Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 68 (2023)

Colourful TFNP and Propositional Proofs

Ben Davis and Robert Robere
School of Computer Science
McGill University

July 18, 2023

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 (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 C-PLS$\cap$C-PPADS = C-SOPL, generalizing the intersection theorem PLS$\cap$PPADS = SOPL. However, while it is known in the uncoloured world that PLS $\cap$ PPAD = EOPL = 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.

ISSN 1433-8092

Contents

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* [JPY88, Pap94] to the theory of *propositional proof complexity.* A total NP search problem is a search problem S satisfying:

- **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 [MP91]. 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 [Pap94]. 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 [DGP09] along with other important problems in economics [CSVY08, CDDT09, CPY17].

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. [BCE+98] 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 $\mathsf{A}^{O} \not\subseteq \mathsf{B}^{O}$ [BCE+98]. Beame et al. [BCE+98] 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. [BCE+98] 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 [BM04], 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 [BJ12, GKRS18, GHJ+22b, BFI22]. One surprising outcome of the emerging work is that: not only can proof complexity lower bounds be used to construct oracle separations (as in [BCE+98]) 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. $\mathsf{A} = \mathsf{PLS}$, PPAD, PPADS, $\mathsf{CLS} = \mathsf{EOPL}$, 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 [GHJ+22b] 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 $\mathsf{CLS} = \mathsf{PLS} \cap \mathsf{PPAD}$ [FGHS21] and its followups $\mathsf{EOPL} = \mathsf{PLS} \cap \mathsf{PPAD}$ and $\mathsf{SOPL} = \mathsf{PLS} \cap \mathsf{PPADS}$ [GHJ+22a] 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 [GHJ+22b], 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 [BFI22] 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.

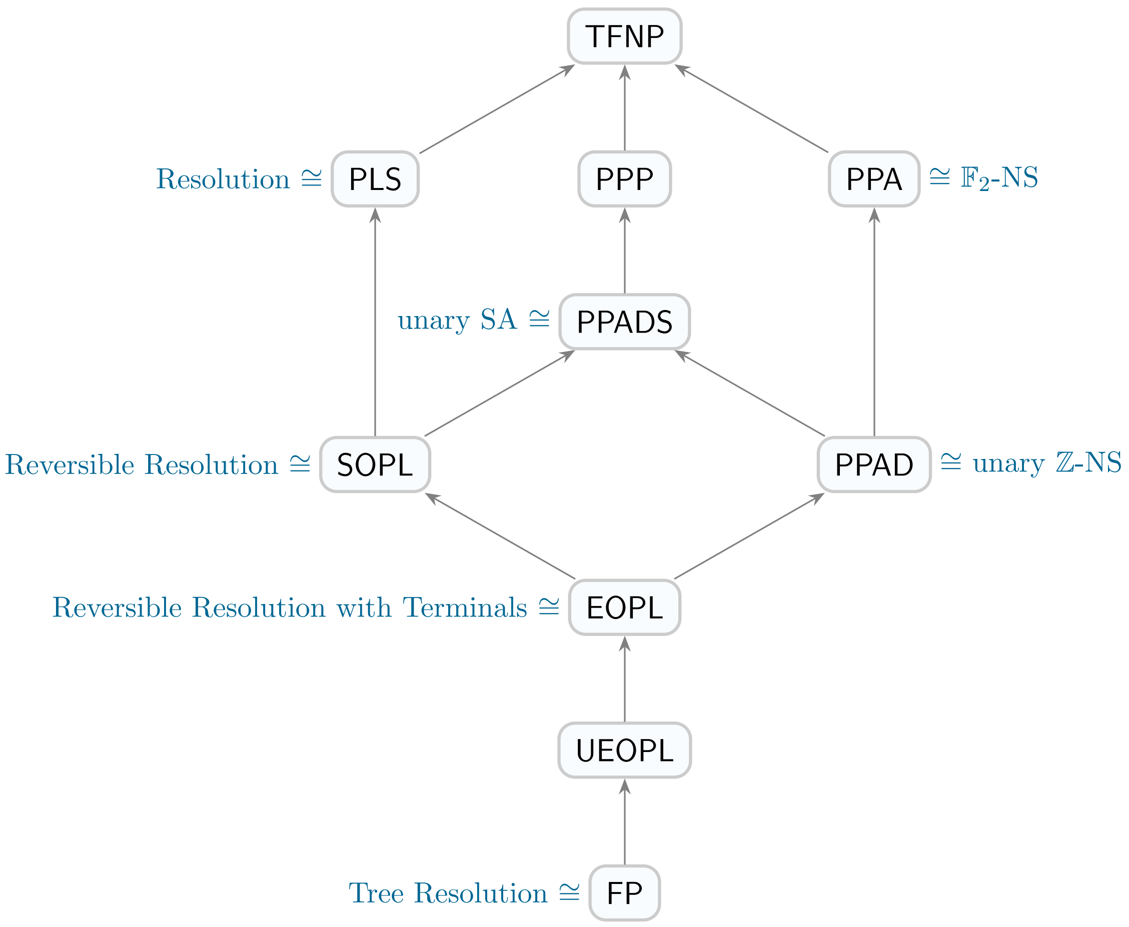

TFNP
Resolution ≅ PLS
PPP
PPA ≅ $\mathbb{F}_2$-NS
unary SA ≅ PPADS
Reversible Resolution ≅ SOPL
PPAD ≅ unary $\mathbb{Z}$-NS
Reversible Resolution with Terminals ≅ EOPL
UEOPL
Tree Resolution ≅ FP

Figure 1: Class inclusion diagram for TFNP. An arrow $A \to B$ means $A \subseteq B$ 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 [SA94], NS refers to the Nullstellensatz proof system [BIK+94], and "unary" refers to the fact that we measure size by the sum of all coefficients occurring in the proof.

- 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).

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 [KST07, Tha16]. 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 how 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[1] in the case of PLS, EOPL, and SOPL — having distinguished source node s with at least one 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[2].

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:

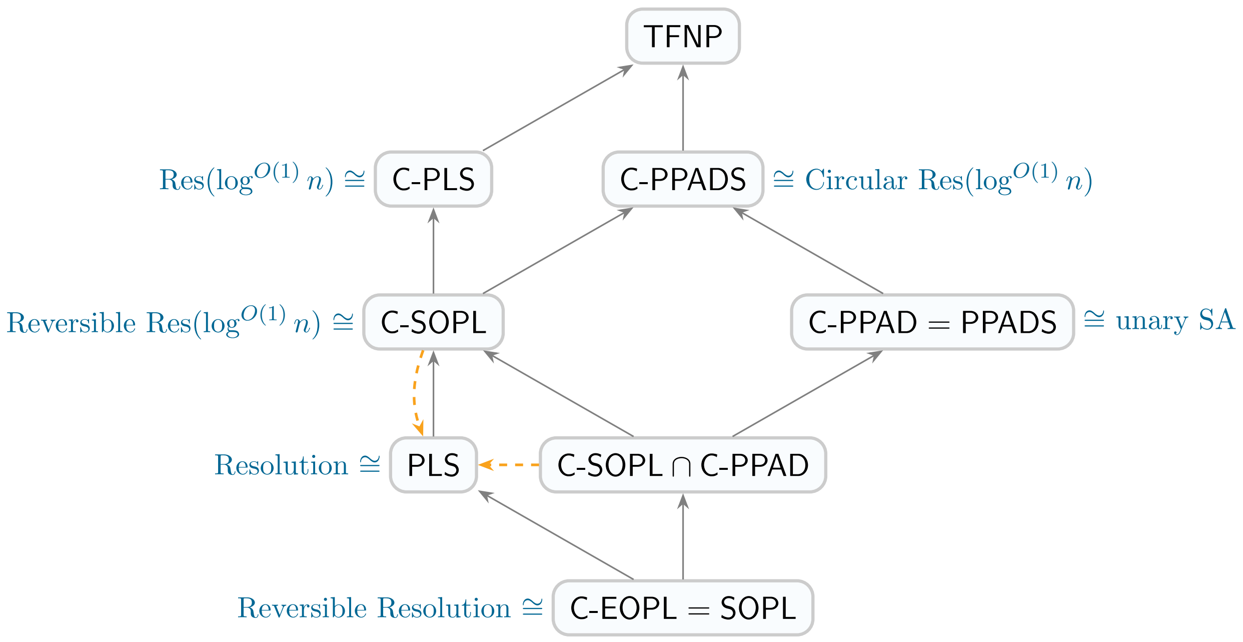
- Any proper source node *with a colour* is a solution,
- Any sink node *with no colour* (i.e. if $C_u = \emptyset$) is a solution, and
- A node u with an out-neighbour v is a solution if there is a colour $\lambda \in C_v$ such that $\lambda \notin C_u$.

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).

[1]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.

[2]For the interested reader, we note that this similarity was identified and formalized as a general GRID search problem in [GHJ+22a].



TFNP
$\text{Res}(\log^{O(1)} n) \cong$ C-PLS
C-PPADS $\cong$ Circular $\text{Res}(\log^{O(1)} n)$
Reversible $\text{Res}(\log^{O(1)} n) \cong$ 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.

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.

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:*

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

In the above theorem, $\text{Res}(\mathsf{polylog}(n))$ is the extension of Resolution to DNF formulas with $\mathsf{polylog}(n)$-width conjunctions on the bottom level (see e.g. [Kra01, SBI04, AB04, Gar20]). 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 [AL19] 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 $\mathsf{PLS}, \mathsf{PPADS}$, 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 $\mathsf{C\text{-}PPAD}$ and $\mathsf{C\text{-}EOPL}$. 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 [GHJ+22b].

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 $\mathsf{PPADS}^{dt} \cong \mathrm{uSA}$ [GHJ+22b], 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 [GHJ+22b], 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 Section 5). 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, 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 [AL19]. 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* [GHJ+22b]. 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) [DMR09] but hard for Res(k) [SBI04].
- 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 [BB10, ST11].

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. Finally, in Section 4 and Section 5 we provide the 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* [GHJ+22b]. The Reversible Resolution variant (and, in particular, the "reversible" rules presented below) were first introduced in the context of MaxSAT solving [BLM07, LHdG08, FMSV20].

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.

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

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).

$\wedge$-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 [BB22]. 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. [AL19]).

$$\frac{}{D} \quad \textbf{(DNF Creation)}$$

Definition 2.4. Let F be a CNF formula. A *Unary Circular* 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 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 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 uCircRes(k) *refutation* of F.

Both Res(k) and CircRes(k) can efficiently simulate RevRes(k), since 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 [GHJ+22b].

Definition 2.5. A *total (query) search problem* is a sequence of relations $\mathrm{R} = \{R_n \subseteq \{0,1\}^n \times O_n\}$, 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 $(x, o) \in \mathrm{R}$.

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

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

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 $\mathrm{S} = (S_m)$ is a sequence of search problems, then we denote by $\mathrm{S}^{dt}(R)$ the minimum of $S_m^{dt}(R)$ over all m. If $\mathrm{R} = (R_n)$ is also a sequence, then we denote by $\mathrm{S}^{dt}(\mathrm{R})$ the function $n \mapsto \mathrm{S}^{dt}(R_n)$.

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-SoPL 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 Sans-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 $\vec{x}_i := x_{i,1} \ldots x_{i,t_i} \in \{0,1\}^{t_i}$, 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 $C_1, C_2, \ldots, C_s = C$ 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 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 $C_1, C_2, \ldots, C_s = C$ 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 Res(k). It is enough to prove this next lemma for RevRes(d) since both Res(d) and uCircRes(d) simulate RevRes(d).

Lemma 3.7. *If T is a decision tree of depth d, then there is a size-2^{2d}* 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 $1 < i \le k$ to obtain $\bigvee_{1<i\le k} C_{p_i} \vee \ell \vee \overline{\ell}$. $\wedge$-introducing this with $D_{T_0} \vee \overline{\ell}$ on C_{p_1} and $\overline{\ell}$, we obtain:

$$\bigvee_{1<i\le k} C_{p_i} \vee C_{p1\cup\overline{\ell}} \vee \ell$$

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

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

Repeating this for the remaining paths p_j for $2 < j \le k$, 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 $O(d)$. $\square$

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 $\mathrm{Res}(O(d))$ 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 $\mathrm{RevRes}(d)$.

Lemma 3.9. *Suppose that x is computed by a depth-d decision tree and G is a DNF. Then there is a* $\mathrm{RevRes}(d)$ *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 $\mathcal{D} := G \vee \mathcal{D}^+(\llbracket x \neq 0 \rrbracket)$ 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 $\mathcal{D}^-(\llbracket x_i \neq j \rrbracket)$.

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 $C_{p_1}, C_{p_2}, \ldots, C_{p_s}$ we must pay $O(|G|^2 2^d)$ to derive the corresponding DNF to cut our preserved formula with, and this will repeat $s \leq 2^d$ times, for a total cost of $O(|G|^2 2^{2d})$. Finally, we must repeat this entire process $\leq 2^d$ 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 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 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 $\vec{y}_1, \ldots, \vec{y}_m$ 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 $A_i := \llbracket x_i \neq \ell_i \rrbracket$ 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 $2^{wd} \leq 2^{cd}$ 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 $\ell = 0, 1, \ldots, r_i - 1$ 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 $\{\mathcal{D}^+(C) \vee \mathcal{D}^+([\![x_i \neq \ell]\!]) \mid \ell \in [r_i - 1]_0\}$. 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 $C \vee [\![x \neq 0]\!], ..., C \vee [\![x \neq n-1]\!]$, 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. $\square$

4 Containments and New Characterizations

4.1 C-PLS and Res(polylog(n))

We will show the following characterization:

Theorem 4.1. $\mathsf{C\text{-}PLS}^{dt} \cong \mathrm{Res}(\mathsf{polylog}(n))$

We prove this in two parts.

Theorem 4.2. *Let m be an integer, and let F be a width-w CNF. If there exists a* $\mathrm{Res}(d)$ *refutation of F of size m, then there is a depth-$O(w+d)$* $\mathrm{C\text{-}SoD}_{O(mw)}$*-formulation of $S(F)$.*

Theorem 4.3. *Let F be a CNF formula. If there exists a depth-d* $\mathrm{C\text{-}SoD}_n$*-formulation of $S(F)$, then there is a* $\mathrm{Res}(d)$ *refutation of F of size $n^{O(1)} \cdot 2^{O(d)}$*

We begin with the first direction:

Proof of Theorem 4.2. Let m and F be as above, and let P be a $\mathrm{Res}(d)$-refutation of F of size m. We construct a $\mathrm{C\text{-}SoD}_{O(mw)}$-formulation as follows. Regardless of input, we introduce a colour λ_C for each conjunction C appearing in some line of P. Likewise, for each DNF D appearing as a line in the proof, we have a corresponding node u_D. We implicitly assume that these nodes are embedded in a grid, the size of which we will determine later. If we select a node as the successor for some u_D which cannot be placed on the immediate next layer, we can easily remedy this by duplicating u_D to the layer below and pointing to that instead, repeating until we reach the required layer. If these nodes do not fill the grid, we can *deactivate* each remaining node v by introducing a single additional colour λ_0, then setting $c_{v,\lambda_0} = 1$, $s_v = 0$ and $e_v = \lambda_0$. This will ensure these extraneous nodes do not give rise to a solution.

Now suppose we have an assignment $\vec{a}$ to the variables of F. For each node u_D in the C-SoD-formulation, we must define functions s_{u_D}, e_{u_D}, and $c_{u_D,\lambda}$, for all colours λ, on input $\vec{a}$ which output the corresponding parameters with sufficiently low decision tree depth.

We begin with the colour indicators. For each line D and colour λ_C, we set $c_{u_d,\lambda_C}(\vec{a}) = C(\vec{a})$ if the conjunction C appears in D and 0 otherwise. Here $C(\vec{a})$ denotes the function which outputs 1 exactly when C is satisfied under $\vec{a}$, and 0 otherwise.

We now select a successor for each node u_D, which we will do to ensure our colouring does not create a violation except at initial clauses of F. We have multiple cases according to the rule used to derive D:

- **Initial Clause.** If D is an initial clause of F, then u_D has no successor (we set $s_{u_D} = 0$ and place it on the final layer).
- **Axiom Introduction.** If D was derived as an axiom, then $s_{u_D}(\vec{a}) = 0$ as exactly one of ℓ and $\overline{\ell}$ are satisfied.
- **Reverse-Cut.** If D was derived by Reverse-Cut on the premise D_0, we set $s_{u_D}(\vec{a}) = u_{D_0}$.
- **$\wedge$-Introduction.** If D was derived by $\wedge$-introduction on $D_0 \vee A$ and $D_0 \vee B$, then either $D = D_0 \vee (A \wedge B)$ or $D = D_0 \vee A \vee B$. In the first case, $s_{u_D}(\vec{a}) = u_{D_0 \vee A}$ if A is falsified under $\vec{a}$, $u_{D_0 \vee B}$ if B is falsified, and 0 if $A \wedge B$ is satisfied under $\vec{a}$. In the second case, we set $s_{u_D} = u_{D_0 \vee A}$.
- **Cut.** Finally, if D was derived by cutting on some conjunction $C = \ell_1 \wedge \cdots \wedge \ell_k$ and its negation $\overline{C} = \overline{\ell}_1 \vee \cdots \vee \overline{\ell}_k$, then $s_{u_D}(\vec{a}) = u_{D \vee C}$ if C is falsified by $\vec{a}$ and $u_{D \vee \overline{C}}$ otherwise.

Finally, we define the decision trees outputting the pointer e_{u_D} for each node. If D is an initial clause of F, $e_{u_D}(\vec{a}) = \ell$, where ℓ is the first satisfied literal of D if one exists and an arbitrary (unsatisfied) literal of D otherwise, to intentionally create a violation whenever D is falsified. If D is not an initial clause of F, then as before we will define e_{u_D} in order to avoid a contradiction. By the construction above either u_D is not a leaf, in which case e_{u_D} has no impact, or u_D is a leaf (which can be checked by running the corresponding successor decision tree) and by construction D contains a conjunction C which is satisfied under $\vec{a}$. Thus we set $e_{u_D}(\vec{a}) = \lambda_C$ if the latter case holds, and choose an arbitrary colour otherwise.

Notice that since all conjunctions appearing as disjuncts in P must either appear as literals in F or be derived by rule applications, the number of colours is $O(mw)$. Thus, since we only require a grid of size $m \times m$, our formulation has size $O(mw)$ overall. Further, since each function in our construction can be computed by either querying each literal in at most two conjunctions of the proof or querying each literal in an initial clause of F, the depth of this formulation is $O(w + d)$.

It remains to show that all solutions to this formulation correspond to solutions of $S(F)$. Let u_D be a node in our construction above. We again have multiple cases depending on the rule used to derive D:

- If D is an **initial clause** of F, u_D was placed on the bottom layer and $s_{u_D} = 0$. If D is satisfied, then e_{u_D} points to the colour of a satisfied literal ℓ in D, so u_D does not form part of a solution. Otherwise, e_{u_D} points to the colour of an unsatisfied literal, so u_D, ℓ is a solution as desired. In fact, we will see that this case is the only one which can produce a solution.

- If D was derived as an **axiom**, then $s_{u_D} = 0$ and e_{u_D} points to the colour of a satisfied literal ℓ, so u_D does not give rise to a solution.
- If D was derived by **Reverse-Cut** D_0 by A, then since $s_{u_D} = u_{D_0}$ and D contains D_0, u_D does not give rise to a solution.
- If D was derived by **$\wedge$-introduction** on $D_0 \vee A$ and $D_0 \vee B$, then either $D = D_0 \vee (A \wedge B)$ or $D = D_0 \vee A \vee B$. In the latter case, $s_{u_D} = u_{D_0 \vee A}$, which contains no new colours. In the first case $s_{u_D} = 0$ if $A \wedge B$ is satisfied, in which case $e_{u_D} = \lambda_{A \wedge B}$. Otherwise, s_{u_D} points to whichever of $u_{D_0 \vee A}$ and $u_{D_0 \vee B}$ contains no additional colours. In all cases, no solution is created.
- If D was derived by **cutting** on a conjunction C and its negation $\overline{C}$, then exactly one of C and $\overline{C}$ is falisified. So since $s_{u_D} = u_{D \vee C}$ if C is falsified and $u_{D \vee \overline{C}}$ otherwise, the chosen successor contains no additional colours, and u_D does not give rise to a solution.

Thus, the only solutions of this formulation consist of nodes corresponding to falsified clauses of F, as desired. □

We now move on to the reverse direction. To prove this direction, we recall the formulation of the principle underlying C-SoD as an unsatisfiable multivalued CNF formula (cf. Section 3.2) and show that this can be refuted by Resolution with a size-$n^{O(1)}$ block-respecting proof.

- $\forall i, j, k, \lambda \in [n]_0$, $[\![s_{i,j} \neq k]\!] \vee [\![c_{i+1,k,\lambda} \neq 1]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!]$
- $\forall \lambda \in [n]_0$, $[\![c_{1,1,\lambda} \neq 1]\!]$
- $\forall i, j, \lambda \in [n]_0$, $[\![s_{i,j} \neq 0]\!] \vee [\![e_{i,j} \neq \lambda]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!]$
- $\forall j, \lambda \in [n]_0$, $[\![e_{n,j} \neq \lambda]\!] \vee [\![c_{n,j,\lambda} \neq 0]\!]$

Lemma 4.4. C-SoD$_n$ *has a width-$O(n)$, size-$O(n^5)$ Blockwise Calculus refutation.*

Proof. We will prove that for all $i, j \in [n]$, we can derive the clause:

$$I_{i,j} := \bigvee_{\lambda \in [n]} [\![c_{i,j,\lambda} \neq 0]\!]$$

which encodes that some colour is present at (i, j). Once this is derived for the node $(1, 1)$, we can cut $I_{i,j}$ with the clause $[\![c_{1,1,\lambda}]\!]$ for each $\lambda \in [n]$ to derive $\perp$ and conclude the refutation.

We derive $I_{i,j}$ as follows. For each bottom node (n, j), resolve all initial clauses of the form $[\![e_j \neq \lambda]\!] \vee [\![c_{n,j,\lambda} \neq 0]\!]$ together on e_j to obtain $\bigvee_{\lambda \in [n]} [\![c_{n,j,\lambda} \neq 0]\!]$.

For all other nodes (i, j):

- For each $k \in [n]$, begin with the clause $[\![s_{i,j} \neq k]\!] \vee [\![c_{i+1,k,1} \neq 1]\!] \vee [\![c_{i,j,1} \neq 0]\!]$ and cut it with $I_{i+1,k}$ on $c_{i+1,k,1}$ to obtain:

$$[\![s_{i,j} \neq k]\!] \vee \bigvee_{1 < \lambda \leq n} [\![c_{i+1,k,\lambda} \neq 0]\!] \vee [\![c_{i,j,1} \neq 0]\!]$$

We then cut this result with the clause $\llbracket s_{i,j} \neq k \rrbracket \vee \llbracket c_{i+1,k,2} \neq 1 \rrbracket \vee \llbracket c_{i,j,2} \neq 0 \rrbracket$ on $c_{i+1,k,2}$ to obtain:

$$\llbracket s_{i,j} \neq k \rrbracket \vee \bigvee_{2<\lambda\leq n} \llbracket c_{i+1,k,\lambda} \neq 0 \rrbracket \vee \bigvee_{1\leq\lambda\leq 2} \llbracket c_{i,j,\lambda} \neq 0 \rrbracket$$

Repeating this for each remaining clause $\llbracket s_{i,j} \neq k \rrbracket \vee \llbracket c_{i+1,k,\lambda} \neq 1 \rrbracket \vee \llbracket c_{i,j,\lambda} \neq 0 \rrbracket$ for $\lambda \in [n]$, we obtain the clause:

$$\llbracket s_{i,j} \neq k \rrbracket \vee \bigvee_{\lambda\in[n]} \llbracket c_{i,j,\lambda} \neq 0 \rrbracket$$

- Successively cut all clauses of the form $\llbracket s_{i,j} \neq 0 \rrbracket \vee \llbracket e_{i,j} \neq \lambda \rrbracket \vee \llbracket c_{i,j,\lambda} \neq 0 \rrbracket$ for $\lambda \in [n]$ together on $e_{i,j}$ to obtain:

$$\llbracket s_{i,j} \neq 0 \rrbracket \vee \bigvee_{\lambda\in[n]} \llbracket c_{i,j,\lambda} \neq 0 \rrbracket$$

- Finally, resolve all the resulting clauses of the form $\llbracket s_{i,j} \neq k \rrbracket \vee \bigvee_{\lambda\in[n]} \llbracket c_{i,j,\lambda} \neq 0 \rrbracket$ for $k \in [n] \cup \{0\}$ together on $\llbracket s_{i,j} \neq k \rrbracket$ to eliminate the successor indices and obtain $\bigvee_{\lambda\in[n]} \llbracket c_{i,j,\lambda} \neq 0 \rrbracket = I_{i,j}$ as desired.

Since for each (i, j) there are n^2 clauses of the form $\llbracket s_{i,j} \neq k \rrbracket \vee \llbracket c_{i+1,k,\lambda} \neq 1 \rrbracket \vee \llbracket c_{i,j,\lambda} \neq 0 \rrbracket$ and n of the form $\llbracket s_{i,j} \neq 0 \rrbracket \vee \llbracket e_{i,j} \neq \lambda \rrbracket \vee \llbracket c_{i,j,\lambda} \neq 0 \rrbracket$, each recursive step contributes $O(n^3)$ to the size of the proof. Thus, since we have n^2 vertices, the proof has size $O(n^5)$ overall. Furthermore, the proof has width $O(n)$ as desired.

□

Theorem 4.3 now follows immediately by applying Theorem 3.6.

4.2 C-SOPL and Reversible Res(polylog(n))

We will show the characterization:

Theorem 4.5. $\mathsf{C\text{-}SOPL}^{dt} \cong \mathrm{RevRes}(\mathsf{polylog}(n))$

We again split this into two directions:

Theorem 4.6. *Let F be a width-w CNF. If there exists a* $\mathrm{RevRes}(d)$ *refutation of F of size m, then there is a depth-$O(w + d)$* $\mathrm{C\text{-}SoPL}_{O(mw)}$*-formulation of $S(F)$.*

Theorem 4.7. *Let F be a CNF formula. If there is a depth-d* $\mathrm{C\text{-}SoPL}_n$*-formulation of $S(F)$, then there is a size-$n^3 2^{O(d)}$* $\mathrm{RevRes}(d)$ *refutation of F.*

Proof. Let F be as above, and let P be a $\mathrm{RevRes}(d)$ refutation of F of size m. The construction of our formulation will follow the proof of Theorem 4.2 closely.

We again include a colour λ_C for each bottom-level conjunction C appearing in the proof, and a node u_D for each DNF appearing as a line in the proof. We implicitly assume these nodes are embedded in a grid, padding with intermediate nodes as required. We also insert an additional column labelled "$NULL$" to the grid to allow us to treat nodes as having no predecessor or successor as we require. All nodes u in this column point to $NULL$ as both their successor and predecessor, and contain a single distinct colour λ_{NULL} which e_u points to. We can now *deactivate* an unused node u by setting $c_{u,\lambda_{NULL}} = 1$, then setting $s_u = p_u = NULL$ and $e_u = \lambda_{NULL}$.

Now consider an assignment $\vec{a}$ to the variables of F. For each node u of our formulation, we must define the behaviour of functions s_u, p_u, e_u, and for each colour λ, $c_{u,\lambda}$, on input $\vec{a}$. The colour indicators are defined as in the proof of Theorem 4.2 – for each node u_D and conjunction C in the proof, $c_{u_D,\lambda_C}(\vec{a}) = C(\vec{a})$ if C appears in D, and 0 otherwise.

We proceed to define the successor and predecessor pointers. We begin with the successor pointers, which for a given node u_D are defined according to the rule used to derive D:

- **Initial Clause** If D is an initial clause of F, we place u_D on the final layer.
- **Axiom Introduction** If D was derived as an axiom, then we make u_D a leaf by setting $s_{u_{D,i}} = NULL$
- **$\wedge$-Introduction** If D was derived by $\wedge$-introducing on $D_0 \vee A$ and $D_0 \vee B$, then either $D = D \vee (A \wedge B)$ or $D_0 \vee A \vee B$. In the first case, $s_{u_D}(\vec{a}) = u_{D_0 \vee A}$ if A is falsified under $\vec{a}$, $u_{D_0 \vee B}$ if B is falsified, and $NULL$ if both are satisfied. In the second case, we set $s_{u_D} = NULL$ if A or B are satisfied under $\vec{a}$ and $u_{D_0 \vee B}$ otherwise.
- **Cut** If D was derived by cutting on some conjunction $C = \ell_1 \wedge \cdots \wedge \ell_k$ and its negation $\overline{C} = \overline{\ell}_1 \vee \cdots \vee \overline{\ell}_k$, then we point s_{u_D} to $u_{D \vee C}$ if C is falsified under $\vec{a}$, and $u_{D \vee \overline{C}}$ if $\overline{C}$ is falsified (exactly one of these must be true).
- **Reverse-Cut** If D was derived from D_0 by Reverse-Cut, then $s_{u_D}(\vec{a}) = u_{D_0}$.

The predecessors are defined similarly, this time according to the rule which **consumes** D:

- **$\wedge$-Introduction** If D is $\wedge$-introduced on to produce formulas $D_0 \vee (A \wedge B)$ and $D_0 \vee A \vee B$, then at most one of these formulas' nodes points to u_D as its successor. We can run the corresponding successor trees to determine which, and point to this as the predecessor. If neither do, set $p_{u_D} = NULL$.
- **Cut** If D is cut on to produce D_0, then $p_{u_D}(\vec{a}) = u_{D_0}$.
- **Reverse-Cut** If D is reverse-cut on on some conjunction C to produce formulas $D \vee C$ and $D \vee \overline{C}$, then we point s_{u_D} to $u_{D \vee C}$ if C is falsified under $\vec{a}$, and $u_{D \vee \overline{C}}$ if $\overline{C}$ is falsified.

It remains only to define the colour pointer e_{u_D}, which we do exactly as before, pointing to the first satisfied literal, if one exists, of D if D is derived as an axiom or is an initial clause of F. In any other situation where a node is made into a sink, there is again some conjunction C, of width at most k, which must be satisfied, so if this occurs we point to the colour corresponding to C.

Since we include colours for each conjunction in the proof, each of which is either a literal of some initial clause of F or was derived at some line of the proof, the number of colours is $O(mw)$ overall. Thus, since we only need a grid of size at most m, this is a C-SoPL$_n$-formulation. Furthermore, all functions can be computed by either querying at most w literals or a constant number of width-k conjunctions, so the depth is $O(w + d)$ as desired.

It remains to see that this formulation is correct. Again suppose we have an assignment $\vec{a}$ to the variables of F and consider a node u_D from our construction above. We will show that solutions can only arise at a node u_D if D is an unsatisfied initial clause of F. We again have multiple cases depending on how D was derived:

- **Initial Clause** If D is an initial clause of F, then $s_{u_D} = NULL$. If D is satisfied under $\vec{a}$, e_{u_D} points to some λ_ℓ, where ℓ is a satisfied literal of D and thus λ_ℓ is present at u_D. Thus, a solution can only arise at u_D if D is unsatisfied.

- **Axiom Introduction** If D was derived as an axiom, it is a sink in our construction, but one of its literals is satisfied by definition and the corresponding colour will be pointed to by e_{u_D}. Thus no solution can arise at u_D.

- **$\wedge$-Introduction** If D was derived by $\wedge$-introducing on $D_0 \vee A$ and $D_0 \vee B$, then either $D = D_0 \vee (A \wedge B)$ or $D_0 \vee A \vee B$. In the latter case, u_D contains all colours of s_{u_D}, whose predecessor pointer points to u_D. In the first case, u_D is a sink if both A and B are satisfied, in which case e_{u_D} points to $\lambda_{A \wedge B}$, which is present. Otherwise, s_{u_D} points to whichever of $u_{D_0 \vee A}$ and $u_{D_0 \vee B}$ introduces no additional colours, as our construction always selects the one containing the unsatisfied conjunction. Further, s_{u_D} points to u_D as its predecessor, so it is not a sink. Thus, in all cases, no solution arises.

- **Cut** If D was derived by cutting on some conjunction C, then exactly one of C and $\overline{C}$ is falsified, and as with $\wedge$-introduction s_{u_D} points to whichever of $u_{D \vee C}$ and $u_{D \vee \overline{C}}$ introduces no additional colours. Each of these nodes points back to u_D as its predecessor, so no solutions arise.

- **Reverse-Cut** If D is of the form $D_0 \vee D'$ and was derived from D_0 by reverse-cut, then u_D points to u_{D_0} as its successor, which contains no additional colours as D_0 is a sub-formula of D. u_{D_0} points back to u_D as its predecessor unless D' is falsified under $\vec{a}$, in which case u_D becomes a sink but e_{u_D} points to D', which is satisfied, creating no solutions.

Thus the only solutions to this formulation which can arise occur at the bottom layer and correspond to unsatisfied initial clauses of F, as desired. □

We proceed to the reverse direction, and again recall the formulation of the principle underlying C-SoPL as a multivalued unsatisfiable CNF formula (cf. Section 3.2).

1. $\forall i \in [n-2]_0, j, k, \lambda \in [n-1]_0$, $[\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee [\![c_{i+1,k,\lambda} \neq 1]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!]$
2. $\forall \lambda \in [n-1]_0$, $[\![c_{0,0,\lambda} \neq 1]\!]$
3. $\forall j, \lambda \in [n-1]_0$, $[\![e_{n-1,j} \neq \lambda]\!] \vee [\![c_{n-1,j,\lambda} \neq 0]\!]$
4. $\forall i \in [n-2]_0, j \neq \ell, j, \lambda \in [n-1]_0$, $[\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq \ell]\!] \vee [\![e_{i,j} \neq \lambda]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!]$

As in the case of $\mathsf{C\text{-}PLS}^{dt}$ and Res(k), by Theorem 3.6 it suffices to instead show the following:

Lemma 4.8. *There is a reversible blockwise calculus refutation of* C-SoPL$_n$ *with size $O(n^5)$ and width $O(n)$.*

Proof. We will prove that for all nodes (i, j) in the formulation, we can derive the formula

$$I_{i,j} := \bigvee_{\lambda \in [n]} [\![c_{i,j,\lambda} \neq 0]\!]$$

encoding that some colour is present at (i, j). Once this is derived for the node $(0, 0)$, we may cut it with each clause (2) to derive $\perp$ and reach a contradiction.

We proceed to derive each $I_{i,j}$. For $i = n-1$ and $j \in [n-1]_0$, we simply resolve all corresponding clauses (3) together on $e_{i,j}$ to obtain $I_{i,j}$. For $i < n-1$, we proceed as follows:

- For each $k \in [n-1]_0$, take $I_{i+1,k}$ and split along all values of $p_{i+1,k}$ to obtain:

  $$[\![p_{i+1,k} \neq j]\!] \vee \bigvee_{\lambda \in [n-1]_0} [\![c_{i+1,k,\lambda} \neq 0]\!]$$

  for each $j \in [n-1]_0$. Weakening each of these by $[\![s_{i,j} \neq k]\!]$ yields:

  $$[\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee \bigvee_{\lambda \in [n-1]_0} [\![c_{i+1,k,\lambda} \neq 0]\!]$$

  for each $j \in [n-1]_0$.

- Take all clauses (1) of the form $[\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee [\![c_{i+1,k,\lambda} \neq 1]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!]$ and cut in order with the corresponding clause from the previous step to derive:

  $$[\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee \bigvee_{\lambda \in [n-1]_0} [\![c_{i,j,\lambda} \neq 0]\!] = [\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee I_{i,j}$$

  for each $j \in [n-1]_0$

To complete the proof, we will exploit reversibility and show that this set of formulas can be derived in reverse from $\bigwedge_{j \in [n-1]_0} I_{i,j}$ without producing any additional formulas.

- We first split $I_{i,j}$ along all values of $s_{i,j}$ to obtain:

  $$[\![s_{i,j} \neq k]\!] \vee I_{i,j}$$

  for all $k \in [n-1]_0$.

- For each $k \in [n-1]_0$, take the corresponding clause from the previous step and split on $p_{i+1,k}$ to obtain:

  $$[\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq \ell]\!] \vee I_{i,j}$$

  for each $\ell \in [n-1]_0$. If $\ell = j$, we are done. Otherwise, we can split this clause on $e_{i,j}$ to obtain weakenings of clauses (4) for each $\lambda \in [n-1]_0$:

  $$[\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq \ell]\!] \vee [\![e_{i,j} \neq \lambda]\!] \vee I_{i,j}$$

Each step above contributes at most $O(n^3)$ to the size of the proof, so since we have n^2 nodes, this proof has size $O(n^5)$ overall. Furthermore, the width of this proof is $O(n)$. □

Theorem 4.7 now follows.

4.3 C-PPADS and Circular Res(polylog(n))

We wish now to characterize C-PPADS^{dt} as follows:

Theorem 4.9. C-PPADS^{dt} $\cong$ uCircRes(polylog(n))

As usual, we prove this in two directions:

Theorem 4.10. *Let F be a width-w CNF. If there exists a* uCircRes(d) *refutation of F of size m, then there is a depth-$O(w+d)$* C-SoL$_{O(mw)}$*-formulation of $S(F)$.*

Theorem 4.11. *Let F be a CNF formula. If there is a depth-d C-SoL_n-formulation of $S(F)$, then there is a size-$n^{O(1)} \cdot 2^{O(d)}$ $\text{uCircRes}(d)$-refutation of F.*

Proof. Let F be a width-w CNF, and let P be a uCircRes(d) refutation of F of size m. We construct a $\text{C-SoL}_{O(mw)}$-formulation as follows. As before, we include a colour λ_C for each bottom-level conjunction C in P, and a node u_D for each DNF appearing as a line.

Now consider an assignment $\vec{a}$ to the variables of F. Again, for each node u we must define functions s_u, p_u, e_u, and for each colour λ, $c_{u,\lambda}$. The definitions of these functions will be largely taken from the proof of Theorem 4.6 – for each line D and conjunction C in P, $c_{u_D,\lambda_C} = C(\vec{a})$ if C appears in D and 0 otherwise.

We first define the successor and predecessor pointers for each node u_D for the usual rules of Res(k), as these behave as usual. We will later show how to handle the new rules of uCircRes(k). The successor pointers for the usual rules are as follows:

- **Initial Clause.** If D is an initial clause of F, we set $s_{u_D} = u_D$.
- **Axioms Introduction.** If D is introduced as an axiom, set $s_{u_D} = u_D$.
- **$\wedge$-Introduction.** If D was derived by $\wedge$-introducing on $D_0 \vee A$ and $D_0 \vee B$, then either $D = D_0 \vee (A \wedge B)$ or $D_0 \vee A \vee B$. In the first case, $s_{u_D}(\vec{a}) = u_{D_0 \vee A}$ if A is falsified under $\vec{a}$, $u_{D_0 \vee B}$ if B is falsified, and u_D otherwise. In the second case, $s_{u_D}(\vec{a}) = u_D$ if either A or B are satisfied under $\vec{a}$, and $u_{D_0 veeB}$ otherwise.
- **Cut.** If D was derived by cutting on some conjunction $C = \ell_1 \wedge \cdots \wedge \ell_k$ and its negation $\overline{C} = \ell_1 \vee \cdots \vee \ell_k$, then we point s_{u_D} to $u_{D \vee C}$ if C is falsified under $\vec{a}$ and $u_{D \vee \overline{C}}$ otherwise.
- **Reverse Cut.** If D was derived by reverse cutting some premise D_0, then point s_{u_D} to u_{D_0}.

And we likewise define the predecessor pointers as follows:

- **$\wedge$-Introduction.** If D was consumed by $\wedge$-Introducing to produce some formulas $D_0 \vee (A \wedge B)$ and $D_0 \vee A \vee B$, then at most one of $u_{D_0 \vee (A \wedge B)}$ and $u_{D_0 \vee A \vee B}$ point to u_D as their successor. If either do, then point p_{u_D} back to it. Otherwise, set $p_{u_D} = u_D$.
- **Cut.** If D was consumed by cutting to produce some formula D_0, then set $p_{u_D} = u_{D_0}$.
- **Reverse-Cut.** If D was reverse-cut on some conjunction C to produce $D \vee C$ and $D \vee \overline{C}$, then point s_{u_D} to $u_{D \vee C}$ if C is falsified under $\vec{a}$ and $\overline{C}$ otherwise.

These will be the only rules which may create sink nodes, so we can already define the colour pointers e_{u_D} for each node D. We do this as usual: If D is an axiom or initial clause of F, then set e_{u_D} to be the first satisfied literal of D. Otherwise, if u_D is a sink in the formulation under input $\vec{a}$, then there is a conjunction C of width $\leq k$ which must be satisfied, so set $e_{u_D} = \lambda_C$.

Since we have changed nothing major so far, the same reasoning as for Theorem 4.6 holds to see that the only solutions arising from these rules under input $\vec{a}$ occur at clauses of F falsified by $\vec{a}$. The same arguments also hold to verify the size and depth of this formulation so far.

We finally extend this by defining pointers for formulas involved in DNF-Introduction. By definition of uCircRes(k), we may match each formula D derived by DNF-Introduction to a unique D' which is never consumed in a rule application. We then simply set $s_{u_D} = u_{D'}$, and $p_{u_{D'}} = u_D$. Since these nodes contain identical colour sets, this creates no additional solutions, and since the functions are constant this only requires decision tree depth 1, completing the formulation. □

Recall now the reverse direction:

Theorem 4.11. *Let F be a CNF formula. If there is a depth-d C-SoL$_n$-formulation of $S(F)$, then there is a size-$n^{O(1)} \cdot 2^{O(d)}$ uCircRes(d)-refutation of F.*

As usual, we will use the formulation of the principle underlying C-SoL as a CNF formula. Recall that C-SoL$_n$ has the following clauses:

1. $[\![s_u \neq v]\!] \vee [\![p_v \neq u]\!] \vee [\![c_{v,\lambda} \neq 1]\!] \vee [\![c_{u,\lambda} \neq 0]\!] \quad \forall u, v, \lambda \in [n-1]_0$
2. $[\![c_{0,\lambda} \neq 1]\!] \quad \forall \lambda \in [n-1]_0$
3. $[\![s_u \neq 0]\!] \quad \forall u \in [n-1]_0$
4. $[\![s_u \neq v]\!] \vee [\![p_v \neq w]\!] \vee [\![e_u \neq \lambda]\!] \vee [\![c_{u,\lambda} \neq 0]\!] \quad \forall u, v, w \neq u, \lambda \in [n-1]_0$

As with the other characterizations, by Lemma 3.6 it suffices to instead show the following:

Lemma 4.12. *There is a circular blockwise calculus refutation of* C-SoL$_n$ *with size $O(n^5)$ and width $O(n)$.*

Proof. The proof will heavily mirror the refutation of C-SoPL$_n$ in RevRes. As usual, we will prove the following formula for each node u of the formulation, encoding that some colour is present at u:

$$I_u := \bigvee_{\lambda \in [n-1]_0} [\![c_{u,\lambda} \neq 0]\!]$$

In particular, we will show that an extra copy of I_0 is derived, after which we may cut it with each clause (2) as before to obtain $\perp$ and conclude the refutation.

To derive each I_u, we will use the following intermediate formula:

$$I_{u,v} := [\![s_u \neq v]\!] \vee [\![p_v \neq u]\!] \vee I_u$$

for each pair of nodes u, v. We will show that $I_{u,v}$ can be derived for each $u \in [n-1]$ and $v \in [n-1]_0$ both in the forward direction from the formulas I_u for each $u \in [n-1]$, and in the backward direction from the formulas I_u for all $u \in [n-1]_0$ (without producing any additional clauses). This will produce an extra copy of I_0, as desired.

We begin with the forward direction. We begin by DNF-Introducing the formula I_u for each $u \in [n-1]$, and fix some $v \in [n-1]_0$. If $v = 0$, then each $I_{u,v}$ is simply a weakening of the corresponding clause (3). Otherwise, we derive each $I_{u,v}$ as follows:

- Take I_v and split along the negated values of p_v to obtain:

  $$[\![p_v \neq u]\!] \vee I_v$$

  for each $u \neq v \in [n-1]_0$. Weakening each of these formulas by $[\![s_u \neq v]\!]$ gives the formula:

  $$[\![s_u \neq v]\!] \vee [\![p_v \neq u]\!] \vee I_v$$

  again for each $u \neq v \in [n-1]_0$.

- Take all clauses (1) of the form $[\![s_u \neq v]\!] \vee [\![p_v \neq u]\!] \vee [\![c_{v,\lambda} \neq 1]\!] \vee [\![c_{u,\lambda} \neq 0]\!]$ and cut in succession with the corresponding formula from the previous step to obtain:

  $$[\![s_u \neq v]\!] \vee [\![p_v \neq u]\!] \vee I_u = I_{u,v}$$

  for all $u \neq v \in [n-1]_0$.

Repeating this process for each $v \in [n-1]_0$ derives all $I_{u,v}$. We now proceed to the reverse direction. Suppose we have the formulas I_u for all $u \in [n-1]_0$, and fix some $u \in [n-1]_0$. We derive each $I_{u,v}$ as follows:

- Take the clause I_u and split along the negation of all values of s_u to obtain:

  $$[\![s_u \neq v]\!] \vee I_u$$

  for all $v \neq u \in [n-1]_0$.

- For each $v \neq u \in [n-1]_0$, take the corresponding clause from the previous step and split on p_v to obtain formulas:

  $$[\![s_u \neq v]\!] \vee [\![p_v \neq w]\!] \vee I_u$$

  for all $w \neq v \in [n-1]_0$. If $w = u$, this is $I_{u,v}$ as desired. Otherwise, we split this formula on all values of e_u to obtain:

  $$[\![s_u \neq v]\!] \vee [\![p_v \neq w]\!] \vee [\![e_u \neq \lambda]\!] \vee \bigvee_{\lambda \in [n-1]_0} [\![c_{u,\lambda} \neq 0]\!]$$

  for each $\lambda \in [n-1]_0$, which are weakenings of initial clauses (4).

As desired, both directions derive exactly one copy of each $I_{u,v}$, and I_0 is derived but never consumed, except at the final step.

This proof again has width $O(n)$, and as each step above contributes $O(n^3)$ to the size of the proof and is applied to $O(n^2)$ nodes or pairs of nodes, the proof has size $O(n^5)$. ☐

Theorem 4.11 now follows.

4.4 C-PPAD = PPADS and Unary Sherali-Adams

We now show that, unlike the previous three characterizations, the characterization for C-PPAD^{dt} is in fact by a depth-0 proof system.

Theorem 4.13. C-PPAD^{dt} $\cong$ degree-polylog(n) uSA

This is shown in two parts:

Theorem 4.14. *Let F be a CNF formula. If there is a degree-d, size-m uSA refutation of F, then there is a depth-$O(d)$ C-EoL$_{O(m)}$-formulation of $S(F)$.*

Theorem 4.15. *Let F be a CNF formula. If there exists a depth-d C-EoL$_n$-formulation of $S(F)$, then there is a size-$O(n^2) \cdot 2^{O(d)}$, degree-$O(d)$ uSA refutation of F.*

We begin with the first part, which follows from the following characterization:

Theorem 4.16. *[GHJ+22a] Let F be a CNF formula. If there is a degree-d, size-m uSA refutation of F, then there is a depth-$O(d)$ $\mathrm{SoL}_{O(m)}$-formulation of $S(F)$. Conversely, if there is a depth-d SoL_n-formulation of $S(F)$, then there is a degree-$O(d)$, size-$n \cdot 2^{O(d)}$ uSA refutation of F.*

Proof of Theorem 4.14. It suffices to observe that any instance of SoL_n is trivially equivalent to an instance of C-EoL_n where no nodes contain any colours. □

We may now proceed to show the more interesting direction.

Proof of Theorem 4.15. Let $F = C_1 \wedge \cdots \wedge C_m$ be a CNF formula and suppose we have a depth-d C-EoL_n-formulation of $S(F)$. Let $V = [n-1]_0 \times [n-1]_0$ be the vertex set of this formulation and let $v* \in V$ denote the distinguished source node. For each $u \in V$, let s_u be the function outputting its successor, p_u be the function outputting its predecessor, and for each colour $\lambda \in [n-1]_0$, let $c_{u,\lambda}$ denote the function indicating whether λ is present at u. For each of these parameters k, let T_k denote the depth-d decision tree computing it, let Q_k denote the degree-d polynomial form of T_k, and for any path p of a decision tree, let Q_p denote the polynomial form of the conjunction which represents it. Finally, for a given input string x, we denote by G_x the graph obtained by including each edge (u, v) if and only if $s_u(x) = v$ and $p_v(x) = u$.

To support the intuition of the proof, we will construct a *counting function* for each node which sums to -1 over all nodes, for which we will first define several simpler expressions. For a given input string x and node u, we define $N_u(x)$ to be the number of colours present at u on input x, and we define $\Delta_u(x)$ to be $N_{s_u(x)}(x) - N_u(x)$. Intuitively, Δ_u represents the change in the number of colours between u and its successor. Thus, this quantity can only be positive if the number of colours at u is less than that at its successor, corresponding to a solution. Furthermore, both quantities can be computed by $O(d)$-degree polynomials. When we compute N_u as a polynomial, we include the colour pointer e_u so that we can handle solutions at sink nodes, as follows:

$$N_u = \sum_{\lambda \in [n-1]_0} \left[\sum_{p \in P_\lambda(T_{e_u})} Q_p Q_{c_{u,\lambda}} + \sum_{p \in P_{\lambda'}(T_{e_u})\,:\,\lambda \neq \lambda'} Q_p Q_{c_{u,\lambda}} \right]$$

Since all decision trees in the formulation have depth at most d, clearly this has degree $O(d)$. We similarly define the polynomial computing Δ_u in order to facilitate handling solutions arising from colouring mismatches, as follows:

$$\Delta_u = \sum_{\substack{p \in P(T_{s_u}) \text{ outputting } v,\\ q \in P_u(T_{p_v})}} Q_p Q_q \sum_{\lambda \in [n-1]_0} Q_{c_{v,\lambda}}(1 - Q_{c_{u,\lambda}}) - Q_{c_{u,\lambda}}(1 - Q_{c_{v,\lambda}})$$

The first portion of this expression detects the successor, while the second detects colouring mismatches. In particular, all terms of this expression are non-positive except the first term inside the sum, which is positive exactly when a colouring violation occurs on the corresponding colour and is thus a weakening of a clause of F. Furthermore, this also has degree $O(d)$, as again all decision trees in the formulation have depth at most d.

We can finally use these two quantities to define our counting function f_u as follows for each node u. For all $u \neq v^*$:

$$f_u(x) := \begin{cases} 1 - \Delta_u(x) - N_u(x) & u \text{ is a proper source in } G_x \\ -\Delta_u(x) & u \text{ is not a source nor sink in } G_x \\ N_u(x) - 1 & u \text{ is a proper sink in } G_x \\ 0 & u \text{ is isolated in } G_x \end{cases}$$

and for $u = v^*$:

$$f_{v^*}(x) := \begin{cases} -\Delta_{v^*}(x) - N_{v^*}(x) & v^* \text{ is a proper source in } G_x \\ -\Delta_{v^*}(x) - 1 & v^* \text{ is not a source nor sink in } G_x \\ N_{v^*}(x) - 2 & v^* \text{ is a proper sink in } G_x \\ -1 & v^* \text{ is isolated in } G_x \end{cases}$$

It is easy to see that for any x and any path P in G_x not containing v^*, $\sum_{u \in P} f_u = 0$, as Δ_u telescopes between nodes to leave the difference $N_s - N_t$ in number of colours between the sink t and source s, which is then compensated for by those nodes' counting functions. Furthermore, since f_{v^*} is always 1 less than f_u for an identical non-distinguished node u, $\sum_{u \in P} f_u = -1$ for any path P containing v^*. Thus, $\sum_{u \in V} f_u = -1$, as desired. However, we need to find a static proof. To do this, we simply sum up each case of f_u for all nodes u, multiplied by the degree-$O(d)$ polynomial which determines when the case occurs. This results in a degree-$O(d)$ polynomial overall, and since all decision trees in the formulation contain at most $2^{O(d)}$ paths, it has size $O(n^2) \cdot 2^{O(d)}$.

It remains to show that it can be separated into the form $\sum_{i \in [m]} p_i \tilde{C}_i + \mathcal{J} = -1$ for some polynomials $p_1, ..., p_m$ and conical junta $\mathcal{J}$. Any non-negative terms can be treated as part of $\mathcal{J}$, so we only need to show that the portions of the proof which may be negative on some input x can be derived from the clauses of F. We consider the cases of f_u. If $u \neq v^*$:

- **Proper source.** If u is a proper source on input x, then $f_u(x)$ only contains negative terms if either u contains a colour or its successor v contains a colour which u does not. In either case, by construction of N_u and Δ_u, the negative terms are weakenings of a clause of C-EoL and thus of F.

- **Middle node.** If u is not a proper source nor sink, then $f_u(x)$ contains negative terms exactly when its successor v contains a colour which u does not, so all negative terms are again weakenings of a clause of F.

- **Proper sink.** If u is a proper sink, $f_u(x)$ contains a negative constant term we must show we can derive. If u is on the bottom layer, we do this by summing all constraints of the form $-Q_p(1 - Q_q) = -Q_p + Q_q Q_p$ for each $p \in P(T_{e_u})$ outputting some λ and $q \in P_1(T_{c_{u,\lambda}})$, which are each a weakening of some clause of F. This sums all paths of $P(T_{e_u})$ to 1 and yields $-1 + \sum_{\lambda \in [n-1]_0} \sum_{p \in P_\lambda(T_{e_u}), q \in P_1(T_{c_{u,\lambda}})} Q_q Q_p = -1 + \sum_{\lambda \in [n-1]_0} \sum_{p \in P_\lambda \lambda_{T_{e_u}}} Q_p Q_q$. The remaining terms of f_u are non-negative, so they can simply be added as part of $\mathcal{J}$. If u is not on the bottom layer, the derivation is analogous; we simply multiply everything by the polynomial determining that u is a proper sink.

- **Isolated node.** If u is isolated, all terms of $f_u(x)$ are non-negative by definition

For $u = v^*$, we only need to consider the case where v^* is a proper source, as all other cases inherently give rise to solutions and their terms are thus weakenings of clauses of F. But in this case, the same argument as for non-distinguished proper sources still holds and we are done. Thus, all terms which are not non-negative can be expressed as weakenings of clauses of F, and this is a valid SA refutation of F. □

A corollary of this characterization is the following surprising collapse:

Corollary 4.17. $\mathsf{C\text{-}PPAD}^{dt} = \mathsf{PPADS}^{dt}$

Proof. This follows from Theorems 4.15, 4.16, and 4.14. □

4.5 C-EOPL = SOPL and Reversible Resolution

In this section we prove the following collapse theorem.

Theorem 4.18. *$\mathsf{C\text{-}EOPL}^{dt} = \mathsf{SOPL}^{dt}$, and thus $S(F) \in \mathsf{C\text{-}EOPL}^{dt}$ if and only if there is a $\mathsf{polylog}(n)$-width Reversible Resolution refutation of F.*

To prove this theorem we will crucially rely on the intersection theorem for SOPL^{dt}.

Theorem 4.19. *[GHJ+22a]* $\mathsf{SOPL}^{dt} = \mathsf{PLS}^{dt} \cap \mathsf{PPADS}^{dt}$

We know that $\mathsf{C\text{-}EOPL}^{dt} \subseteq \mathsf{C\text{-}PPAD}^{dt} = \mathsf{PPADS}^{dt}$ by Corollary 4.17. It therefore suffices to show that $\mathsf{C\text{-}EOPL}^{dt} \subseteq \mathsf{PLS}^{dt}$, since $\mathsf{C\text{-}EOPL}^{dt} \supseteq \mathsf{SOPL}^{dt}$ trivially (to obtain SOPL^{dt} we just turn off all colours in a $\mathsf{C\text{-}EOPL}^{dt}$ instance). To do this, we will use the following characterization of PLS^{dt}:

Theorem 4.20. *[BKT14, Kam20] For any unsatisfiable CNF formula F on n variables, $S(F) \in \mathsf{PLS}^{dt}$ if and only if there is a $\mathsf{polylog}(n)$-width Resolution refutation of F.*

It thus suffices to refute C-EoPL$_n$ in low-width Resolution.

Theorem 4.21. *Let F be a CNF formula. If there exists a depth-d C-EoPL$_n$-formulation of $S(F)$, then there is a size-$n^{O(1)} \cdot 2^{O(d)}$, width-$O(d)$ Resolution refutation of F.*

Proof. We will use the CNF encoding of C-EoPL$_n$, with clauses as follows:

1. $[\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee [\![c_{i+1,k,\lambda} \neq 1]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!] \qquad \forall i \in [n-2]_0, j, k, \lambda \in [n-1]_0$
2. $[\![c_{0,0,\lambda} \neq 1]\!] \qquad \forall \lambda \in [n-1]_0$
3. $[\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee [\![c_{1,j,\lambda} \neq 1]\!] \qquad \forall j \neq 1, k, \lambda \in [n-1]_0$
4. $[\![p_{i,j} \neq k]\!] \vee [\![s_{i-1,k} \neq g]\!] \vee [\![s_{i,j} \neq h]\!] \vee [\![p_{i+1,h} \neq j]\!] \vee [\![c_{i,j,\lambda} \neq 1]\!] \qquad \forall i > 1, j, k, g \neq j, h, \lambda \in [n-1]_0$
5. $[\![e_{n,j} \neq \lambda]\!] \vee [\![c_{n,j,\lambda} \neq 0]\!] \qquad \forall j, \lambda \in [n-1]_0$
6. $[\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq \ell]\!] \vee [\![e_{i,j} \neq \lambda]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!] \qquad \forall i < n, j, k, \ell \neq k, \lambda \in [n-1]_0$

By Theorem 3.10, it suffices to instead show the following:

Claim. There is a blockwise calculus refutation of C-EoPL$_n$ with size $O(n^3)$ and width $O(1)$.

Proof of Claim. Our refutation will proceed in two stages. In the first, we will prove the following clause for each node (i, j) for $i < n$, each k, and each colour λ, encoding that either $(i+1, k)$ is not the successor of (i, j) or λ is not present at (i, j):

$$I^{\rightarrow}_{i,j,k,\lambda} := [\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee [\![c_{i,j,\lambda} \neq 1]\!]$$

Together, these formulas imply that (i, j) either has no successor or contains no colours. In the next stage, we will prove from this that for each k, $(i+1, k)$ is not the successor of (i, j) (and thus (i, j) has no successor), encoded by the following:

$$I^{\leftarrow}_{i,j,k} := [\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!]$$

We begin by deriving each $I^{\rightarrow}_{i,j,k,\lambda}$. For $i = 1$, these are simply clauses (3). For $i > 1$, we proceed as follows for each i, j, k, λ:

- For each $h \in [n-1]_0$, take the clause $[\![s_{i-1,h} \neq j]\!] \vee [\![p_{i,j} \neq h]\!] \vee [\![c_{i,j,\lambda} \neq 1]\!] \vee [\![c_{i-1,h,\lambda} \neq 0]\!]$ and cut with $I^{\rightarrow}_{i-1,h,j,\lambda}$ on $c_{i-1,h,\lambda}$ to obtain the clause:
$$[\![s_{i-1,h} \neq j]\!] \vee [\![p_{i,j} \neq h]\!] \vee [\![c_{i,j,\lambda} \neq 1]\!] =: D^{\rightarrow}_{i,j,h,\lambda}$$
- Again for each $h \in [n-1]_0$, successively cut all clauses (4) of the form $[\![p_{i,j} \neq h]\!] \vee [\![s_{i-1,h} \neq g]\!] \vee [\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee [\![c_{i,j,\lambda} \neq 1]\!]$ together with the corresponding formula $D^{\rightarrow}_{i,j,h,\lambda}$ on $s_{i-1,h}$ to obtain:
$$[\![p_{i,j} \neq h]\!] \vee I^{\rightarrow}_{i,j,k,\lambda}$$
We finally cut these formulas together on $p_{i,j}$ to obtain $I^{\rightarrow}_{i,j,k,\lambda}$ as desired.

We now proceed to prove each clause $I^{\leftarrow}_{i,j,k}$. For $i = n$ and each $j, k \in [n-1]_0$, we take all clauses (5) of the form $[\![e_{n,j} \neq \lambda]\!] \vee [\![c_{n,j,\lambda} \neq 0]\!]$ and cut in succession with $I^{\rightarrow}_{n,j,k,\lambda}$ to obtain:
$$[\![s_{n,j} \neq k]\!] \vee [\![p_{n+1,k} \neq j]\!] \vee [\![e_{n,j} \neq \lambda]\!] = I \leftarrow_{i,j,k} \vee [\![e_{n,j} \neq \lambda]\!]$$
for all $\lambda \in [n-1]_0$. Cutting these formulas together on $e_{n,j}$ yields $I^{\leftarrow}_{i,j,k}$. Now for each remaining $i \in [n-2]_0, j, k \in [n-1]_0$, we proceed as follows:

- For each $\lambda \in [n-1]_0$, take the clause $[\![s_{i,j} \neq k]\!] \vee [\![p_{i+1,k} \neq j]\!] \vee [\![c_{i+1,k,\lambda} \neq 1]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!] = I^{\leftarrow}_{i,j,k} \vee [\![c_{i+1,k,\lambda} \neq 1]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!]$ and cut with $I^{\rightarrow}_{i,j,k,\lambda}$ on $c_{i,j,\lambda}$ to obtain:
$$I^{\leftarrow}_{i,j,k} \vee [\![c_{i+1,k,\lambda} \neq 1]\!] =: D^{\leftarrow}_{i,j,k,\lambda}$$
- Again for each $\lambda \in [n-1]_0$, take each clause (6) of the form $[\![s_{i+1,k} \neq h]\!] \vee [\![p_{i+2,h} \neq \ell]\!] \vee [\![e_{i+1,k} \neq \lambda]\!] \vee [\![c_{i+1,k,\lambda} \neq 0]\!]$ and cut them together with the corresponding formula $I^{\leftarrow}_{i+1,k,h}$ on $p_{i+2,h}$ to obtain:
$$[\![s_{i+1,k} \neq h]\!] \vee [\![e_{i+1,k} \neq \lambda]\!] \vee [\![c_{i+1,k,\lambda} \neq 0]\!]$$
for each $h, \lambda \in [n-1]_0$. Cutting these formulas together for each λ on $s_{i+1,k}$ yields $[\![e_{i+1,k} \neq \lambda]\!] \vee [\![c_{i,j,\lambda} \neq 0]\!]$
- Finally, take the clauses from the previous step and cut with the corresponding formula $D^{\leftarrow}_{i,j,k,\lambda}$ on $c_{i+1,k,\lambda}$ to obtain:
$$I^{\leftarrow}_{i,j,k} \vee [\![e_{i+1,k} \neq \lambda]\!]$$
for all $\lambda \in [n-1]_0$. These can then be cut together on $e_{i+1,k}$ to obtain $I^{\leftarrow}_{i,j,k}$.

Once $I^{\rightarrow}_{0,0,k}$ is derived for all $k \in [n-1]_0$, we are ready to conclude the refutation. We do this by first taking all axioms (6) of the form $[\![s_{0,0} \neq k]\!] \vee [\![p_{2,k} \neq \ell]\!] \vee [\![e_{0,0} \neq \lambda]\!] \vee [\![c_{0,0,\lambda} \neq 0]\!]$ and cutting with the corresponding clauses (2) to obtain:
$$[\![s_{0,0} \neq k]\!] \vee [\![p_{1,k} \neq \ell]\!] \vee [\![e_{0,0} \neq \lambda]\!]$$
for all $k, \ell \neq k, \lambda \in [n-1]_0$. Cutting these together for each $k, \ell \neq k \in [n-1]_0$ on $e_{0,0}$ yields:
$$[\![s_{0,0} \neq k]\!] \vee [\![p_{1,k} \neq \ell]\!]$$
which we can then cut together with $I^{\leftarrow}_{0,0,k}$ for each $k \in [n-1]_0$ on $p_{2,k}$ to obtain $[\![s_{0,0} \neq k]\!]$ for all $k \in [n-1]_0$. We can finally cut these together to obtain $\perp$.

The width of this proof is $O(1)$, as all clauses contain at most 3 blocks of at most $\log n$ literals, and the size is $O(n^5)$, as each step above is applied to n^2 nodes and contributes $O(n^3)$ to the size. $\square$

Corollary 4.22. $\mathsf{C\text{-}EOPL}^{dt} \subseteq \mathsf{SOPL}^{dt}$

Proof. This follows from Theorems 4.19, and 4.21, 4.20. $\square$

4.6 $\mathsf{PLS} \subseteq \mathsf{C\text{-}SOPL}$

We recall the definition of the Sink-of-Dag problem encoded as a CNF.

Definition 4.23. Let $n = 2^k - 1$ be a positive integer. The Sink-of-Dag problem has, for each $i \in [n-1], j \in [n]$ a block of variables $s_{i,j} \in \{0,1\}^k$ encoding a value in $[n]_0$. The clauses of SoD_n are defined as follows, corresponding to the constraints of the SoD search problem:

- **Active Source Node.** $[\![s_{1,1} \neq 0]\!]$,
- **No Proper Sinks.** $[\![s_{i,j} \neq k]\!] \vee [\![s_{i+1,k} \neq 0]\!]$ for each $i, j, k \in [n]$,
- **No Proper Sinks.** $[\![s_{n-1,j} \neq k]\!]$ for each $j, k \in [n]$

We first show that this has an efficient Reversible Blockwise Calculus refutation.

Lemma 4.24. *There is a polynomial-size Reversible Blockwise Calculus refutation of* SoD_n.

Proof. We will prove the following formula for each layer $2 < i < n$:

$$I_i := \bigvee_{j \in [n]} [\![s_{i,j} \neq 0]\!]$$

Once we have derived I_{n-1}, we may with all clauses of the form $[\![s_{n-1,j} \neq k]\!]$ for each j in succession to obtain $\perp$.

We derive I_i as follows for each i. If $i = 2$, this may be obtained by first weakening each clause $[\![s_{1,1} \neq k]\!] \vee [\![s_{2,k} \neq 0]\!]$ by $[\![s_{2,k'} \neq 0]\!]$ for all $k' \neq k$. This yields the clause:

$$[\![s_{1,1} \neq k]\!] \vee I_2$$

for each $k \in [n]$. We can then cut these with the clause $[\![s_{1,1} \neq 0]\!]$ to obtain I_2.

For $i > 2$, we take all clauses of the form $[\![s_{i-1,1} \neq k]\!] \vee [\![s_i i, k \neq 0]\!]$ and weaken as before to obtain $[\![s_{i-1,1} \neq k]\!] \vee C_i$ for each $k \in [n]$. We may then cut this with $C_{i-1} = \bigvee_{1 \leq j \leq n} [\![s_{i-1,j} \neq 0]\!]$ on $s_{i-1,1}$ to obtain:

$$\bigvee_{2 \leq j \leq n} [\![s_{i-1,j} \neq 0]\!] \vee C_i$$

We then likewise weaken all clauses $[\![s_{i-1,2} \neq k]\!] \vee [\![s_i i, k \neq 0]\!]$ and cut with this result to obtain:

$$\bigvee_{3 \leq j \leq n} [\![s_{i-1,j} \neq 0]\!] \vee C_i$$

Continuing this for each remaining set of clauses $[\![s_{i-1,j} \neq k]\!] \vee [\![s_i i, k \neq 0]\!]$ with $3 \leq j \leq n$, we cut away the remaining blocks of C_{i-1} to leave us with C_i as desired.

□

By Theorems 3.6 and 4.6, the desired containment now follows:

Theorem 4.25. $\mathsf{PLS}^{dt} \subseteq \mathsf{C\text{-}SOPL}^{dt}$

4.7 Coloured Intersection: $\mathsf{C\text{-}PLS} \cap \mathsf{C\text{-}PPADS} = \mathsf{C\text{-}SOPL}$

We finally prove the coloured intersection theorem for sink classes. Our proof is inspired by the proof of the uncoloured intersection theorem $\mathsf{SOPL}^{dt} = \mathsf{PLS}^{dt} \cap \mathsf{PPADS}^{dt}$ [GHJ+22a], but the colours introduce an extra complication.

Theorem 4.26. $\mathsf{C\text{-}SOPL}^{dt} = \mathsf{C\text{-}PLS}^{dt} \cap \mathsf{C\text{-}PPADS}^{dt}$

To prove this, since $\mathsf{C\text{-}SOPL}^{dt} \subseteq \mathsf{C\text{-}PPADS}^{dt}$ and $\mathsf{C\text{-}SOPL}^{dt} \subseteq \mathsf{C\text{-}PLS}^{dt}$, it suffices to show the following:

Theorem 4.27. *Let S be a total query search problem. If there is both a depth-d C-SoL$_n$ and C-SoD-formulation of S, then there is also a depth-$2d$ C-SoPL$_{n^2}$-formulation of S.*

Rather than work directly with C-SoL, we will instead interpret it as an instance of a the *injective coloured pigeonhole* problem INJ-C-PHP, which is defined identically but now each node is instead interpreted as an index of both a hole and pigeon – the successor pointers map pigeons to holes, the predecessor pointers map holes to pigeons, and as before edges are added when these pointers agree. Pigeons without an outgoing edge are then required to have a colour while the distinguished pigeon may not have any colours and the corresponding hole may not be mapped to, so as in the usual interpretation of C-SoL a solution is guaranteed by the fact that the component attached to the distinguished source must contain an unmatched pigeon.

We could imagine using copies of formulations of this problem to "merge" paths in a larger formulation, but this only allows us to merge two paths at a time. To make this more useful we will massage this problem even further into something which can merge a larger number of paths, allowing us to resolve one of the key incompatibilities between C-SoD and C-SoPL:

Definition 4.28. PATH-INJ-C-PHP^n_m (where $n > m$) is a problem defined on an $n \times n$ grid, where each node (i, j) has the following parameters:

- a successor pointer $s_{i,j} \in [n] \cup 0$, indicating that the successor is $(i + 1, s_{i,j})$ if $s_{i,j} \neq 0$, and that there is no successor otherwise.
- a predecessor pointer $p_{i,j} \in [n] \cup 0$, indicating that the predecessor is $(i - 1, p_{i,j})$ if $p_{i,j} \neq 0$ and that there is no predecessor otherwise.
- for each $\lambda \in [n]$, an indicator $c_{i,j,\lambda}$ determining whether the colour λ is present at (i, j).
- a colour pointer $e_{i,j} \in [n]$ indexing a colour

Intuitively, we will include edges between nodes (i, j) and $(i+1, k)$ whenever $s_{i,j} = k$ and $p_{i+1,k} = j$. We require that all but the first m nodes on the first layer contain no colours, and that if a node has no outgoing edge then the colour it points to is present. Furthermore, only the first m nodes on the final layer may be pointed to. Formally then, a node-colour pair $((i, j), \lambda)$ is a solution if one of the following is true:

- $p_{i,j} = k$, $s_{i-1,k} = j$, $c_{i,j,\lambda} = 1$, and $c_{i-1,k,\lambda} = 0$,
- $i = 1, j > m$, and $c_{i,j,\lambda} = 1$,
- $i = n - 1$ and $s_{i,j} > m$, or
- $i < n$, (i, j) has no outgoing edge, $e_{i,j} = \lambda$, and $c_{i,j,\lambda} = 0$

Lemma 4.29. *Let S be a search problem. If there is a depth-d $\textsc{inj}$-C-PHP$_m$-formulation of S, then for any $n > m$ there is a depth-d $\textsc{Path-inj}$-C-PHP$^n_{m-1}$-formulation of S. Furthermore, the colours in this formulation are identical between all layers.*

Proof. The construction of the $\textsc{Path-inj}$-C-PHP-formulation is very simple. We will simply duplicate the $\textsc{inj}$-C-PHP-formulation on each layer of the grid. We can then feed the additional nodes one-at-a-time into the copies of the distinguished node, merging into $m-1$ paths after $n-(m-1)$ layers.

Formally, we have a grid of $n \times n$ nodes. For each node i of the original formulation, let $s_i^{\textsc{inj}\text{-C-PHP}}$ denote its forward mapping, $p_i^{\textsc{inj}\text{-C-PHP}}$ its backward mapping, $e_i^{\textsc{inj}\text{-C-PHP}}$ its guaranteed colour, and for each $\lambda \in [n]$, let $c_{i,\lambda}^{\textsc{inj}\text{-C-PHP}}$ denote the corresponding colour indicator.

We define the parameters as follows for a node (i,j). The colours are copied from the original formulation:

$$c_{i,j,\lambda} = \begin{cases} c_{i,\lambda}^{\textsc{inj}\text{-C-PHP}} & i \leq n \\ 1 & i > n, \lambda = 1, \textit{ and } n-j < i-1 \\ 0 & o/w \end{cases}$$

$$e_{i,j} = \begin{cases} e_i^{\textsc{inj}\text{-C-PHP}} & i \leq n \\ 1 & o/w \end{cases}$$

For the successors and predecessors, we must handle the merging of the extra. We do so as follows:

$$s_{i,j} = \begin{cases} s_i^{\textsc{inj}\text{-C-PHP}} & i \leq n \\ j-1 & i > n \textit{ and } n-j \geq i-1 \\ 0 & o/w \end{cases}$$

$$p_{i,j} = \begin{cases} p_i^{\textsc{inj}\text{-C-PHP}} & i \leq n \\ j+1 & i > n \textit{ and } n-j \geq i-1 \\ 0 & o/w \end{cases}$$

It is easy to verify that any solution formed from the first n nodes of a layer corresponds to a solution of the original formulation, and furthermore, since the remaining nodes all either have a successor with no colours or are deactivated, no additional solutions can arise. All parameters can also clearly be computed by a depth-d decision tree, as they are either constant or require simulating a single decision tree from the original formulation. Thus, this is a valid depth-d $\textsc{Path-inj}$-C-PHP$^n_{m-1}$-formulation of S. □

We are now ready to prove Theorem 4.26.

Proof of Theorem 4.26. Suppose there are depth-d $\textsc{inj}$-C-PHP$_n$ and C-SoD$_n$-formulations of some search problem S. By Lemma 4.29 we may derive a depth-d $\textsc{Path-inj}$-C-PHP$^{n^2}_{n-1}$-formulation of S in which all layers are coloured identically. Intuitively, we will convert the C-SoD-formulation into a C-SoPL-formulation by using the path-pigeonhole formulation to handle multiple predecessors. This process will also simplify the computation of predecessors, as it will inherently "split" the original C-SoD nodes and allow them to point to a single fixed predecessor.

More precisely, we proceed by first replacing each node in the $n \times n$ grid of the C-SoD-formulation with an $n^2 \times n^2$ sub-grid, resulting in a $n^3 \times n^3$ overall. This will be used to plant a copy of the $\textsc{Path-inj}$-C-PHP-formulation in place of u. For convenience, we will use (u,v) to index the copy of the node v of the Path-inj-C-PHP-formulation used to replace u in the C-SoD-formulation – ie. $((i,j),(i',j'))$ indexes the node $((i-1)\cdot n^2+i',(j-1)\cdot n^2+j')$.

Before we define the decision trees computing the new parameters, for any node $u \in [n]\times[n]$ of the C-SoD-formulation, let $s_u^{\text{C-SoD}}$ denote its successor, $e_u^{\text{C-SoD}}$ denote its guaranteed colour, and for each $\lambda \in [n]$, let $c_{u,\lambda}$ denote the corresponding colour indicator. Likewise, for any node $u \in [n^2]\times[n^2]$ of the C-PHP-formulation, let $s_u^{\text{C-PHP}}$ denote its successor, $p_u^{\text{C-PHP}}$ its successor, $e_u^{\text{C-PHP}}$ its guaranteed colour, and for each $\lambda \in [n^2]$, let $c_{u,\lambda}^{\text{C-PHP}}$ denote the corresponding colour indicator. Colours are defined for each node (u,v) by simply taking the union of the colour sets of u and v in their respective original formulations, as follows:

$$c_{u,v} = \begin{cases} 1 & c_u^{\text{C-SoD}} = 1 \ or \ c_v^{\text{C-PHP}} = 1 \\ 0 & o/w \end{cases}$$

Now suppose $u=(i,j)$ and $v=(i',j')$. The remaining parameters will be defined depending on the position of v within the Path-inj-C-PHP-formulation. If $i' < n^2$ or $j' > n$, we define the successor and colour pointer according to the Path-inj-C-PHP-formulation – $s_{u,v}=(u,s_v)$ and $e_{u,v}=e_v$. Otherwise, these are instead defined by the C-SoD-formulation, so in order to ensure each node maps to a distinct node on the next layer we set $s_{u,v}=(s_u,((j-1)\cdot n+j',1))$ and $e_{u,v}=e_u$.

We finally proceed to define the predecessor pointer $p_{u,v}$. If $i'>1$, we define the predecessor according to the Path-inj-C-PHP-formulation and $p_{u,v}=(u,p_v)$. Otherwise, we wish to match the indices chosen by the successor pointers on the previous layer, so divide $[n^2]$ into n consecutive blocks of size n and suppose j' is the k^{th} element of the b^{th} block. Then $p_{u,v}=((i-1,b-1),(n^2,k))$.

Finally, let v_0^* be the distinguished node of the C-SoD-formulation. We conclude the construction by selecting the distinguished source node v^* to be $(v_0^*,(1,n))$

Clearly, all parameters as described above can be computed by depth-$2d$ decision trees, as we need only simulate at most two decision trees of the original formulations to compute each. This formulation also has size n^3, as we only define n^2 colours and have a grid of size n^3. It then remains to show that any solution of this formulation can be mapped back to a solution of one the original formulations.

Suppose then that $((u,v),\lambda)$ forms a solution to the C-SoPL-formulation. Again let $u=(i,j)$ and $v=(i',j')$. We have multiple cases:

- If $(u,v)=v^*$ and $c_{u,v,\lambda}=1$, then u is the distinguished node of the C-SoD-formulation, $j'>n-1$, and either $c_u^{\text{C-SoD}}=1$ or $c_v^{\text{C-PHP}}=1$, so either (u,λ) is a solution to the C-SoD-formulation or (v,λ) is a solution to the Path-inj-C-PHP-formulation.

- If (u,v) has no successor and $c_{u,v,e_{u,v}}=0$, then either u has no successor in the C-SoD-formulation and $c^{\text{C-SoD}}_{u,e_u^{\text{C-SoD}}}=0$, in which case $(u,e_u^{\text{C-SoD}})$ forms a solution, or $i'<n$, v has no successor in the Path-inj-C-PHP-formulation, and $c^{\text{C-PHP}}_{v,e_v^{\text{C-PHP}}}=0$, so $(v,e_v^{\text{C-PHP}})$ is a solution of the Path-inj-C-PHP-formulation.

- If (u,v) is a solution because $c_{u,v,\lambda}=1$ and $c_{p_{u,v},\lambda}=0$, we have further cases:
  - If $i'>1$ then observe that all colours from the C-SoD-formulation remain constant between (u,v) and its predecessor. Thus, we must have that $c^{\text{C-PHP}}_{v,\lambda}=1$ while $c^{\text{C-PHP}}_{p_v^{\text{C-PHP}},\lambda}=0$, and (v,λ) forms a solution to the Path-inj-C-PHP-formulation.

– Otherwise, $i' = 1$ and the colouring violation occurred between copies of the Path-inj-C-PHP-formulation. But now notice that by our construction, if $j' <= n$, all colours originating from the Path-inj-C-PHP-formulation are constant between (u, v) and its predecessor, and if $j' > n$, v contains no colours in the Path-inj-C-PHP-formulation. Thus, $c^{\text{C-SoD}}_{u,\lambda} = 1$ and $c^{\text{C-SoD}}_{u,\lambda} = 0$, so (u, λ) is a solution to the C-SoD-formulation

This covers all possible solutions, so this is indeed a valid C-SoPL-formulation of S. □

5 Separation Results

In this section, we will show our results by separations from PLS^{dt}, using resolution width. As such, we begin by defining a useful tool for bounding resolution width:

Definition 5.1. [AD03, dRGN+21] Let F be a CNF formula on n variables $x_1, ..., x_n$. The *Prover-Delayer* game on F is a game played in multiple rounds between two players, a Prover and a Delayer. In each round of the game, the state of the game is captured by a partial assignment $\rho \in \{0, 1, *\}^n$ to the variables of F. The game begins with $\rho = *^n$, and in each round the Prover may update ρ by one of the following two moves:

- **Querying:** The prover may choose to query some variable x_i of F with $\rho_i = *$. The Delayer selects some value $b \in \{0, 1\}$, and ρ is updated by setting $\rho_i \leftarrow b$.
- **Forgetting:** The prover may choose to forget some variable x_i with $\rho_i \neq *$, at which point ρ is updated by setting $\rho_i \leftarrow *$.

The Prover wins once ρ falsifies some clause of F. A Prover strategy is a function of the current game state, and a Delayer strategy is a function of the current game state and a selected move, each of which induce a tree of states which are reachable under that strategy. The width of such a Prover strategy is then the maximum number of non-$*$ entries of ρ over all nodes in the corresponding strategy tree, while the width of a Delayer strategy is instead the minimum width required by any prover strategy playing against it. The resolution width of F is exactly the minimum width of all Prover strategies for F, or the maximum width of all Delayer strategies for F.

5.1 $\mathsf{PLS}^{dt} \neq \mathsf{C\text{-}SOPL}^{dt}$

Theorem 5.2. $\mathsf{PLS}^{dt} \neq \mathsf{C\text{-}SOPL}^{dt}$

Proof. By Theorem 4.20 and Lemma 4.8, it suffices to show the following:
Claim. C-SoPL$_n$ requires width-$\Omega(n)$ to refute in Resolution.

Proof of Claim. We give a delayer strategy for C-SoPL$_n$ guaranteeing high width. Here we use the boolean encoding of C-SoPL$_n$. Let ρ denote the current state of the game. To handle queries to variables encoded by more than one coordinate, we maintain a superset ρ^* of ρ throughout the game. We say that ρ^* encodes a path P if the values of s_u and p_u for each $u \in P$ agree with their assignments under ρ^*. Let P_0 denote the maximal such path starting from the distinguished source v^*, and denote the final node of P_0 by v. A new path P is then obtained from P_0 by selecting a successor for v for which no parameter is defined by ρ^*, and which does not lie in the final column. If no choice of P with the desired properties remains, the Delayer may simply respond arbitrarily, as exhausting all possibilities requires width $\Omega(n)$. Otherwise, we update ρ^* as follows:

- If the prover queries a bit which is already assigned by ρ^*, do nothing.
- If the prover queries a bit of s_u for some u, update ρ^* with its successor in P if $u \in P$ and otherwise point to the last column. Likewise, if the prover queries a bit of p_u for some $u \in P$, answer according to P. Otherwise, point it to the last column.
- If the prover queries $c_{u,\lambda}$ for some u, λ, update it to 0 in ρ^* unless:
  - λ is the last unassigned colour at u
  - $u \notin P$ and e_u is assigned λ under ρ^*
  - $u \notin P$ and either $c_{w,\lambda} = 1$ or e_w is assigned λ for some descendant w of u under ρ^*, or
  - $u \in P$, u is on the final layer n, and e_u is assigned λ by ρ^*.

  In these cases, we update $c_{u,\lambda}$ to 1.
- If a bit of e_u is queried for some u, return any λ such that $c_{u,\lambda}$ and $c_{w,\lambda}$ for every ancestor w of u are not assigned under ρ^*.
- If all bits of a variable are forgotten by the prover, reset all corresponding bits of ρ^* to $*$.

We now simply respond to the query according to ρ^*.

We claim that after concluding any round of the game, ρ^*, and thus ρ, will not falsify any clause of C-SoPL$_n$ unless its width exceeds $o(n)$. For the first round this is clear – the only clauses which could be falisified by a single query are the clauses enforcing that no colour is present at v^*. However, since this is the first round the prover will always respond to such queries with 0.

For later rounds, assume that the claim held for all previous rounds, so any falsified clauses must involve a newly-queried variable. Assuming the width of ρ^* is at most $o(n)$, there exists some path P as specified above, as in order to exhaust all possibilities ρ^* must assign at least one parameter to n nodes on some layer. Thus, the delayer responded as above. The colours and colour pointers returned by the strategy are specifically constructed to avoid contradicting any clauses unless $c_{v^*,\lambda}$ is assigned for some λ and $c_{u,\lambda} = 1$ under ρ^* for some descendant u of v^*. However, this can only occur if a final-layer node is present in P, in which case ρ^* must have assigned the predecessor and successor pointers for the preceding $\Omega(n)$ nodes, contradicting the width requirement. Thus, this cannot occur.

If a successor or predecessor for some $u \notin P$ was queried, no contradiction can arise, as the colours are chosen such that such a node will simply become inactive. If those parameters were queried for some $u \in P$, again no contradiction can arise – by construction the predecessors and successors will agree along P, and again, a final-layer node can only be present in P if we have exceeded the width requirement, so no colours can be present along P to cause a contradiction. □

5.2 $\mathsf{C\text{-}PLS}^{dt} \cap \mathsf{C\text{-}PPAD}^{dt} \neq \mathsf{C\text{-}EOPL}^{dt}$

Recall that $\mathsf{C\text{-}EOPL}^{dt} \subseteq \mathsf{PLS}^{dt}$. We will show that $\mathsf{C\text{-}PLS}^{dt} \cap \mathsf{C\text{-}PPAD}^{dt} \not\subseteq \mathsf{PLS}^{dt}$, and thus cannot be equal to $\mathsf{C\text{-}EOPL}^{dt}$. To do this, we first prove a general statement about Delayer strategies:

Theorem 5.3. *Let $A(x)$ and $B(y)$ be unsatisfiable CNF formulas over disjoint sets of variables. If there is a Delayer strategy for $A(x)$ of width w_A and a Delayer strategy for $B(y)$ of width w_B, then there is a Delayer strategy for $A(x) \wedge B(y)$ of width $\min(w_A, w_B)$.*

Proof. The combined strategy is simple: If the Prover queries a variable of x, respond according to the Delayer strategy for $A(x)$. Otherwise, respond according to the strategy for $B(y)$.

We prove by induction that no Prover can falsify a clause of $A(x) \wedge B(y)$ against this Delayer without reaching the required width. In the first round, this is clear, as width at least 1 is required to falsify any clause.

For later rounds, assume the claim held for all previous rounds. Thus, any falsified clauses under the current assignment ρ must be newly-falsified by the last variable queried. Assume wlog. this was some variable of x and suppose for contradiction that the width of ρ is less than $\min(w_A, w_B) \leq w_A$. Then less than w_A variables of x are assigned by ρ. But each of these were assigned according to the strategy for $A(x)$, so that strategy has width less than w_A, a contradiction. $\square$

We will also require the following separation between PLS^{dt} and PPADS^{dt}:

Theorem 5.4. *[GHJ+22a]* $\mathsf{PPADS}^{dt} \not\subseteq \mathsf{PLS}^{dt}$

Theorem 5.5. $\mathsf{C\text{-}SOPL}^{dt} \cap \mathsf{C\text{-}PPAD}^{dt} \not\subset \mathsf{PLS}^{dt}$

Proof. By Theorems 5.2 and 5.4, and since $\mathsf{C\text{-}PPAD} = \mathsf{PPADS}$ by Theorem 4.15, there are delayer strategies for both C-SoPL_n and C-EoL_n of width $\Omega(n)$. Thus, by Theorem 5.3, there is a delayer strategy for $\text{C-SoPL}_n \wedge \text{C-EoL}_n$ of width $\Omega(n)$, so by Theorem 4.20 $S(\text{C-SoPL}_n \wedge \text{C-EoL}_n) \notin \mathsf{PLS}^{dt}$. $\square$

Corollary 5.6. $\mathsf{C\text{-}EOPL}^{dt} \subsetneq \mathsf{C\text{-}SOPL}^{dt} \cap \mathsf{C\text{-}PPAD}^{dt}$

Proof. This follows as discussed from the previous theorem and Theorem 4.21. $\square$

References

[AB04] Albert Atserias and Maria Luisa Bonet. On the automatizability of resolution and related propositional proof systems. *Inf. Comput.*, 189(2):182–201, 2004.

[AD03] Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 239–247. IEEE Computer Society, 2003.

[AL19] 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.

[BB10] 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.

[BB22] 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.

[BCE+98] 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.

[BFI22] Sam Buss, Noah Fleming, and Russell Impagliazzo. Tfnp characterizations of proof systems and monotone circuits. *Electron. Colloquium Comput. Complex.*, TR22-141, 2022.

[BIK$^+$94] 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.

[BJ12] 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.

[BKT14] Samuel Buss, Leszek Aleksander Kołodziejczyk, and Neil Thapen. Fragments of approximate counting. *The Journal of Symbolic Logic*, 79(2):496–525, 2014.

[BLM07] María Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 171(8-9):606–618, 2007.

[BM04] 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.

[CDDT09] 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.

[CPY17] Xi Chen, Dimitris Paparas, and Mihalis Yannakakis. The complexity of non-monotone markets. *Journal of the ACM*, 64(3):20:1–20:56, 2017.

[CSVY08] 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.

[DGP09] Constantinos Daskalakis, Paul Goldberg, and Christos Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[DMR09] 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.

[dRGN$^+$21] Susanna F. de Rezende, Mika Göös, Jakob Nordström, Toniann Pitassi, Robert Robere, and Dmitry Sokolov. Automating algebraic proof systems is np-hard. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 209–222. ACM, 2021.

[FGHS21] John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: CLS = PPAD ∩ PLS. In *Proceedings of the 53rd Symposium on Theory of Computing (STOC)*, pages 46–59, 2021.

[FMSV20] 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.

[Gar20] Michal Garlík. Failure of feasible disjunction property for k-dnf resolution and np-hardness of automating it. *CoRR*, abs/2003.10230, 2020.

[GHJ$^+$22a] 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.

[GHJ$^+$22b] 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.

[GKRS18] 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.

[JPY88] David Johnson, Christos Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988.

[Kam20] Pritish Kamath. *Some hardness escalation results in computational complexity theory.* PhD thesis, Massachusetts Institute of Technology, 2020.

[Kra01] Jan Krajíček. On the weak pigeonhole principle. *Fundamenta Mathematicae*, 170(1-3):123–140, 2001.

[KST07] Jan Krajícek, Alan Skelley, and Neil Thapen. NP search problems in low fragments of bounded arithmetic. *J. Symb. Log.*, 72(2):649–672, 2007.

[LHdG08] Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, 172(2-3):204–233, 2008.

[MP91] Nimrod Megiddo and Christos Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991.

[Pap94] 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.

[SA94] 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, jul 1994.

[SBI04] 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.

[ST11] Alan Skelley and Neil Thapen. The provably total search problems of bounded arithmetic. *Proceedings of the London Mathematical Society*, 103(1):106–138, 2011.

[Tha16] Neil Thapen. A tradeoff between length and width in resolution. *Theory Comput.*, 12(1):1–14, 2016.