outputs the list of in- and out-neighbours of u. When described in this implicit encoding, we can no longer exhaustively search through the graph to find a solution to the search problem, but, when given a potential solution we can still verify its correctness in polynomial time.

Another natural way of implicitly representing an input to a total search problem is by using *black-box* (also called *query*) access, where the input is represented by an oracle. Following the earlier example, now the graph G would be represented by an oracle which receives a node $u \in V(G)$ as input and outputs the list of neighbours of u. For now, we informally define TFNP^{dt} as the class of total search problems where the inputs are represented as black-boxes in this way. The seminal work of Beame et al. [3] demonstrated that this model is closely related to *oracle separations* between the standard TFNP classes. In particular, if we have two black-box TFNP subclasses A^{dt} and B^{dt}, then a containment $\mathsf{A}^{dt} \subseteq \mathsf{B}^{dt}$ by a sufficiently uniform simulation implies that $\mathsf{A} \subseteq \mathsf{B}$ – since we can always simulate the black-box by evaluating the circuit – but if $\mathsf{A}^{dt} \not\subseteq \mathsf{B}^{dt}$ then there is an oracle O such that $\mathsf{A}^{O} \not\subseteq \mathsf{B}^{O}$ [3]. Beame et al. [3] used this strategy to construct oracle separations between many pairs of TFNP classes that were not previously separated.

Another major contribution of Beame et al. [3] was pioneering the use of *propositional proof complexity* in the study of TFNP classes. They showed that if a total search problem S lies in the (black-box) class $\mathsf{PPA}^{dt} \subseteq \mathsf{TFNP}^{dt}$, then a particular unsatisfiable CNF formula F_S associated with S has efficient refutations in the well-studied algebraic Nullstellensatz proof system. By combining this with a lower bound against Nullstellensatz proofs refuting the pigeonhole principle $\textsc{Php}_n^{n+1}$ they provided the first oracle separation between the classes PPP^O and PPA^O. Proof complexity was also employed as a crucial tool by Buresh-Oppenheim and Morioka [8], who used it to unify previous oracle separations in black-box TFNP and also provide new results about the class PLS^{dt}.

Very recently, the relationships between TFNP^{dt} subclasses and propositional proof systems have been revisited [10, 21, 20, 9]. One surprising outcome of the emerging work is that: not only can proof complexity lower bounds be used to construct oracle separations (as in [3]) but, proof complexity lower bounds in fact turn out to be *equivalent* to these oracle separations! More formally, for many of the most well-studied TFNP classes A (e.g. A = PLS, PPAD, PPADS, CLS = EOPL, SOPL), there is a corresponding propositional proof system P_{A} such that the following relationship holds:

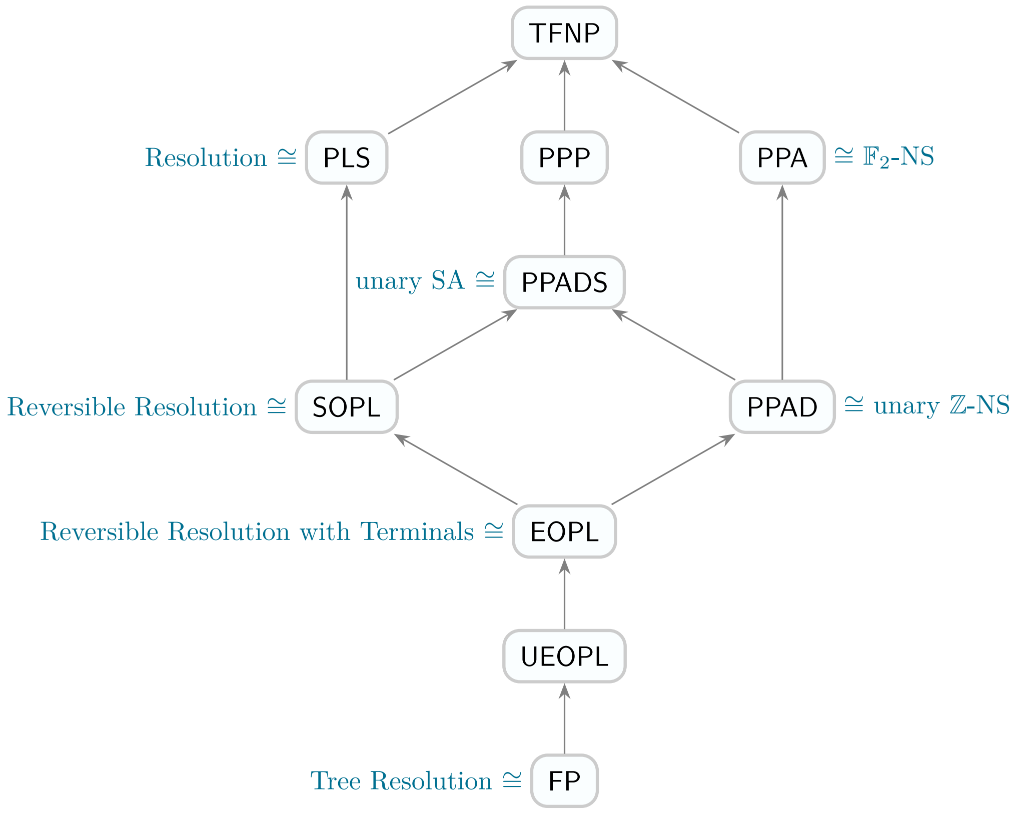
A total search problem S lies in the class A^{dt}
if and only if
The propositional encoding of the totality of S can be efficiently proved in P_{A}.

These new equivalences led to a number of new results in both the theory of propositional proof complexity and the theory of TFNP. In the theory of TFNP, for example, for each of the classes listed above, such propositional proof systems not only exist, but they are natural proof systems that have been well-studied in the proof complexity literature (cf. Figure 1). In [20] these new equivalences led to the proofs of oracle separations between the TFNP classes PLS and PPP as well as between UEOPL and EOPL, finally resolving all oracle separations between the classical TFNP classes. On the other hand, the breakthrough collapse CLS = PLS ∩ PPAD [16] and its followups EOPL = PLS ∩ PPAD and SOPL = PLS ∩ PPADS [19] led to brand-new *intersection theorems* in proof complexity. In particular, there are natural proof systems – P_1, P_2, P_3 – such that a formula F has an efficient proof in P_1 *if and only if* F has efficient proofs in P_2 *and* efficient proofs in P_3. This is illustrated by the work of [20], which showed that the proof system *Reversible Resolution* – which is closely related to Max-SAT solving – is the intersection of the classical *Resolution* proof system and the *Sherali-Adams* proof system. Section 2 outlines the formal definitions of these proof systems.

Next Steps

In light of these results a number of open problems – both concrete and conceptual – remain, namely:

- Do *all* TFNP subclasses defined by a syntactic existence principle admit a characterization by a natural proof system? The recent work of Buss, Fleming, and Impagliazzo [9] constructs a Cook-Reckhow proof system for *every* black-box TFNP class, but, it is not clear if these proof systems are equivalent to standard systems occurring in the literature.
- If the above is not true, what is special about these "weak" TFNP classes that do admit characterizations by proof systems?
- Are the intersection theorems CLS = EOPL = PLS ∩ PPAD and SOPL = PLS ∩ PPADS a unique phenomenon, or do other instances of intersection theorems exist? If so, do they imply other intersection results for proof complexity?
- Do other well-studied TFNP classes not depicted above that correspond to natural proof systems? (Note that many other well-studied TFNP subclasses – such as the classes PPP and UEOPL – and other classes corresponding to the weak pigeonhole principle or Ramsey's theorem are currently not known to admit nice characterizations by proof systems).



TFNP
Resolution ≅ PLS
PPP
PPA ≅ $\mathbb{F}_2$-NS
unary SA ≅ PPADS
Reversible Resolution ≅ SOPL
PPAD ≅ unary $\mathbb{Z}$-NS
Reversible Resolution with Terminals ≅ EOPL
UEOPL
Tree Resolution ≅ FP

Figure 1 Class inclusion diagram for TFNP. An arrow A → B means A ⊆ B relative to all oracles. In the black-box model some classes can be captured using propositional proof systems, as indicated in blue. Above SA refers to the Sherali-Adams proof system [29], NS refers to the Nullstellensatz proof system [4], and "unary" refers to the fact that we measure size by the sum of all coefficients occurring in the proof.

1.2 Our Results

In this paper we introduce a new family of TFNP classes and demonstrate that they have natural corresponding propositional proof systems. Specifically, we consider a systematic way to generalize the TFNP classes PLS, PPAD, PPADS, EOPL, SOPL, obtaining their *coloured* generalizations C-PLS, C-PPAD, C-PPADS, C-EOPL, and C-SOPL. The formulas embodying the class C-PLS have previously been studied in proof complexity and bounded arithmetic, particularly in connection with *witnessing theorems* for the bounded arithmetic theory T^2_2 [24, 31]. For the other classes, however, the coloured variants are introduced and systematically studied here for the first time to the best of our knowledge. Before we discuss our results for these coloured classes, let us first describe to generalize a TFNP class to its coloured variant.

From Uncoloured to Coloured TFNP Classes

The key shared property between the classes PLS, PPADS, PPAD, EOPL, SOPL is the following: the input to each of these problems is a directed graph – enforced to be acyclic[2] in the case of PLS, EOPL, and SOPL – having distinguished source node s with at least one

[2] We can enforce acyclicity by adding in a decreasing potential function on the nodes of the graph, and requiring that edges must point from nodes of higher potential to nodes of lower potential.

outgoing edge. The goal of the search problem is to either output a *proper sink node* in the input graph (i.e. a sink node with at least one in-neighbour) or, in the case of PPAD and EOPL, one can also output a *proper source node* (i.e. a source node with at least one out-neighbour) other than the distinguished one[3].

In the coloured generalization of these problems, we receive a list of n *colours* $C_u \subseteq [n]$ for each node $u \in V(G)$ along with the directed graph G as input, and the solutions are updated as follows:

- Any proper source node *with a colour* is a solution,
- Any sink node *with no colour* (i.e. if $C_u = \emptyset$) is a solution, and
- A node u with an out-neighbour v is a solution if there is a colour $\lambda \in C_v$ such that $\lambda \notin C_u$.

To state the totality as an unsatisfiable system of constraints: the graph G has at least one proper source, all proper sources are colourless, all sinks have at least one colour, and colours propagate *backwards* across directed edges – if a node u has v as an out-neighbour then $C_v \subseteq C_u$. All of these constraints are obviously testable in polynomial time, with the possible exception of testing for a colourless sink. For this, we require that if a node is a sink node, then there is a polynomial-time function that points to some colour that is present at that sink. Note that knowing only the identity of a node, it is no longer simple to test whether it is colourless. This is unlike the analogous and easily-testable property in the uncoloured problems that a node has a successor, giving some intuition for the increased difficulty of the coloured problems. See Figure 2 for the hierarchy of these coloured problems and how they relate to classical TFNP classes, and Section 2.2 for formal definitions.

Statement of Results

Before stating our main results we require some formal definitions. A *query total search problem* is a sequence of relations $R_n \subseteq \{0,1\}^n \times O_n$, one for each $n \in \mathbb{N}$, such that $\forall x \in \{0,1\}^n \exists o \in O_n : (x, o) \in R_n$. We think of x as being provided to us via query access to its individual bits, and so an "efficient" algorithm would intuitively be provided by a $\mathsf{polylog}(n)$-depth decision tree solving the search problem. The search problem $\mathrm{R} = (R_n)_n$ is in TFNP^{dt} if, for each $o \in O_n$, there is a $\mathsf{polylog}(n)$-depth decision tree T_o such that $T_o(x) = 1$ iff $(x, o) \in R_n$. Furthermore, given a search problem R, we can define a corresponding subclass of TFNP^{dt}, denoted R^{dt}, obtained by taking all query total search problems that have low-depth decision-tree reductions to R (see Section 2.2 for the formal definition of a reduction in this model).

The canonical examples of total search problems in TFNP^{dt} come from low-width unsatisfiable CNF formulas. Any unsatisfiable CNF formula $F = C_1 \wedge \cdots \wedge C_m$ over variables $x_1, \ldots, x_n$ yields a closely related total search problem $S(F) \subseteq \{0,1\}^n \times [m]$: given an assignment x to the variables of F, output the index of a falsified clause of $F(x)$. Given a sequence of unsatisfiable CNF formulas $F = (F_n)_n$, the search problem $S(F) := (S(F_n))_n \in \mathsf{TFNP}^{dt}$ if and only if the width of (some unsatisfiable subformula of) F is $\mathsf{polylog}(n)$. Conversely, given any total search problem $R_n \subseteq \{0,1\}^n \times O_n$ we can define the unsatisfiable CNF formula $\bigwedge_{o \in O_n} \neg T_o$, where $\neg T_o$ is the encoding of the negation of the decision tree T_o as a CNF formula. It is easy to see that a query-efficient algorithm for R_n exists iff one exists for $S(\bigwedge_{o \in O_n} \neg T_o)$, and thus we can focus on search problems of the form $S(F) \in \mathsf{TFNP}^{dt}$ without loss of generality.

[3] For the interested reader, we note that this similarity was identified and formalized as a general GRID search problem in [19].

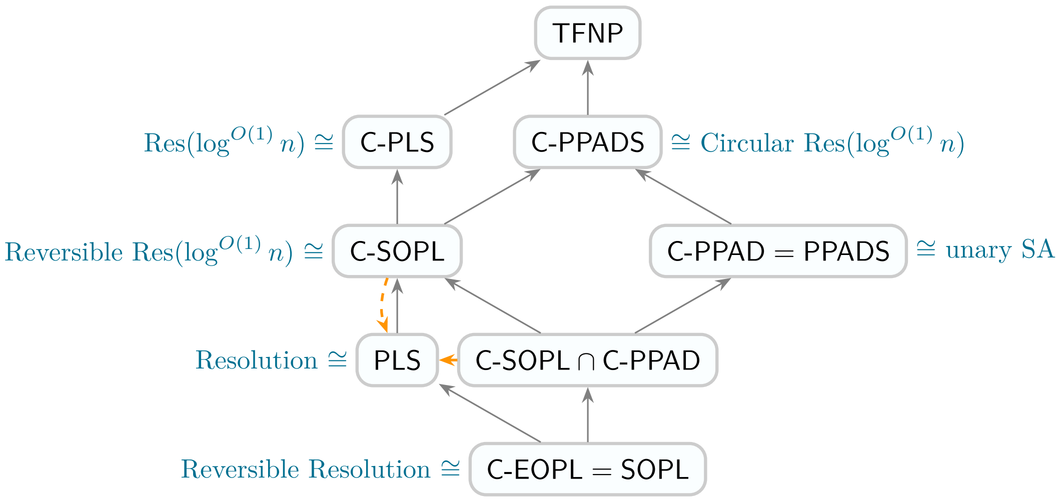

TFNP
$\mathrm{Res}(\log^{O(1)} n) \cong$ C-PLS
C-PPADS $\cong$ Circular $\mathrm{Res}(\log^{O(1)} n)$
Reversible $\mathrm{Res}(\log^{O(1)} n) \cong$ C-SOPL
C-PPAD = PPADS $\cong$ unary SA
Resolution $\cong$ PLS
C-SOPL ∩ C-PPAD
Reversible Resolution $\cong$ C-EOPL = SOPL

Figure 2 The coloured TFNP classes and the corresponding proof systems considered in this paper. A solid line from A to B indicates that A is contained in B relative to every oracle, while a red dashed line means A is not contained in B relative to some oracle.

Given these definitions we can state our main results, summarized in Figure 2. First, we show that *every* coloured class defined above has an equivalent propositional proof system. Moreover, these proof systems are closely related to the proof systems for the uncoloured variants. Given a black-box TFNP class A^{dt} and a proof system P, we write $\mathsf{A}^{dt} \cong P$ if the following holds: for every sequence of unsatisfiable CNFs $F = (F_n)_n \in \mathsf{TFNP}^{dt}$, $S(F) \in \mathsf{A}^{dt}$ if and only if there is a $n^{\mathsf{polylog}(n)}$-size, $\mathsf{polylog}(n)$-degree refutation of F_n in P.

▶ **Theorem 1.1.** *The following equivalences between* TFNP^{dt} *classes and proof systems hold:*

- $\mathsf{C\text{-}PLS}^{dt} \cong \mathrm{Res}(\mathsf{polylog}(n))$,
- $\mathsf{C\text{-}PPADS}^{dt} \cong \mathrm{CircRes}(\mathsf{polylog}(n))$,
- $\mathsf{C\text{-}SOPL}^{dt} \cong \mathrm{RevRes}(\mathsf{polylog}(n))$.

In the above theorem, $\mathrm{Res}(\mathsf{polylog}(n))$ is the extension of Resolution to DNF formulas with $\mathsf{polylog}(n)$-width conjunctions on the bottom level (see e.g. [23, 28, 1, 18]). The system $\mathrm{Res}(n)$ is equivalent to depth-2 Frege, and thus $\mathrm{Res}(\mathsf{polylog}(n))$ sits between Resolution and depth-2 Frege in power. The *Reversible* $\mathrm{Res}(\mathsf{polylog}(n))$ system (denoted $\mathrm{RevRes}(\mathsf{polylog}(n))$) is the natural extension of reversible Resolution to DNF formulas. The *Circular* $\mathrm{Res}(\mathsf{polylog}(n))$ system is exactly the *higher-depth analogue* of Sherali-Adams which is allowed to "operate" on DNF formulas. It is obtained by augmenting the $\mathrm{RevRes}(\mathsf{polylog}(n))$ system with a new rule that allows us to introduce any DNF formula D for free, as long as we (eventually) derive a copy of D later in the proof to make up for the introduced copy. This notion of a "circular, yet sound" proof was introduced by [2] in the setting of Resolution, where it was observed that Circular Resolution is exactly the same as Sherali-Adams. It is quite remarkable that augmenting the three TFNP classes PLS, PPADS, and SOPL with colours yields new natural classes whose corresponding proof systems are simply the proof systems for the uncoloured class where the lines have one greater depth!

Our second main result deals with the coloured "source-or-sink" classes C-PPAD and C-EOPL. Here, we show an *a-priori* unexpected collapse actually occurs: the *coloured* source-or-sink classes are exactly the same as the *uncoloured* sink classes. Consequentially, we obtain propositional proof systems equivalent to these TFNP^{dt} classes by relying on earlier work [20].

▶ **Theorem 1.2.** *The collapses* $\mathsf{C\text{-}EOPL} = \mathsf{SOPL}$ *and* $\mathsf{C\text{-}PPAD} = \mathsf{PPADS}$ *hold. As a consequence,* $\mathsf{C\text{-}EOPL}^{dt} \cong$ *Reversible Resolution and* $\mathsf{C\text{-}PPAD}^{dt} \cong$ *Unary Sherali-Adams.*

In order to prove the above collapses between TFNP classes, we actually proceed entirely through proof complexity. That is, we exploit the prior results $\mathsf{SOPL}^{dt} \cong \mathrm{RevRes}$, as well as $\mathsf{PPADS}^{dt} \cong \mathrm{uSA}$ [20], and give *refutations* of the defining principles of $\mathsf{C\text{-}EOPL}$ and $\mathsf{C\text{-}PPAD}$ in the corresponding proof systems. By applying the known characterization results we then obtain the collapses between these TFNP classes immediately. While we do not see how to prove these collapses directly in the language of TFNP, this only further necessitates studying the relationship between the two areas.

Our third major result is a generalization of the intersection theorem $\mathsf{SOPL} = \mathsf{PLS} \cap \mathsf{PPADS}$ to the coloured setting. This proves, as an immediate consequence, that the proof system $\mathrm{RevRes}(k)$ is the "intersection" of $\mathrm{Res}(k)$ and $\mathrm{CircRes}(k)$.

▶ **Theorem 1.3.** $\mathsf{C\text{-}SOPL}^{dt} = \mathsf{C\text{-}PLS}^{dt} \cap \mathsf{C\text{-}PPADS}^{dt}$.

▶ **Corollary 1.4.** *For any* $\mathsf{polylog}(n)$*-width CNF formula* F *on* n *variables, there is a* $n^{\mathsf{polylog}(n)}$*-size* $\mathrm{RevRes}(\mathsf{polylog}(n))$ *refutation of* F *if and only if there is a* $n^{\mathsf{polylog}(n)}$*-size* $\mathrm{Res}(\mathsf{polylog}(n))$ *refutation of* F *and a* $n^{\mathsf{polylog}(n)}$*-size* $\mathrm{CircRes}(\mathsf{polylog}(n))$ *refutation of* F*.*

Finally, and quite surprisingly, we show that the intersection theorem $\mathsf{C\text{-}EOPL} = \mathsf{C\text{-}PLS} \cap \mathsf{C\text{-}PPAD}$ actually *fails* relative to an oracle. In other words, there is an oracle O such that $\mathsf{C\text{-}EOPL}^O \neq \mathsf{C\text{-}PLS}^O \cap \mathsf{C\text{-}PPAD}^O$.

▶ **Theorem 1.5.** $\mathsf{C\text{-}EOPL}^{dt} \subsetneq \mathsf{C\text{-}PLS}^{dt} \cap \mathsf{C\text{-}PPAD}^{dt}$*, or, equivalently* $\mathsf{SOPL}^{dt} \subsetneq \mathsf{C\text{-}PLS}^{dt} \cap \mathsf{PPADS}^{dt}$.

We show this theorem as follows. Since $\mathsf{C\text{-}EOPL}^{dt} = \mathsf{SOPL}^{dt} \subseteq \mathsf{PLS}^{dt}$, the intersection theorem would imply that

$$\mathsf{C\text{-}PLS}^{dt} \cap \mathsf{C\text{-}PPAD}^{dt} = \mathsf{C\text{-}PLS}^{dt} \cap \mathsf{PPADS}^{dt} \subseteq \mathsf{PLS}^{dt}.$$

However, we can actually show the (even stronger) separation that $\mathsf{C\text{-}SOPL}^{dt} \cap \mathsf{PPADS}^{dt} \not\subseteq \mathsf{PLS}^{dt}$. The fact that $\mathsf{PPADS}^{dt} \not\subseteq \mathsf{PLS}^{dt}$ follows from [20], and we can prove directly that $\mathsf{C\text{-}SOPL}^{dt} \not\subseteq \mathsf{PLS}^{dt}$. We then show that for PLS^{dt}, one can *combine* the adversary arguments from the previous two separations to create an adversary for $\mathsf{PPADS}^{dt} \cap \mathsf{C\text{-}SOPL}^{dt}$. The combination of adversaries holds generically, and shows that PLS^{dt} is itself *not* a non-trivial intersection class (we discuss this more in the full version of the paper.) Taken together, our results paint a intriguing picture for how the coloured TFNP classes relate to the uncoloured classes.

The Blockwise Calculus

The results that we prove in this paper have the unfortunate property of becoming quite proof-theoretically technical when trying to proceed directly. Our primary technical innovation – when compared to the recent work between TFNP and proof complexity – is the use of *multivalued logic* to simplify these arugments. In particular, we found it useful to abstract out a generalized calculus – called the *Blockwise Calculus* – in which to implement our proofs. One can think of the Blockwise Calculus as the natural extension of the Resolution proof system to multivalued variables. In Section 3 we define the Blockwise Calculus and its Reversible and Circular variants, as well as discuss its basic properties. In particular, we show how to translate refutations in the Blockwise Calculus and its variants *automatically* into refutations in Resolution, $\mathrm{Res}(k)$, and their variants.

Open Problems

In general this work suggests that further investigation of the connection between TFNP subclasses and propositional proof complexity is an avenue ripe for exploration.

- As previously outlined, Atserias and Lauria showed that Sherali-Adams is polynomially equivalent to the Circular Resolution proof system [2]. Is there an analogue of this result for Circular Res(k)? That is, is there a natural semi-algebraic proof system that generalizes Sherali-Adams and is polynomially equivalent to Circular Res(k)?
- It was recently shown that Resolution does not polynomially-simulate unary Sherali-Adams and *vice-versa* [20]. Can we prove similar separations between Res(k) and CircRes(k)? Note that one direction of this separation is already known: Res(k) cannot simulate CircRes(k) as the retraction Pigeonhole Principle is easy for CircRes(k) [14] but hard for Res(k) [28].
- What combinatorial principles capture even higher-depth proof systems? We note that some principles (e.g. the *Game Induction principles*) are known using translations from bounded arithmetic [5, 30].

Paper Organization

The rest of the paper proceeds as follows. In Section 2 we introduce the formal definitions of the propositional proof systems and TFNP subclasses that we consider. In Section 3 we define the Blockwise Calculus and its variants, as well as prove our main technical theorems relating the Blockwise Calculus to the boolean proof systems introduced in Section 2. We refer to the full version of the paper for proofs of our new containment and separation results, respectively.

2 TFNP Classes and Propositional Proof Systems

2.1 Propositional Proof Systems

In this section we recall the definitions of some of the standard proof systems considered in this paper. First, we recall the simplest proof system, *Resolution*, and its variant *Reversible Resolution* [20]. The Reversible Resolution variant (and, in particular, the "reversible" rules presented below) were first introduced in the context of MaxSAT solving [7, 25, 17].

▶ **Definition 2.1.** *Let F be a CNF formula and let C be a clause. A* Resolution proof *of C from F is given by a sequence of clauses $C_1, C_2, \ldots, C_s = C$ where the sequence is generated as follows. Starting from the empty sequence we either choose a clause from F to append to the sequence, or, we choose earlier clauses in the sequence and apply one of the proof rules depicted below to generate new clauses to append to the sequence.*

$$\frac{C \vee \ell \quad C \vee \overline{\ell}}{C}\ (\textbf{\textit{Resolution}}) \qquad \frac{C}{C \vee \ell \quad C \vee \overline{\ell}}\ (\textbf{\textit{Reverse Resolution}})$$

The proof is a refutation *if $C = \bot$. The* length *of the proof is s, the number of clauses, and the* width *of the proof is the size of the widest clause in the proof. Finally, the proof is a* Reversible Resolution *proof if every clause is used as the hypothesis of at most one proof rule.*

We will also use *Sherali-Adams proofs*, which are one of the basic semi-algebraic proof systems studied in the literature. In particular we need its *unary* variant.

▶ **Definition 2.2.** *If $C = \bigwedge_{i \in S} x_i \vee \bigwedge_{j \in T} \overline{x}_j$ is a conjunction then we let $p(C) := \prod_{i \in S} x_i \prod_{j \in T}(1 - x_j)$ denote the encoding of C as a real polynomial. A* conical junta *is a non-negative combination of conjunctions $\sum_\lambda \lambda p(C)$ where all coefficients are positive integers. A* Sherali-Adams refutation *of a CNF formula $F = C_1 \wedge \cdots \wedge C_m$ is given by a set of polynomials $p_1, ..., p_m$ and a conical junta $\mathcal{J}$ such that:*

$$\sum_{i=1}^{m} p_i \cdot p(\overline{C}_i) + \mathcal{J} = -1,$$

where all polynomial arithmetic is performed modulo the ideal generated by $\langle x_i^2 - x_i \rangle_{i=1}^n$. The unary size *of the refutation is the sum of all coefficients of all monomials in the expression above (after expansion), and the* degree *of the proof is the maximum degree of any monomial in the expanded expression above. We write* uSA *to denote the Sherali-Adams system where we measure size by unary size.*

The main focus of the present work is the higher-depth analogue of Resolution, known as Res(k), which operates on low-width DNF formulas. We consider three different variants of the Res(k) system (the *standard*, *reversible*, and *circular* variants), and for the sake of uniformity define them all using the same proof rules (cf. Figure 3).

∧-Introduction $\dfrac{D \vee A \qquad D \vee B}{D \vee (A \wedge B) \quad D \vee A \vee B}$ **Cut** $\dfrac{D \vee A \qquad D \vee \overline{A}}{D}$

Reverse Cut $\dfrac{D}{D \vee A \quad D \vee \overline{A}}$ **Axiom Introduction** $\dfrac{}{\ell \vee \overline{\ell}}$

Figure 3 The Res(k) Proof Rules. Above D is any DNF formula, A is a conjunction of boolean literals, ℓ is a boolean literal, and we use the convention that $\overline{A} = \bigvee_{\ell \in A} \overline{\ell}$.

▶ **Definition 2.3.** *Let F be a CNF formula, let G be a DNF formula, and let k be a positive integer. A* Res(k) proof *of G from F is a sequence of k-DNF formulas $D_1, ..., D_s = G$ where the sequence is generated as follows: starting from the empty sequence we either choose a clause from F to append to the sequence (interpreted as a width-1 DNF), or, we choose earlier DNFs in the sequence and apply any* Res(k)*-proof rule (cf. Figure 3) to generate new DNFs and append them to the sequence. The proof is a* refutation *of F if $G = \bot$, the empty disjunction. The* size *of the proof is $\sum_{i=1}^{s} |D_i|$, where $|D_i|$ represents the size of each DNF. The proof is* reversible *(or a* RevRes(k) proof*) if every DNF is used as a hypothesis of at most one proof rule.*

We now define *Unary Circular* Res(k) (or uCircRes(k)) proofs, which are a generalization of Res(k) in which the proofs can have cycles. As discussed in the introduction this is the higher-depth analogue of Sherali-Adams [6]. To define it we must introduce one additional proof rule called DNF Creation, defined next, that allows to create any DNF D in one proof step. While this rule is (obviously) not sound by itself, it turns out that one can make a sound proof system as long as we require that the proof eventually *derives* at least as many copies of D from other proof rules than were created by using the DNF Creation rule, and strictly more if D is the clause we wish to prove (cf. [2]).

$\dfrac{}{D}$ (**DNF Creation**)

▶ **Definition 2.4.** *Let F be a CNF formula. A* Unary Circular $\mathrm{Res}(k)$ proof *of a DNF G from F is a sequence of DNFs $D_1, D_2, \ldots, D_s = G$ that is generated as follows: starting from the empty sequence we either choose a clause C from F and append it to the sequence, we apply the DNF Creation rule to generate a new DNF and add it to the list, or we choose earlier DNFs in the sequence and apply a* $\mathrm{Res}(k)$ *proof rule to generate new DNFs and append them to the sequence. In addition, we make the following stipulations: each DNF D_i in the sequence is used as the hypothesis of at most one* $\mathrm{Res}(k)$ *rule, and every DNF D appearing in the proof is derived as the output of some proof rule at least as many times as it is created using DNF Creation, except the conclusion G which must be derived strictly more times than it is created with DNF-Creation. The* size *of the proof is $\sum_{i=1}^{s} |D_i|$. If $G = \bot$ then we call this a* $\mathrm{uCircRes}(k)$ *refutation of F.*

Both $\mathrm{Res}(k)$ and $\mathrm{CircRes}(k)$ can efficiently simulate $\mathrm{RevRes}(k)$, since $\mathrm{RevRes}(k)$ is a restriction of both systems – of the first system because of the fanout restriction, and of the second system because of its inability to apply DNF Creation.

2.2 Search classes

In this section we define the relevant background for TFNP. We follow the treatment of black-box TFNP used by [20].

▶ **Definition 2.5.** *A* total (query) search problem *is a sequence of relations $R = \{R_n \subseteq \{0,1\}^n \times O_n\}$, where O_n are finite sets, such that for all $x \in \{0,1\}^n$ there is an $o \in O_n$ so that $(x, o) \in R_n$. A total search problem R is in* TFNP^{dt} *if for each $o \in O_n$ there is a decision tree T_o with depth* $\mathsf{poly}(\log n)$ *such that for every $x \in \{0,1\}^n$, $T_o(x) = 1$ iff $(x, o) \in R$.*

As discussed in the introduction the canonical problems in TFNP^{dt} are the *false clause search problems* associated with an unsatisfiable $\mathsf{polylog}(n)$-width CNF formula $F = C_1 \wedge \cdots \wedge C_m$ defined as $S(F) \subseteq \{0,1\}^n \times [m]$ with $(x, i) \in S(F)$ iff $C_i(x) = 0$. Every problem in TFNP^{dt} is equivalent to $S(F)$ for some $\mathsf{polylog}(n)$-width CNF formula.

▶ **Definition 2.6.** *Let $R \subseteq \{0,1\}^n \times O$ and $S \subseteq \{0,1\}^m \times O'$ be total search problems. An S-*formulation *of R is a decision-tree reduction $(f_i, g_o)_{i \in [m], o \in O'}$ from R to S. Formally, for each $i \in [m]$ and $o \in O'$ there are functions $f_i \colon \{0,1\}^n \to \{0,1\}$ and $g_o \colon \{0,1\}^n \to O$ such that*

$$(x, g_o(x)) \in R \iff (f(x), o) \in S$$

where $f(x) \in \{0,1\}^m$ is the string whose i-th bit is $f_i(x)$. The depth *of the reduction is*

$$d \;:=\; \max\big(\{D(f_i) : i \in [m]\} \cup \{D(g_o) : o \in O'\}\big),$$

where $D(h)$ denotes the decision-tree depth of h. The size *of the reduction is m, the number of input bits to S. The* complexity *of the reduction is $\log m + d$. We write $S^{dt}(R)$ to denote the minimum complexity of an S-formulation of R.*

We extend these notations to sequences in the natural way. If R is a single search problem and $S = (S_m)$ is a sequence of search problems, then we denote by $S^{dt}(R)$ the minimum of $S^{dt}_m(R)$ over all m. If $R = (R_n)$ is also a sequence, then we denote by $S^{dt}(R)$ the function $n \mapsto S^{dt}(R_n)$.

Using the previous definition we can now define complexity classes of total search problems via reductions. For total search problems $\mathrm{R} = (R_n), \mathrm{S} = (S_n)$, we write

$$\mathrm{S}^{dt} \;:=\; \{\mathrm{R} : \mathrm{S}^{dt}(\mathrm{R}) = \mathsf{polylog}(n)\}.$$

Coloured TFNP Classes

With the definition of reductions established, we can define the search problems characterizing our coloured TFNP classes. We define the notation $[n]_0 := [n] \cup \{0\}$.

▶ **Definition 2.7** (**Coloured Sink-of-Dag**). C-SoD_n *is a total search problem defined on an $n \times n$ grid of nodes, where $(1, 1)$ is a special distinguished node. As input, we receive the following parameters for each node $(i, j) \in [n] \times [n]$:*

- *An index $s_{i,j} \in [n]_0$, indicating that the successor of (i, j) is $(i + 1, s_{i,j})$, or if $s_{i,j} = 0$, that (i, j) is a leaf.*
- *An indicator $c_{i,j,\lambda} \in \{0, 1\}\ \forall\lambda \in [n]$, indicating the presence of colours at each grid node*
- *An index $e_{i,j} \in [n]$, indexing a colour at each node*

Here the index $e_{i,j}$ is used to ensure that sinks can be efficiently verified to have a colour.

Any node on the final layer or any node with successor 0 is called a leaf and the node $(1, 1)$ is called the distinguished source. If the set of colours at each node contains the set of colours at its successor node, and there is at least one colour at each leaf, then clearly there must be at least one colour at the source node. The goal of the search problem is to find a witness of this fact. Formally, a solution to the C-SoD_n *search problem is*

- *$((i, j), \lambda)$ if $s_{i,j} = k$, $c_{i+1,k,\lambda} = 1$, and $c_{i,j,\lambda} = 0$ for some k.* (colour propagation)
- *$((1, 1), \lambda)$ if $c_{1,1,\lambda} = 1$* (distinguished source should be colourless)
- *$((i, j), \lambda)$ if (i, j) is a leaf, $e_{i,j} = \lambda$, and $c_{i,j,\lambda} = 0$* (sinks should have a colour)

▶ **Definition 2.8** (**Coloured Sink- and End-of-Line**). C-SoL_n *is a search problem defined on a set of n nodes, denoted $[n - 1]_0$, distinguishing the node 0. We define a graph on these nodes using the following parameters for each node $u \in [n - 1]_0$:*

- *An index $s_u \in [n - 1]_0$ indexing the successor of u.*
- *An index $p_u \in [n - 1]_0$ indexing the predecessor of u.*
- *An indicator $c_{u,\lambda} \in \{0, 1\}$ for each $\lambda \in [n]$, indicating the presence of the colour λ at u.*
- *An index $e_u \in [n]$, indexing a distinguished colour at each node.*

We define a graph G on $[n]$ by including an edge (u, v) if and only if $s_u = v$ and $p_v = u$. Again, if the set of colours at each node contains that at its successor, and there is at least one colour at each sink, then each source must contain at least one colour. The goal of the search problem is to find a witness of this. A pair (u, λ) is then a solution to an instance of C-SoL_n *if:*

- *$s_u = v$, $p_v = u$, $c_{v,\lambda} = 1$ and $c_{u,\lambda} = 0$ for some node $v \neq u$* (colour propagation)
- *$u = 0$ and $c_{0,\lambda} = 1$* (distinguished source should be colourless)
- *u is a sink node, $e_u = \lambda$, and $c_{u,\lambda} = 0$* (sinks should have a colour)

The C-EoL_n *problem is obtained by adding the following solutions to the* C-SoL_n *problem:*

- *u is a source node and $c_{u,\lambda} = 1$* (sources should be colourless)

▶ **Definition 2.9** (**Coloured Sink- and End-of-Potential-Line**). *The* C-SoPL_n *and* C-EoPL_n *problems are search problems combining the constraints of* C-SoD *and* C-EoL*. As with* C-SoD *they are defined on an $n \times n$ grid. We have the following parameters for each node $(i, j) \in [n] \times [n]$:*

- *An index $s_{i,j} \in [n - 1]_0$ indicating that the successor of (i, j) is $(i + 1, s_{i,j})$.*
- *An index $p_{i,j} \in [n - 1]_0$ indicating that the predecessor of (i, j) is $(i - 1, p_{i,j})$.*
- *An indicator $c_{i,j,\lambda} \in \{0, 1\}$ for each $\lambda \in [n]$ indicating the presence of the colour λ at (i, j).*
- *An index $e_{i,j} \in [n]$ indexing a distinguished colour at each node.*

As with C-SoL *and* C-EoL *we define a graph G on $[n]\times[n]$ by including an edge $((i,j),(i+1,k))$ if and only if $s_{i,j}=k$ and $p_{i+1,k}=j$. The solutions are then defined exactly as for* C-SoL$_n$ *and* C-EoL$_n$ *adapted to the $n\times n$ grid.*

We denote the TFNP^{dt} classes obtained by taking formulations of the above problems in San-Serif font, e.g. $\mathsf{C\text{-}SOPL}^{dt}=$ C-SoPL^{dt}.

3 The Blockwise Calculus

3.1 Multivalued CNFs and Blockwise Calculus Proofs

The proof-theoretic results in this paper have the unfortunate property of becoming technical when proved directly in the boolean proof systems defined in the previous section. To aid exposition we have found it useful to abstract out a generalized calculus – the *Blockwise Calculus* – to phrase our proofs in. In this section we introduce the Blockwise Calculus and prove our main technical results illustrating its relationship with the proof systems introduced in the previous section. Intuitively the Blockwise Calculus is the extension of Resolution to variables in a wider range than $\{0,1\}$.

▶ **Definition 3.1.** *A* multivalued variable *is a pair (x,n) where x is a formal variable and $n\in\mathbb{N}$ is a positive integer representing the range $[n-1]_0$ that the variable x can take values in. We will suppress the range parameter n when it is obvious from context. An* atom *is an expression of the form $[\![x\neq i]\!]$ where $i\in[n-1]_0$ is an element of the range. Given an $[n-1]_0$-assignment to x the atom evaluates to true iff the inequality inside the atom is satisfied. A* clause *is a disjunction ($\vee$) of atoms, where each variable in the clause can be quantified over its own range. The* width *of a clause C is the number of atoms in it.*

Using multivalued variables we can introduce the notion of a multivalued CNF formula.

▶ **Definition 3.2.** *Let $(x_1,r_1),(x_2,r_2),\dots,(x_n,r_n)$ be a collection of multivalued variables. A* multivalued CNF formula *$F=C_1\wedge\cdots\wedge C_m$ over these variables is a conjunction of clauses of atoms over the same variables. We say that F is* unsatisfiable *if there is no assignment of each variable to their respective ranges such that the resulting CNF is satisfied, and define the corresponding search problem $S(F)\subseteq[r_1-1]_0\times\cdots\times[r_n-1]_0\times[m]$ in the natural way: given a multivalued assignment to the corresponding variables, output a false clause of F.*

While the Blockwise Calculus operates on multivalued CNF formulas, we ultimately want to convert everything back to refutations in boolean logic. For this, we introduce the *booleanization* of a multivalued CNF, which is obtained by encoding each multivalued variable (x,r) in binary.

▶ **Definition 3.3.** *Let (x_i,r_i) for $i\in[n]$ be a collection of multivalued variables, and let $F=\bigwedge_{i=1}^m C_i$ be a multivalued CNF formula over these variables. The* booleanization *of F is the following CNF formula F_{bool}. For each variable (x_i,r_i) we introduce $t_i:=\lceil\log r_i\rceil$ boolean variables in a block, denoted $\vec{x}_i:=x_{i,1}\dots x_{i,t_i}\in\{0,1\}^{t_i}$, encoding the value of the variable x_i in binary. Then, for each clause in F we substitute each occurrence of an atom $[\![x_i\neq k]\!]$ with the disjunction on the variables $\vec{x}_i$ that is false exactly when $x_i=k$. Finally, for each $i\in[n]$ and each value $\ell\in[2^{t_i}]$ with $\ell\geq r_i$, we add a clause to F_{bool} over the variables $\vec{x}_i$ encoding that $x_i\neq\ell$.*

For each of the search problems defined in the previous section, there is a natural multivalued encoding of that search problem as an unsatisfiable multivalued CNF formula (cf. Section 3.2). Our current focus is to define the Blockwise Calculus and its variants. The rules of the Blockwise Calculus are shared among the three systems and defined in Figure 4, where C is a multivalued clause and (x, r) is a multivalued variable.

$$\textbf{(Reverse Cut)} \quad \frac{C}{C \vee [\![x \neq 0]\!] \quad C \vee [\![x \neq 1]\!] \cdots C \vee [\![x \neq r-1]\!]}$$

$$\textbf{(Cut)} \quad \frac{C \vee [\![x \neq 0]\!] \quad C \vee [\![x \neq 1]\!] \cdots C \vee [\![x \neq r-1]\!]}{C}$$

Figure 4 Proof Rules for the Blockwise Calculus.

▶ **Definition 3.4.** *Let F be a multivalued CNF formula and let C be a clause. A* Blockwise Calculus proof *of C from F is a sequence of clauses $C_1, C_2, \dots, C_s = C$ where the sequence is generated as follows. Starting from the empty sequence we either choose a clause from F to append to the sequence, or, we choose earlier clauses in the sequence and apply one of the Blockwise Calculus proof rules (cf. Figure 4) to generate new clauses and append them to the sequence. The* length *of the proof is s, the number of clauses, and the* width *of the proof is the size of the largest clause in the proof. The proof is a* refutation *if $C = \bot$, the empty clause. Finally, the proof is a* Reversible Blockwise Calculus proof *if every clause is used as the hypothesis of at most one proof rule.*

As in the case of Res(k), we can also introduce the *Circular* variant of Blockwise Calculus. The analogous rule we need to introduce is the following, for any multivalued clause C:

$$\frac{}{C} \quad \textbf{(Clause Creation)}$$

▶ **Definition 3.5.** *Let F be a multivalued CNF formula and let C be a clause. A* Circular Blockwise Calculus proof *of C from F is a sequence of clauses $C_1, C_2, \dots, C_s = C$ where the sequence is generated as follows. Starting from the empty sequence we can either choose a clause from F and append it to the end of the sequence, apply the Clause Creation rule to create an arbitrary clause C and append it to the sequence, or choose earlier clauses in the sequence and apply a Blockwise Calculus rule to generate new clauses and append them to the sequence. In addition, we make the following stipulations: each clause C_i in the sequence is used as the hypothesis of at most one Blockwise Calculus rule, and every clause C appearing in the proof is derived as the output of some proof rule more times than it is created using the Clause Creation rule. The length of this proof is s and the width of the proof is the maximum width of any clause C in the proof. The proof is a* refutation *if $C = \bot$.*

Similarly to the Res(k) systems, it is easy to see that Reversible Blockwise Calculus is a subsystem of both the Blockwise Calculus and the Circular Blockwise Calculus.

3.2 Encoding TFNP Problems as Multivalued CNFs

Given any of the total search problems introduced in Section 2, we can create a natural unsatisfiable multivalued CNF formula F expressing that the search problem has no solution. Intuitively, the negation of F encodes that the search problem is total.

Coloured Sink-of-Dag

We first show how to encode the Coloured Sink-of-Dag (C-SoD$_n$) problem. For each $i, j \in [n]$ we have a multivalued variable $(s_{ij}, n+1)$ expressing that the pointer of the node s_{ij} is either 0 or points to a node on the next level. For each $i, j \in [n]$ and each $\lambda \in [n]$ we have a multivalued variable $(c_{i,j,\lambda}, 2)$ expressing whether or not the colour λ is present at node (i, j). Finally, for each $i, j \in [n]$ we have a second multivalued variable (e_{ij}, n) indexing a colour at that node. We can now phrase the totality of the search problem using the following unsatisfiable multivalued CNF formula C-SoD, containing the following clauses:

- **Colourless Distinguished Source.** For each $\lambda \in [n-1]_0$, $[\![c_{11\lambda} \neq 1]\!]$.
- **Propagating Colours.** For each $i \in [n-1]$, each $j, k \in [n]$, and each $\lambda \in [n-1]_0$,

$$[\![s_{ij} \neq k]\!] \vee [\![c_{i+1,k,\lambda} \neq 1]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!] \,.$$

- **Coloured Sinks.** For each $i \in [n-1], j \in [n], \lambda \in [n-1]_0$,

$$[\![s_{ij} \neq 0]\!] \vee [\![e_{ij} \neq \lambda]\!] \vee [\![c_{ij\lambda} \neq 0]\!] \,, \text{ and}$$

$$[\![e_{nj} \neq \lambda]\!] \vee [\![c_{nj\lambda} \neq 0]\!] \,.$$

Coloured Sink-of-Line and Coloured End-of-Line

The variables of both C-SoL$_n$ and C-EoL$_n$ are the same, but the two formulas differ on their defining constraints. For each $u, \lambda \in [n-1]_0$ we have multivalued variables $(s_u, n), (p_u, n), (e_u, n)$, and $(c_{u,\lambda}, 2)$ encoding successor pointers, predecessor pointers, colour pointers, and colours for each node. The nodes range in the set $[n-1]_0$ and we treat 0 as the distinguished source node. The clauses of the C-SoL$_n$ formula are defined as follows:

- **Colourless Distinguished Source.** For each $\lambda \in [n-1]_0$, $[\![c_{0,\lambda} \neq 1]\!]$.
- **Colour Propagation.** For each $u \neq v \in [n-1]_0$ and each $\lambda \in [n]$,

$$[\![s_u \neq v]\!] \vee [\![p_v \neq u]\!] \vee [\![c_{u,\lambda} \neq 0]\!] \vee [\![c_{v,\lambda} \neq 1]\!] \,.$$

- **Coloured Sinks.** For each $u, v, w, \lambda \in [n-1]_0$ with $u \neq w$:

$$[\![s_u \neq v]\!] \vee [\![p_v \neq w]\!] \vee [\![e_u \neq \lambda]\!] \vee [\![c_{u,\lambda} \neq 0]\!] \,.$$

The C-EoL$_n$ formula adds the following clauses to the C-SoL$_n$ formula:

- **Colourless Sources.** For each $u \in [n-1]$, $v, w, \lambda \in [n-1]_0$ with $u \neq w$:

$$[\![p_u \neq v]\!] \vee [\![s_v \neq w]\!] \vee [\![c_{u,\lambda} \neq 1]\!] \,.$$

Coloured Sink-of-Potential-Line and Coloured End-of-Potential-Line

The variables of C-SoPL$_n$ and C-EoPL$_n$ are the same. For each $i, j \in [n-1]_0$ and each $\lambda \in [n-1]_0$ we have variables $(s_{ij}, n), (p_{ij}, n), (e_{ij}, n), (c_{ij\lambda}, n)$ encoding successor pointers, predecessor pointers, colour pointers, and colours for each node. The clauses of the C-SoPL$_n$ formula are defined as follows:

- **Colourless Distinguished Source.** For each $\lambda \in [n-1]_0$, $[\![c_{0,0,\lambda} \neq 1]\!]$.
- **Colour Propagation.** For each $i \in [n-2]_0, j, k \in [n-1]_0$ and each $\lambda \in [n-1]_0$,

$$[\![s_{ij} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!] \vee [\![c_{i+1,k,\lambda} \neq 1]\!] \,.$$

- **Coloured Sinks.** For each $i \in [n-2]_0, j, k, \ell \in [n-1]_0$ with $\ell \neq j$ and $\lambda \in [n-1]_0$

$$[\![s_{ij} \neq k]\!] \vee [\![p_{i+1,k} \neq \ell]\!] \vee [\![e_{i,j} \neq \lambda]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!]\,, \text{ and}$$

$$[\![e_{n-1,j} \neq \lambda]\!] \vee [\![c_{n-1,j,\lambda} \neq 0]\!]\,.$$

The C-EoPL$_n$ formula adds the following clauses to the C-SoPL$_n$ formula:

- **Colourless Sources.** For each $i \in [n-1]$, $j, k, \ell, m \in [n-1]_0$ with $m \neq j$, and each $\lambda \in [n-1]_0$

$$[\![p_{ij} \neq \ell]\!] \vee [\![s_{i-1,\ell} \neq m]\!] \vee [\![s_{ij} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee [\![c_{ij\lambda} \neq 1]\!]$$

  and

$$[\![s_{0,j} \neq k]\!] \vee [\![p_{1,k} \neq j]\!] \vee [\![c_{0,j,\lambda} \neq 1]\!]\,.$$

3.3 Blockwise Calculus vs. Boolean Proof Systems

In this section we prove the main technical theorem necessary for our main results. Essentially, it says that if we have a refutation of a formula F in the Blockwise Calculus or one if its variations, then we can automatically obtain a refutation of any F-formulation in a related boolean proof system.

▶ **Theorem 3.6.** *Let F, G be any width-c multivalued CNF formulas for which there is a depth-d $S(F)$-formulation of $S(G)$. Then*

- *If there is a size-s Blockwise Calculus refutation of F, then there is a size-$s2^{O(d)}$ $\mathrm{Res}(O(d))$-refutation of G_{bool}.*
- *If there is a size-s Circular Blockwise Calculus refutation of F, then there is a size-$s2^{O(d)}$ $\mathrm{uCircRes}(O(d))$-refutation of G_{bool}.*
- *If there is a size-s Reversible Blockwise Calculus refutation of F, then there is a size-$s2^{O(d)}$ $\mathrm{RevRes}(O(d))$-refutation of G_{bool}.*

Proving this theorem is much easier after we introduce some auxiliary technical lemmas for working with decision trees. For a given decision tree T, let $P_\ell(T)$ denote the set of all root-leaf paths ending in a leaf labelled by ℓ, and let $P(T) := \bigcup_\ell P_\ell(T)$. For a given path $p \in P(T)$ let $C_p := x_1 \wedge \cdots \wedge x_k$ and $\overline{C}_p := \overline{x}_1 \vee \cdots \vee \overline{x}_k$ where $x_1, ..., x_k$ are the queries made along p.

Let $D_T := \bigvee_{p \in P(T)} C_p$, intuitively encoding the fact that some branch of a decision tree must be followed under an input. We will rely on these formulas heavily, so we now demonstrate that they can be efficiently derived in $\mathrm{Res}(k)$. It is enough to prove this next lemma for $\mathrm{RevRes}(d)$ since both $\mathrm{Res}(d)$ and $\mathrm{uCircRes}(d)$ simulate $\mathrm{RevRes}(d)$.

▶ **Lemma 3.7.** *If T is a decision tree of depth d, then there is a size-2^{2d} $\mathrm{RevRes}(d)$ proof of the formula $\bigvee_{p \in P(T)} C_p$.*

Proof. If T consists of a single query of some literal ℓ, then $D_T = \ell \vee \overline{\ell}$, which can be derived in a single line as an axiom. Otherwise we proceed by induction, so let ℓ be the first literal queried by T. Let T_0 be the subtree of T followed when ℓ is falsified, and T_1 be the one followed when ℓ is satisfied. By induction, we can derive D_{T_0} and D_{T_1} with size $2 \cdot 2^{2(d-1)} = 2^{2d-1}$. We begin by $\vee$-weakening T_0 with ℓ and T_1 with $\overline{\ell}$, and introducing the axiom $\ell \vee \overline{\ell}$.

Now let $p_1, ..., p_k$ be the paths of T_0. Weaken the axiom $\ell \vee \overline{\ell}$ by C_{p_i} for all $1 < i \leq k$ to obtain $\bigvee_{1<i\leq k} C_{p_i} \vee \ell \vee \overline{\ell}$. $\wedge$-introducing this with $D_{T_0} \vee \overline{\ell}$ on C_{p_1} and $\overline{\ell}$, we obtain:

$$\bigvee_{1<i\leq k} C_{p_i} \vee C_{p_1 \cup \overline{\ell}} \vee \ell$$

Now weaken $\ell \vee \overline{\ell}$ by C_{p_i} for all $2 < i \leq k$, and by $C_{p_i \cup \overline{\ell}}$ for $1 \leq i < 2$ to obtain $\bigvee_{2<i\leq k} C_{p_i} \vee \bigvee_{1\leq i<2} C_{p_i \cup \overline{\ell}} \vee \ell \vee \overline{\ell}$. We again $\wedge$-introduce this, this time with $\bigvee_{1<i\leq k} C_{p_i} \vee C_{p_i \cup \overline{\ell}} \vee \ell$ on C_{p_2} and $\overline{\ell}$ to obtain:

$$\bigvee_{2<i\leq k} C_{p_i} \vee \bigvee_{1\leq i\leq 2} C_{p_i \cup \overline{\ell}} \vee \ell$$

Repeating this for the remaining paths p_j for $2 < j \leq k$, we obtain:

$$\bigvee_{p\in P(T_0)} C_{p \cup \overline{\ell}} \vee \ell$$

and we can repeat this process for T_1 to likewise derive:

$$\bigvee_{p\in P(T_1)} C_{p \cup \ell} \vee \overline{\ell}$$

Since $P(T) = \bigcup_{p\in P(T_0)}(p \cup \overline{\ell}) \cup \bigcup_{p\in P(T_1)}(p \cup \ell)$, we can finally cut these two formulas on ℓ to obtain $\bigvee_{p\in P(T)} C_p = D_T$. All conjunctions created in this process have width at most d, as they each correspond to a path or subpath of a path of T, and since there are 2^{d-1} paths in each subtree this process adds an additional $2 \cdot (2^{d-1})^2 = 2^{2d-1}$ lines to the proof. Thus the total size of the proof is $2 \cdot 2^{2d-1} = 2^{2d}$. Further, since all root–leaf paths are bounded in length by d, the proof has width $O(d)$. ◀

We now show that cutting and weakening along negated paths of decision trees can be done inside of Reversible Resolution.

▶ **Lemma 3.8.** *Let C be a width-w clause and let T be a depth-d decision tree. Then there is a size-2^d, width-$(w+d)$* RevRes *derivation of C from the set of clauses $\{C \vee \overline{C}_p \mid p \in P(T)\}$ and* vice-versa.

Proof. We proceed by induction on d. In the base case, $d = 1$ and a single variable x is queried by T; in this case we have the formulas $C \vee x$ and $C \vee \overline{x}$ and we resolve on x to obtain C.

By induction suppose that the claim holds for a decision tree of depth at most $d - 1$. Let T_0 be the decision tree obtained by discarding all leaves of T (the new leaves may be labelled arbitrarily). For each path $p \in P(T)$, let x be the final variable it queries, let q be the path of T which differs from p only on x, and let p_0 be the path of T_0 obtained by truncating p before x. Then we may cut the formulas $C \vee \overline{p}_0 \vee x$ and $C \vee \overline{p}_0 \vee \overline{x}$, corresponding to p and q, on x to obtain $C \vee \overline{p}_0$. Repeating this for each such pair of paths in T yields $C \vee \overline{p}_0$ for each $p_0 \in T_0$ in 2^{d-1} steps, allowing us to apply the induction hypothesis to complete the derivation of C in a further 2^{d-1} steps, for a total size of 2^d as desired. Furthermore, this is reversible, as each path of T belongs to a single such pair. The width claims are also clear as all formulas are of the form $C \vee \overline{p}$ for some path p of a depth-d decision tree. ◀

Now, let F be a multivalued CNF formula on variables (x_i, r_i) for $i \in [n]$, let G be a multivalued CNF formula on variables (y_i, s_i) for $i \in [m]$, and suppose that we have a depth-d $S(F)$-formulation of $S(G)$. This means that each variable x_i is computed by a depth-d decision tree f_i which queries variables y_j and outputs a value in $[r_i - 1]_0$, and we also have, for each clause C in F, a decision tree g_C which queries y_j variables and outputs a clause of $S(G)$. For simplicity, we will identify the variable x_i with its decision tree f_i that computes it.

Suppose that we have a Blockwise Calculus refutation Π of F. Our goal is to give a Res$(O(d))$ refutation of G. In order to prove this theorem we need to encode atoms $[\![x_i \neq j]\!]$ into boolean formulas. We introduce two such encodings: the *positive* and *negative* encoding. In the positive encoding we encode each atom as a d-DNF formula, while in the negative encoding we encode the atom as a family of width-d clauses. We emphasize that in the definitions below we identify the variable x_i of the formula F with the decision tree f_i outputting the value of x_i in the reduction from G.

$$\mathcal{D}^+([\![x_i \neq j]\!]) := \bigvee_{k \neq j} \bigvee_{p \in P_k(x_i)} C_p$$

$$\mathcal{D}^-([\![x_i \neq j]\!]) := \{\overline{C}_p : p \in P_j(x_i)\}$$

If C is a clause over multivalued atoms we write $\mathcal{D}^+(C)$ to denote the DNF formula obtained by substituting each atom A in C with its positive encoding $\mathcal{D}^+(A)$, and write $\mathcal{D}^-(C)$ to denote the CNF formula obtained by substituting $\bigwedge \mathcal{D}^-(A)$ for each atom in C and then re-writing the result in CNF by distributing the $\vee$ over the $\wedge$s.

The next lemma is arguably the main technical lemma used in the proof of Theorem 3.6. It shows that it is possible to derive positive encodings of multivalued clauses from negative encodings and vice-versa efficiently in RevRes(d).

▶ **Lemma 3.9.** *Suppose that x is computed by a depth-d decision tree and G is a DNF. Then there is a* RevRes(d) *proof of all the DNFs in $\{G \vee C \mid C \in \mathcal{D}^-([\![x \neq j]\!])\}$ from $G \vee \mathcal{D}^+([\![x \neq j]\!])$ and vice-versa in size $|G|^2 \cdot 2^{O(d)}$*

Proof. We begin by proving $G \vee \mathcal{D}^+([\![x \neq j]\!])$ from $\{G \vee C \mid C \in \mathcal{D}^-([\![x \neq j]\!])\}$. This direction is simpler. By applying Lemma 3.7 we can derive the DNF $\bigvee_{p \in P(x)} C_p$ in size 2^{2d} from axioms, and then by applying reverse cut repeatedly we can derive $G \vee \bigvee_{p \in P(x)} C_p$ in size $O(|G|^2 2^d)$. From $G \vee \bigvee_{p \in P(x)} C_p$ we can repeatedly cut on $G \vee C$ for each $C \in \mathcal{D}^-([\![x \neq j]\!])$ to in sequence to derive $G \vee \mathcal{D}^+([\![x \neq j]\!])$. The total size is $|G|^2 2^{O(d)}$.

We now prove the other direction. Without loss of generality suppose that $j = 0$ and let $\mathcal{D} := G \vee \mathcal{D}^+([\![x \neq 0]\!])$ for the sake of brevity. By definition we have $\mathcal{D} = G \vee \bigvee_{k \neq j} \bigvee_{p \in P_k(x)} C_p$. Let $\mathcal{P} = \bigcup_{k \neq 0} P_k(x)$ denote the set of all paths appearing in the above disjunction and write $\mathcal{P} = \{p_1, p_2, \ldots, p_s\}$.

We begin by applying reverse cut repeatedly along the variables in the decision tree computing x to derive the set of DNFs $\{\mathcal{D} \vee \overline{C}_q \mid q \in P(x)\}$. Fix an arbitrary path $q \in P_0(x)$. For each path $p_i \in \mathcal{P}$ there is a literal ℓ_i such that ℓ_i is queried positively in p_i and negatively in q. Therefore, by using an axiom-introduction we can introduce the clause $\ell_1 \vee \overline{\ell}_1$ and then repeatedly using reverse-cut we can derive $G \vee \bigvee_{i=2}^s C_{p_i} \vee \overline{C}_q \vee \overline{C}_{p_1}$ We can then cut this result with $\mathcal{D} \vee \overline{C}_q$ to derive $G \vee \bigvee_{i=2}^s C_{p_i} \vee \overline{C}_q$. We can now repeat this process: there is another literal ℓ_2 appearing positively in p_2 and negatively in q, and thus we can axiom-introduce $\ell_2 \vee \overline{\ell}_2$ and then use reverse cut to derive $G \vee \bigvee_{i=3}^s C_{p_i} \vee \overline{C}_{p_2} \vee \overline{C}_q$. Cutting this with the result of the previous stage yields $G \vee \bigvee_{i=3}^s C_{p_i} \vee \overline{C}_q$, and we can repeat this process s times in order to derive $G \vee \overline{C}_q$. We can then repeat this for each $q \in P_0(x)$ to derive $\mathcal{D}^-([\![x_i \neq j]\!])$.

We now estimate the size of the derivation. The first line has size at most $|G| + 2^d$, and we begin by deriving a set of 2^d DNFs, each of size $O(|G| + 2^d)$, and thus the cost of the first step is $O(|G|2^{2d})$. To cut each of the paths $C_{p_1}, C_{p_2}, \dots, C_{p_s}$ we must pay $O(|G|^2 2^d)$ to derive the corresponding DNF to cut our preserved formula with, and this will repeat $s \le 2^d$ times, for a total cost of $O(|G|^2 2^{2d})$. Finally, we must repeat this entire process $\le 2^d$ times for each $q \in P_0(x)$, and thus the final size is $O(|G|^2 2^{3d}) = |G|^2 2^{O(d)}$. ◀

Using the lemma we can now prove Theorem 3.6.

Proof of Theorem 3.6. The basic idea of this proof is simple: for each clause $C \in \Pi$ we replace C with the width-d DNF encoding $\mathcal{D}^+(C)$, noting that the final clause $\perp$ remains empty. We prove two claims:

Claim 1. For each clause C in F we can deduce $\mathcal{D}^+(C)$ from the clauses of G_{bool} in $\mathrm{RevRes}(O(d))$.

Claim 2. For each proof rule of the Blockwise Calculus we can deduce the positive encodings of each consequent of the rule from the positive encodings of each antecedent of the rule efficiently in $\mathrm{RevRes}(O(d))$.

To prove the first claim, let $F = C_1 \wedge \cdots \wedge C_s$ and $G_{bool} = C'_1 \wedge \cdots \wedge C'_t$, let (x_i, r_i) for $i \in [n]$ denote the variables of F, and let $\vec{y}_1, \dots, \vec{y}_m$ denote the (boolean block) variables of G_{bool}. By the definition of an $S(F)$-formulation, for each variable (x_i, r_i) of F we have a depth-d decision tree f_i querying variables of G_{bool} and outputting a value for x_i, as well as a decision tree g_k for each $k \in [s]$ such that $(f(y), k) \in S(F) \Rightarrow (y, g_k(y)) \in S(G)$. We can interpret this definition in terms of proofs as follows. Let $C_k = A_1 \vee \cdots \vee A_w$ be any clause of F and assume w.l.o.g. that $A_i := [\![x_i \neq \ell_i]\!]$ for some ℓ_i. For each $i \in [w]$ let $p_i \in P_{\ell_i}(x_i)$ be any path in the corresponding decision tree from the formulation outputting ℓ_i, and let $q \in P(g_k)$ be any path in the tree g_k. Then the clause $\bigvee_{i=1}^{w} \overline{C}_{p_i} \vee \overline{C}_q$ is a weakening of clause of G. Since there are at most 2^d paths in each decision tree and the width of C_k is w it follows that there are at most $2^{wd} \le 2^{cd}$ such clauses, and each can be deduced from clauses of G using weakening rules. Next, we observe that from the collection of clauses $\{\bigvee_{i=1}^{w} \overline{C}_{p_i} \vee \overline{C}_q \mid q \in P(g_k)\}$ we can use reversible cuts up the decision tree g_k in order to deduce the family of clauses $\{\bigvee_{i=1}^{w} \overline{C}_{p_i}\}$, and taking the union over all such paths $p_i \in \mathcal{P}_{\ell_i}(x_i)$ yields exactly $\mathcal{D}^-(C_k)$. Finally, applying Lemma 3.9 yields $\mathcal{D}^+(C_k)$. Applying this strategy to all clauses of F we can deduce $\mathcal{D}^+(C_k)$ for each clause of F, as desired.

We move on to proving the second claim. We first consider the Cut rule

$$\frac{C \vee [\![x_i \neq 0]\!] \quad C \vee [\![x_i \neq 1]\!] \quad \cdots \quad C \vee [\![x_i \neq r_i - 1]\!]}{C}$$

for which we need to show how to derive $\mathcal{D}^+(C)$ from $\mathcal{D}^+(C) \vee \mathcal{D}^+([\![x_i]\!] \neq \ell)$ for each $\ell \in [r_i - 1]_0$. We can apply Lemma 3.9 to $\mathcal{D}^+(C) \vee \mathcal{D}^+([\![x_i \neq \ell]\!])$ for each $\ell = 0, 1, \dots, r_i - 1$ in order to derive the family

$$\bigcup_{\ell=0}^{r_i - 1} \{\mathcal{D}^+(C) \vee D \mid D \in \mathcal{D}^-([\![x \neq \ell]\!])\} = \bigcup_{p \in P(x_i)} \{\mathcal{D}^+(C) \vee \overline{C}_p\}.$$

From this family we can apply Lemma 3.8 in order to derive $\mathcal{D}^+(C)$, as desired.

We now consider the Reverse Cut rule

$$\frac{C}{C \vee [\![x_i \neq 0]\!] \quad C \vee [\![x_i \neq 1]\!] \quad \cdots \quad C \vee [\![x_i \neq r_i - 1]\!]}.$$

Starting from $\mathcal{D}^+(C)$ we must derive the family $\{\mathcal{D}^+(C) \vee \mathcal{D}^+(\llbracket x_i \neq \ell \rrbracket) \mid \ell \in [r_i - 1]_0\}$. This direction is easy: since this rule is the reverse of the previous rule, and since we gave a RevRes(d) simulation of the previous rule, running the previous construction in reverse handles this case as well.

Using the two claims we can now complete the proof of the theorem. For Res(d) and RevRes(d) the result follows immediately by induction over the proof Π. For Circular Res(d) we can similarly apply induction over Π, additionally observing that if we ever apply the Clause Creation rule in a Circular Blockwise Calculus proof to create a clause C, we can simply apply the DNF creation rule in Circular Res(d) to create $\mathcal{D}^+(C)$. Since the Circular Blockwise Calculus proof must derive each clause C more times than it is introduced by a Clause Creation rule, the same property holds for the uCircRes(d) proof. This completes the proof of the theorem. ◀

A similar result can also be obtained for low-width Resolution, which we will use to show collapses to uncoloured classes. The main difference in this proof is that we do not use the positive encoding, only the negative encoding.

▶ **Theorem 3.10.** *Let F, G be any multivalued CNF formulas for which there is a depth-d $S(F)$-formulation of $S(G)$. Then*

- *If there is a size-s, width-$\log^{O(1)} s$ Blockwise Calculus refutation of F, then there is a size-$s^{O(1)}2^{O(d)}$, width-$d \cdot \log^{O(1)} s$ Resolution refutation of G_{bool}.*
- *If there is a size-s, width-$\log^{O(1)} s$ Circular Blockwise Calculus refutation of F, then there is a size-$s^{O(1)}2^{O(d)}$, width-$d \cdot \log^{O(1)} s$* uCircRes-*refutation of G_{bool}.*
- *If there is a size-s, width-$\log^{O(1)} s$ Reversible Blockwise Calculus refutation of F, then there is a size-$s^{O(1)}2^{O(d)}$, width-$d \cdot \log^{O(1)} s$* RevRes-*refutation of G_{bool}.*

Proof. Let Π be the size-s, width-$\log^{O(1)} s$ Blockwise Calculus refutation (potentially reversible or circular) of F. We construct a Resolution refutation of G from Π By first proving $\mathcal{D}^-(F) := \bigwedge_{C \in F} \mathcal{D}^-(C)$, then converting Π into a refutation of $\mathcal{D}^-(F)$, so we may finally combine these proofs into a refutation of G. Again, this requires us to show the following:

- For each clause C of F, there is an efficient proof of $\mathcal{D}^-(C)$ from G
- Each rule of the blockwise calculus can be efficiently simulated by Reversible Resolution using the negative encoding of blocks

Let $C_1, ..., C_s$ denote the clauses of F, and $C'_1, ..., C'_t$ the clauses of G_{bool}. Likewise, dnote the variables of F by $(x_1, r_1), ..., (x_n, r_n)$ and the variables of G_{bool} by $\vec{y}_1, ..., \vec{y}_m$. For each variable x_i of F, we have a depth-d decision tree f_i decision tree in the formulation over variables of G_{bool} computing it, and for each clause C_i of F, we have a depth-d decision tree g_i outputting a corresponding clause of G. By definition of a $S(F)$-formulation then, for each such clause C_i, each clause $C' \in \mathcal{D}^-(C_i)$, and each path $p \in P(g_i)$, the clause $C' \vee \overline{C}_p$ is a weakening of at least one clause of G – if C' and C_p are both falsified under some assignment $\vec{a}$ to the variables of G, then so too must the clause output by p be falsified under $\vec{a}$. Thus, for each such C', we can derive the clause $C' \vee \overline{C}_p$ from G for every $p \in P(g_i)$, at which point we may apply Lemma 3.8 to obtain $\mathcal{D}^-(C_i)$. Repeating for all clauses of F yields $\mathcal{D}^-(F)$.

We proceed now to show that we can simulate the rules of the blockwise calculus in Reversible Resolution:

- If some clause C was derived by cutting earlier clauses $C \vee \llbracket x \neq 0 \rrbracket, ..., C \vee \llbracket x \neq n-1 \rrbracket$, then we have the set of clauses $C' \vee \overline{p}$ for each $C' \in \mathcal{C}$ and $p \in P(T_x)$, from which we wish to derive each $C' \in \mathcal{C}$. Thus, by Lemma 3.8 this can be done in 2^d steps with width $d + d \cdot \log^{O(1)} s$.

- If C was derived by weakening some earlier clause C_0 on some variable x, then begin with each $C' \in \mathcal{C}$, from which we wish to derive $C' \vee \overline{p}$ for each $C' \in \mathcal{C}$ and $p \in P(T_x)$. By reversibility of Lemma 3.8, this can be done in 2^d steps with width $d + d \cdot \log^{O(1)} s$.

Since all families of clauses $\mathcal{C}$ corresponding to an original clause C of Π have size $s^{O(1)} 2^{O(d)}$ and each new clause requires 2^d additional steps to derive, this results in a proof of size $n^{O(1)} 2^{O(d)}$ overall. Furthermore, all clauses in the new proof consist of $\log^{O(1)} s$ negated paths of depth-d decision trees, and thus have width $d \cdot \log^{O(1)} s$ overall. ◀

References

1 Albert Atserias and Maria Luisa Bonet. On the automatizability of resolution and related propositional proof systems. *Inf. Comput.*, 189(2):182–201, 2004. doi:10.1016/j.ic.2003.10.004.

2 Albert Atserias and Massimo Lauria. Circular (yet sound) proofs. In *Proceedings of the 22nd Theory and Applications of Satisfiability Testing (SAT)*, pages 1–18. Springer, 2019. doi:10.1007/978-3-030-24258-9_1.

3 Paul Beame, Stephen Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences*, 57(1):3–19, 1998. doi:10.1006/jcss.1998.1575.

4 Paul Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, and Pavel Pudlák. Lower bounds on Hilbert's Nullstellensatz and propositional proofs. In *Proceedings of the 35th Symposium on Foundations of Computer Science (FOCS)*, pages 794–806, 1994. doi:10.1109/SFCS.1994.365714.

5 Arnold Beckmann and Samuel R. Buss. Characterising definable search problems in bounded arithmetic via proof notations. In *Ways of Proof Theory*, ONTOS Series in Mathematical Logic, pages 65–134, 2010.

6 Ilario Bonacina and Maria Luisa Bonet. On the strength of sherali-adams and nullstellensatz as propositional proof systems. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 25:1–25:12. ACM, 2022. doi:10.1145/3531130.3533344.

7 María Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 171(8-9):606–618, 2007. doi:10.1016/j.artint.2007.03.001.

8 Joshua Buresh-Oppenheim and Tsuyoshi Morioka. Relativized NP search problems and propositional proof systems. In *Proceedings of the 19th IEEE Conference on Computational Complexity (CCC)*, pages 54–67, 2004. doi:10.1109/CCC.2004.1313795.

9 Sam Buss, Noah Fleming, and Russell Impagliazzo. Tfnp characterizations of proof systems and monotone circuits. *Electron. Colloquium Comput. Complex.*, TR22-141, 2022. arXiv:TR22-141.

10 Samuel R. Buss and Alan S. Johnson. Propositional proofs and reductions between NP search problems. *Annals of Pure and Applied Logic*, 163(9):1163–1182, 2012. doi:10.1016/j.apal.2012.01.015.

11 Xi Chen, Decheng Dai, Ye Du, and Shang-Hua Teng. Settling the complexity of Arrow-Debreu equilibria in markets with additively separable utilities. In *Proceedings of the 50th Symposium on Foundations of Computer Science (FOCS)*, pages 273–282, 2009. doi:10.1109/FOCS.2009.29.

12 Xi Chen, Dimitris Paparas, and Mihalis Yannakakis. The complexity of non-monotone markets. *Journal of the ACM*, 64(3):20:1–20:56, 2017. doi:10.1145/3064810.

13 Bruno Codenotti, Amin Saberi, Kasturi Varadarajan, and Yinyu Ye. The complexity of equilibria: Hardness results for economies via a correspondence with games. *Theoretical Computer Science*, 408(2–3):188–198, 2008. doi:10.1016/j.tcs.2008.08.007.

14 Stefan S. Dantchev, Barnaby Martin, and Mark Nicholas Charles Rhodes. Tight rank lower bounds for the sherali-adams proof system. *Theor. Comput. Sci.*, 410(21-23):2054–2063, 2009. doi:10.1016/j.tcs.2009.01.002.

15 Constantinos Daskalakis, Paul Goldberg, and Christos Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009. doi:10.1137/070699652.

16 John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: CLS = PPAD $\cap$ PLS. In *Proceedings of the 53rd Symposium on Theory of Computing (STOC)*, pages 46–59, 2021. doi:10.1145/3406325.3451052.

17 Yuval Filmus, Meena Mahajan, Gaurav Sood, and Marc Vinyals. MaxSAT resolution and subcube sums. In *Proceedings of the 23rd Theory and Applications of Satisfiability Testing (SAT)*, pages 295–311. Springer, 2020. doi:10.1007/978-3-030-51825-7_21.

18 Michal Garlík. Failure of feasible disjunction property for k-dnf resolution and np-hardness of automating it. *CoRR*, abs/2003.10230, 2020. arXiv:2003.10230.

19 Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Further collapses in TFNP. In *Proceedings of the 37th Computational Complexity Conference (CCC)*, pages 33:1–33:15, 2022. doi:10.4230/LIPICS.CCC.2022.33.

20 Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Separations in proof complexity and TFNP. *Electron. Colloquium Comput. Complex.*, TR22-058, 2022. arXiv:TR22-058.

21 Mika Göös, Pritish Kamath, Robert Robere, and Dmitry Sokolov. Adventures in monotone complexity and TFNP. In *Proceedings of the 10th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 124, pages 38:1–38:19, 2018. doi:10.4230/LIPIcs.ITCS.2019.38.

22 David Johnson, Christos Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.

23 Jan Krajíček. On the weak pigeonhole principle. *Fundamenta Mathematicae*, 170(1-3):123–140, 2001.

24 Jan Krajícek, Alan Skelley, and Neil Thapen. NP search problems in low fragments of bounded arithmetic. *J. Symb. Log.*, 72(2):649–672, 2007. doi:10.2178/jsl/1185803628.

25 Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, 172(2-3):204–233, 2008. doi:10.1016/j.artint.2007.05.006.

26 Nimrod Megiddo and Christos Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991. doi:10.1016/0304-3975(91)90200-L.

27 Christos Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994. doi:10.1016/s0022-0000(05)80063-7.

28 Nathan Segerlind, Samuel R. Buss, and Russell Impagliazzo. A switching lemma for small restrictions and lower bounds for k-dnf resolution. *SIAM J. Comput.*, 33(5):1171–1200, 2004. doi:10.1137/S0097539703428555.

29 Hanif Sherali and Warren Adams. A hierarchy of relaxations and convex hull characterizations for mixed-integer zero–one programming problems. *Discrete Applied Mathematics*, 52(1):83–106, July 1994. doi:10.1016/0166-218x(92)00190-w.

30 Alan Skelley and Neil Thapen. The provably total search problems of bounded arithmetic. *Proceedings of the London Mathematical Society*, 103(1):106–138, 2011.

31 Neil Thapen. A tradeoff between length and width in resolution. *Theory Comput.*, 12(1):1–14, 2016. doi:10.4086/toc.2016.v012a005.